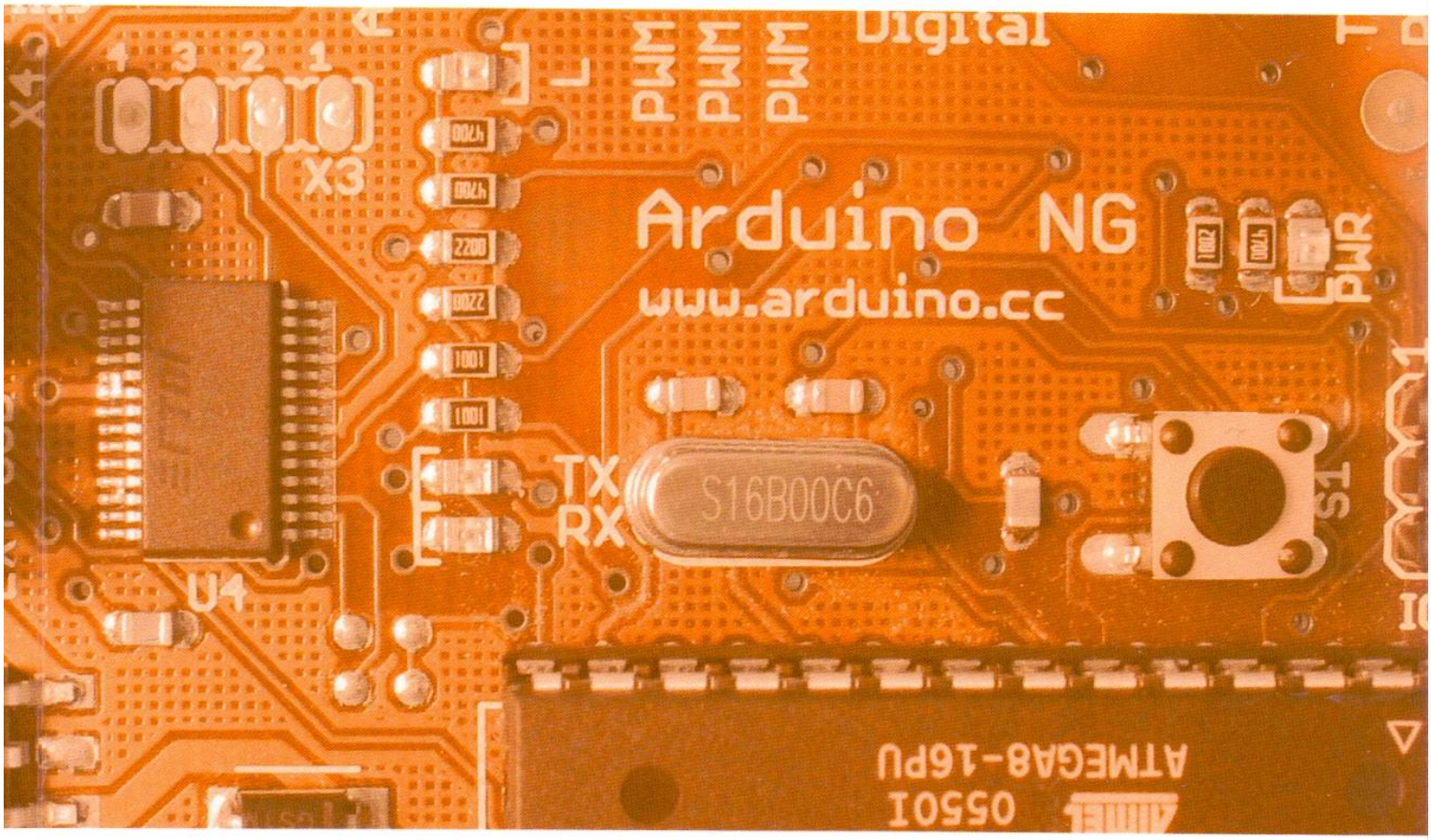
ARDUINO

Guía práctica



Byron O. Ganazhapa

Alfaomega

libros

ARDUINO

GUÍA PRÁCTICA

ARDUINO

GUÍA PRÁCTICA

Byron O. Ganazhapa

Alfaomega

libros

Diseño de colección, cubierta
y pre-impresión: Grupo RC

Datos catalográficos

Ganazhapa, Byron O.
Arduino. Guía práctica
Primera Edición
Alfaomega Grupo Editor, S.A. de C.V., México

ISBN: 978-607-622-698-8
Formato: 17 x 23 cm Páginas: 228

Arduino. Guía práctica
Byron O. Ganazhapa Jiménez
ISBN: 978-84-943055-9-7 edición original publicada por RC Libros, Madrid, España.

Primera edición: Alfaomega Grupo Editor, México, junio 2016

Dr. Isidoro Olvera (Eje 2 sur) No. 74, Col. Doctores, 06720, Ciudad de México.

Miembro de la Cámara Nacional de la Industria Editorial Mexicana
Registro No. 2317

Pág. Web: **http://www.alfaomega.com.mx**
E-mail: **atencionalcliente@alfaomega.com.mx**

ISBN: 978-607-622-698-8

Nota importante:
La información contenida en esta obra tiene un fin exclusivamente didáctico y, por lo tanto, no está previsto su aprovechamiento a nivel profesional o industrial. Las indicaciones técnicas y programas incluidos, han sido elaborados con gran cuidado por el autor y reproducidos bajo estrictas normas de control. ALFAOMEGA GRUPO EDITOR, S.A. de C.V. no será jurídicamente responsable por: errores u omisiones; daños y perjuicios que se pudieran atribuir al uso de la información comprendida en este libro, ni por la utilización indebida que pudiera dársele.

Edición autorizada para venta en México y todo el continente americano.

Impreso en México. Printed in Mexico.

Empresas del grupo:

México: Alfaomega Grupo Editor, S.A. de C.V. – Dr. Isidoro Olvera (Eje 2 sur) No. 74, Col. Doctores, C.P. 06720, Del. Cuauhtémoc, Ciudad de México. Tel.: (52-55) 5575-5022 – Fax: (52-55) 5575-2420 / 2490. Sin costo: 01-800-020-4396 E-mail: atencionalcliente@alfaomega.com.mx

Colombia: Alfaomega Colombiana S.A. – Calle 62 No. 20-46, Barrio San Luis, Bogotá, Colombia, Tels.: (57-1) 746 0102 / 210 0415 – E-mail: cliente@alfaomega.com.co

Chile: Alfaomega Grupo Editor, S.A. – Av. Providencia 1443. Oficina 24, Santiago, Chile Tel.: (56-2) 2235-4248 – Fax: (56-2) 2235-5786 – E-mail: agechile@alfaomega.cl

Argentina: Alfaomega Grupo Editor Argentino, S.A. – Paraguay 1307 P.B. Of. 11, C.P. 1057, Buenos Aires, Argentina, – Tel./Fax: (54-11) 4811-0887 y 4811 7183 – E-mail: ventas@alfaomegaeditor.com.ar

ÍNDICE

PREFACIO

Bienvenidos al emocionante mundo de Arduino como plataforma libre en hardware y software. Esta plataforma, a través de este material, fue diseñada principalmente para ayudar en la formación de estudiantes en programación C/C++, computación, electrónica básica e ingeniería aplicada.

Arduino nos ha permitido una gran facilidad en el momento de desarrollar aplicaciones útiles con el fin de solucionar problemas comunes de una forma sencilla, barata, eficiente y al alcance de cualquier diseñador o desarrollador. Arduino lo hace todo más fácil que cualquier plataforma o entorno electrónico conocido. Cuando teníamos la necesidad de encender un LED, era necesario estudiar toda una teoría muy compleja en electrónica y programación para lograr el objetivo, pero Arduino lo hace en pocos minutos sin necesidad de tener grandes conocimientos en electrónica y programación. En realidad, no es necesario disponer de amplios conocimientos para desarrollar proyectos, solo necesitas leer este libro paso a paso y construir tus propios proyectos.

En este libro encontrarás una teoría para entender la funcionalidad de la plataforma Arduino en hardware y en software, la descripción de componentes básicos necesarios y los conceptos de electrónica básica para el desarrollo de prácticas.

Con su lectura, no solo aprenderás cómo usar componentes electrónicos y armarlos, o el uso del entorno Arduino, sino sabrás cómo escribir el código fuente necesario para dar vida a tus propios proyectos.

En definitiva, este libro es una guía práctica y apropiada para la formación de estudiantes desde niveles de educación secundaria hasta educación superior.

AGRADECIMIENTOS

Todos mis agradecimientos a familiares y amigos quienes me inspiran en seguir adelante en mi carrera profesional y darme todo su apoyo para continuar desarrollando proyectos para la enseñanza en electrónica y programación.

Agradezco a cada uno de mis compañeros de trabajo, quienes han sido parte de este proyecto de formación y quienes me han enseñado a crecer personal y profesionalmente.

ACERCA DEL AUTOR

Byron O. Ganazhapa es ingeniero en electrónica y telecomunicaciones, y programador desde 2008. Especialista en programación industrial, C/C++, y desarrollo de proyectos electrónicos. Ha sido un desarrollador activo de proyectos en Arduino desde 2011 y en la actualidad desempeña el cargo de desarrollador de aplicaciones automatizadas y de control.

INTRODUCCIÓN

La facilidad de explorar el entorno Arduino es enorme debido a las tecnologías de hoy en día, teniendo una gran variedad de compiladores basados en gráficos y texto, y para dispositivos móviles y portátiles gracias a las tecnologías pos-PC.

Este libro abarca temas fundamentales en hardware y software libre "Open Source" de la plataforma Arduino. La electrónica y la programación son temas que se analizarán conforme se desarrolla cada práctica del libro y no es necesario tener conceptos amplios para el avance y la continuidad de proyectos en Arduino. En el primer capítulo se describe el entorno Arduino tanto en hardware como en software. El segundo capítulo detalla el uso de la plataforma: estructura de programación, funciones, variables, operaciones, sentencias condicionales, etc. El tercer capítulo ayuda a extender el uso de entornos online como, por ejemplo, Codebender. El cuarto capítulo nos muestra cómo construir proyectos con el software fritzing para la simulación de montajes electrónicos semejantes a la realidad y cómo usar el software para preparar circuitos en protoboards. En el quinto capítulo se describe una gran variedad de dispositivos y materiales necesarios para el desarrollo de prácticas; el último capítulo ayuda a desarrollarlas con detalle mostrando el uso de componentes electrónicos necesarios para el montaje de circuitos, así como la descripción y el desarrollo de códigos de programación.

Todo se explica en pasos claros y fáciles de seguir. El texto contiene una gran cantidad de diagramas y fotografías para que sea lo más fácil posible desarrollar cada capítulo y las prácticas correctamente.

1 ENTORNO ARDUINO

¿QUÉ ES ARDUINO?

Arduino es una plataforma de hardware de código abierto, basada en una sencilla placa con entradas y salidas analógicas y digitales, en un entorno de desarrollo que está basado en el lenguaje de programación Processing. Es un dispositivo que conecta el mundo físico con el mundo virtual, o el mundo analógico con el digital.

Sus creadores son el zaragozano David Cuartielles, ingeniero electrónico y docente de la Universidad de Mälmo, Suecia, y Massimo Banzi, italiano, diseñador y desarrollador web. El proyecto fue concebido en Italia en el año 2005.

Arduino puede tomar información del entorno físico a través de sus puertos de entrada; para ello, toda una gama de sensores se pueden usar para el control de luces, motores, pantallas y otros actuadores, creando una interfaz de comunicación de un sistema a otro. El microcontrolador en la placa Arduino se programa mediante el lenguaje de programación Arduino (basado en Wiring) y el entorno de desarrollo Arduino (basado en Processing). Los proyectos realizados con Arduino pueden ejecutarse sin necesidad de conectarlo a un ordenador, si bien tienen la posibilidad de hacerlo y comunicar con diferentes tipos de software (por ejemplo, Flash, Processing, MaxMSP).

Las placas pueden ser construidas a mano o comprarse montadas de fábrica; el software se puede descargar de forma gratuita. Los ficheros de diseño de referencia (CAD) están disponibles bajo una licencia abierta; así pues, eres libre de adaptarlos a tus necesidades.

www.arduino.cc

¿CÓMO USAR ARDUINO?

Para usar el entorno Arduino, se empieza a configurar el hardware, por ejemplo: configurar puertos de entrada y salida, puertos analógicos, comunicaciones seriales, I2C, SPI, etc. Explicaremos cómo instalar el software Arduino en un ordenador que ejecute cualquiera de los siguientes sistemas operativos: Windows, MAC OS X, GNU/Linux. Explicaremos Arduino IDE y cómo usarlo antes de empezar a desarrollar algún proyecto, desde el más básico hasta el más avanzado. Cada proyecto comenzará con una descripción de cómo configurar el hardware y qué código es necesario para que funcione. A continuación, se describirá la funcionalidad del hardware y del software por separado.

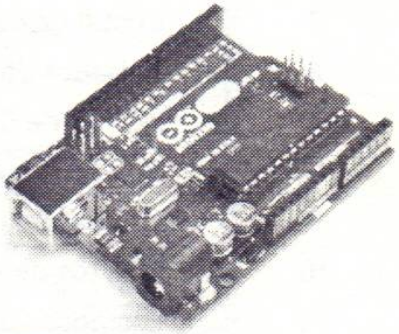
PLATAFORMA LIBRE

Hardware libre. Se llama hardware libre, electrónica libre o máquinas libres a aquellos dispositivos de hardware cuyas especificaciones y diagramas esquemáticos son de acceso público, ya sea bajo algún tipo de pago o de forma gratuita. La filosofía del software libre es aplicable a la del hardware libre y por eso forma parte de la cultura libre. Un ejemplo de hardware libre es la arquitectura UltraSparc cuyas especificaciones están disponibles bajo una licencia libre.

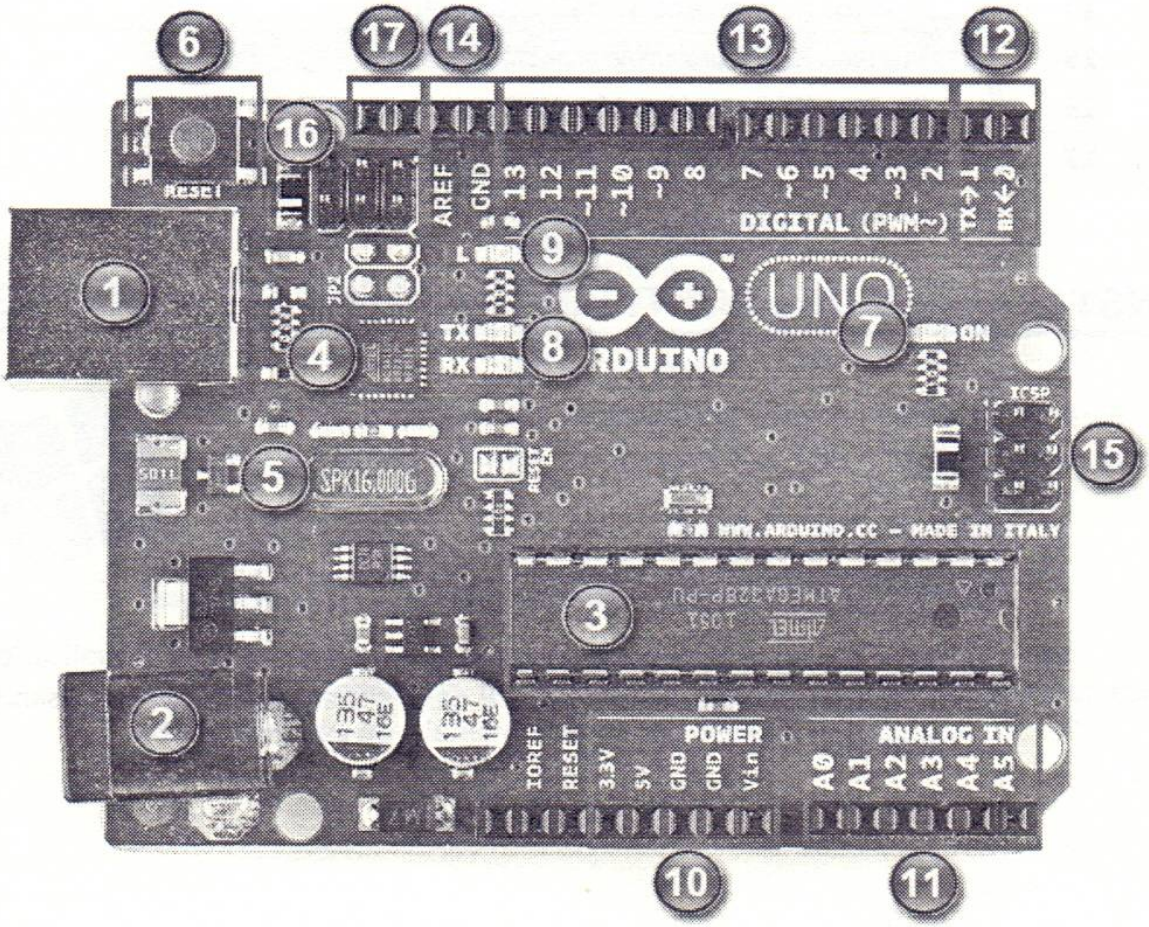
Software libre. (Tambien llamado Free Software) Es la denominación del software que respeta la libertad de todos los usuarios que adquirieron el producto y, por tanto, una vez obtenido el mismo, puede ser usado, copiado, estudiado, modificado y redistribuido libremente de varias formas. Según Free Software Foundation, el software libre se refiere a la

seguridad de los usuarios para ejecutar, copiar, distribuir y estudiar el software, e incluso modificarlo y distribuirlo.

PLACA ARDUINO UNO Y SUS PARTES

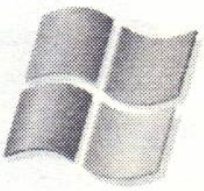


Arduino UNO es una placa electrónica basada en el ATmega328P. Cuenta con 14 puertos digitales de entrada/salida (de los cuales 6 se pueden utilizar como puertos de salida PWM), 6 entradas analógicas, un resonador de 16 MHz, una conexión USB, un conector de alimentación, un encabezado ICSP y un botón de reinicio. Contiene todo lo necesario para desarrollar una infinidad de proyectos prácticos de electrónica; simplemente conectarlo a un ordenador con un cable USB o con un adaptador de corriente continua para empezar.



N°	Descripción de partes
1	USB para el cable tipo AB
2	Conector de voltaje de 9 a 12 voltios
3	Microcontrolador ATmega328P, cerebro de Arduino
4	ATmega16U2 encargado de la comunicación USB con el PC
5	Cristal de cuarzo de 16 Mhz
6	Pulsador de Reset
7	LED verde de placa encendida
8	LED TX (Transmisor) y LED RX (Receptor) de la comunicación serial
9	LED naranja conectado al pin 13
10	Puertos de voltaje 5 voltios, 3,3 voltios y tierra
11	Entradas analógicas
12	Puertos de recepción (RX→0) y transmisión (TX→1) serial
13	Puertos de E/S digitales y PWM
14	Puertos de referencia analógica y tierra
15	Puertos ICSP para programación serial
16	Puertos ICSP para interfaz USB
17	Puertos I2C (SDA, SCL)

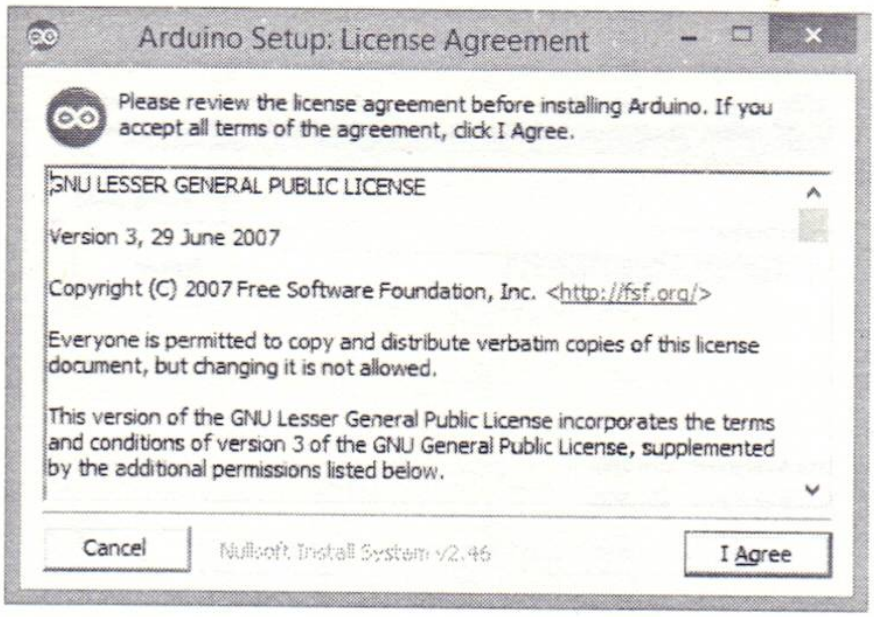
INSTALACIÓN DEL SOFTWARE ARDUINO EN WINDOWS



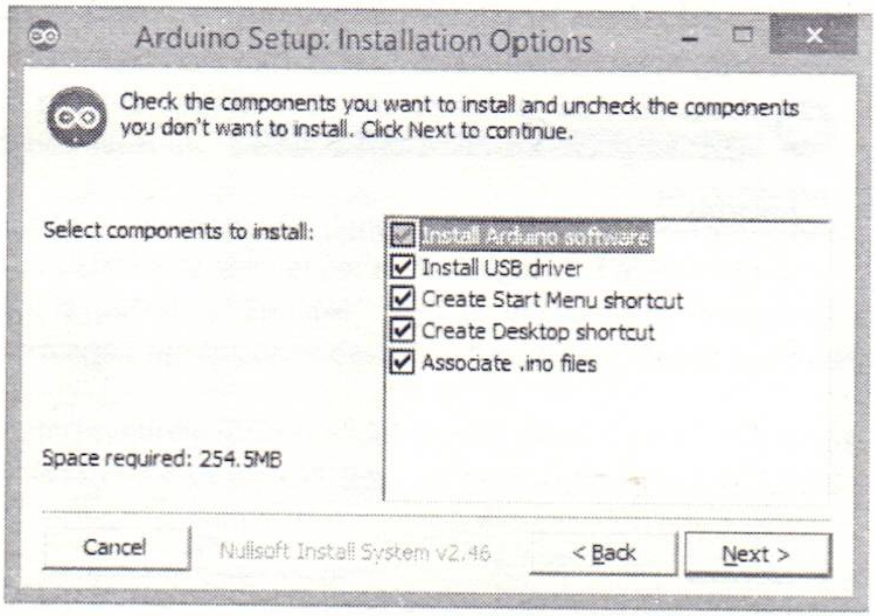
Para un sistema operativo Windows en sus versiones 10, 8, 7 o Vista, debes realizar la siguiente sucesión de sencillos pasos:

1) Descarga el software de su página oficial: www.arduino.cc/en/Main/Software para Windows.

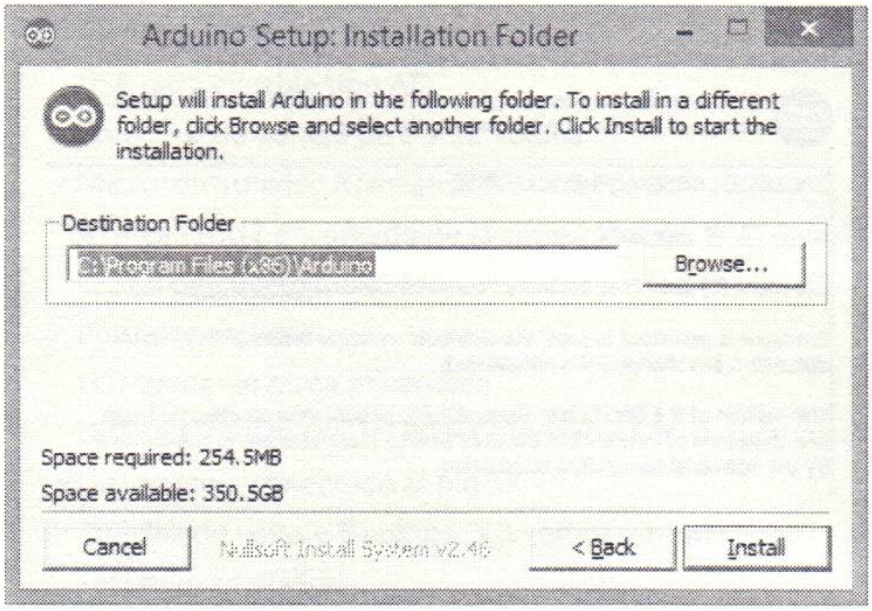
2) Haz doble clic en el instalador de acuerdo a cada versión de actualización para Windows. Acepta los términos y condiciones del software pulsando en "**I Agree**".



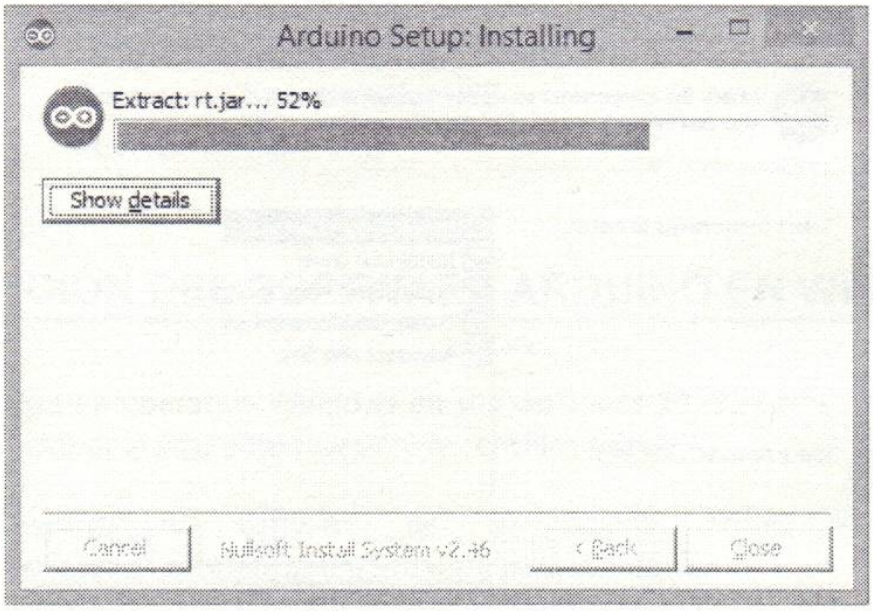
3) Selecciona todos los componentes de instalación y pulsa en "**Next**".



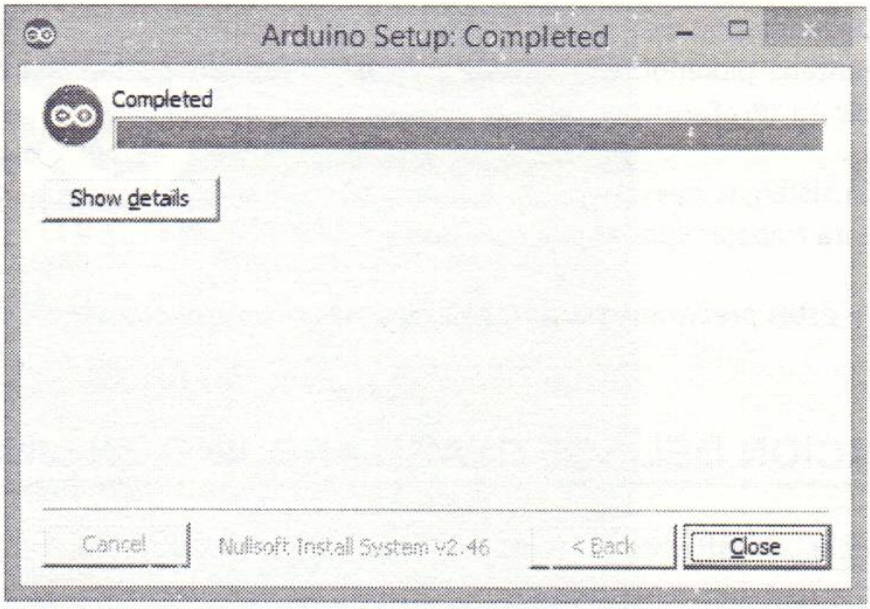
4) Selecciona la dirección de instalación del software en el disco C. Esta opción viene ya por defecto. Haz clic en "**Install**".



5) Espera a que termine la instalación.



6) Por último pulsa en "**Close**" para acabar con la instalación.



INSTALACIÓN DEL SOFTWARE ARDUINO EN MAC OS X

Para este sistema operativo en sus versiones 10.8 o superior, lo único que debes hacer es:

1) Como primera opción tienes que dirigirte a "**Preferencias del sistema**" y abrir el panel de "**Seguridad & Privacidad**". En la pestaña "**General**" bajo el encabezado "**Permitir descargar aplicaciones desde**", y haz clic en "**Desde cualquier lugar**".

2) Como segunda opción y más directa puedes dirigirte a la página oficial en la sección de descargas: www.arduino.cc/en/Main/Software para MAC OS X.

3) Después de completar la descarga, haz doble clic sobre el archivo zip para descomprimirlo.

4) Copia la aplicación Arduino a la carpeta de Aplicaciones, o en cualquier otro sitio donde desees instalar el software.

5) Conecta la placa Arduino UNO al ordenador.

6) Dependiendo de la versión del sistema operativo, podría aparecer una ventana pidiéndote si deseas abrir las "**Preferencias del sistema**". Hacer clic en "**Preferencias de Red**" y después en "**Aplicar**".

7) En sistemas operativos OS X, no es necesario instalar ningún controlador para trabajar con las placas Arduino.

8) Ya estás preparado para trabajar y desarrollar proyectos en Arduino.

INSTALACIÓN DEL SOFTWARE ARDUINO EN LINUX

Para instalar el software Arduino en cualquier distribución de LINUX, lo único que debes hacer es:

1) Instalar algunos programas dependiendo de la distribución de LINUX. Para ello visita la siguiente página web: www.arduino.cc/linux.

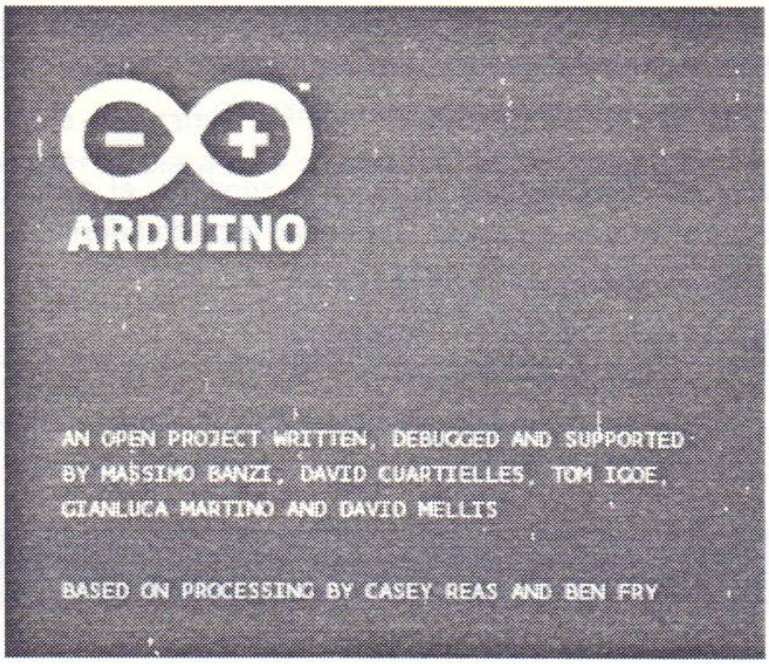
2) Después dirígete a la página oficial de Arduino en la sección de descargas: www.arduino.cc/en/Main/Software para LINUX.

3) Después de completar la descarga. Copia y pega todos los archivos a un directorio (o al escritorio) y ejecuta el script de Arduino. Asegúrate de que todo el directorio extraído "**location/name**" no tenga espacios en el "**/location/name**".

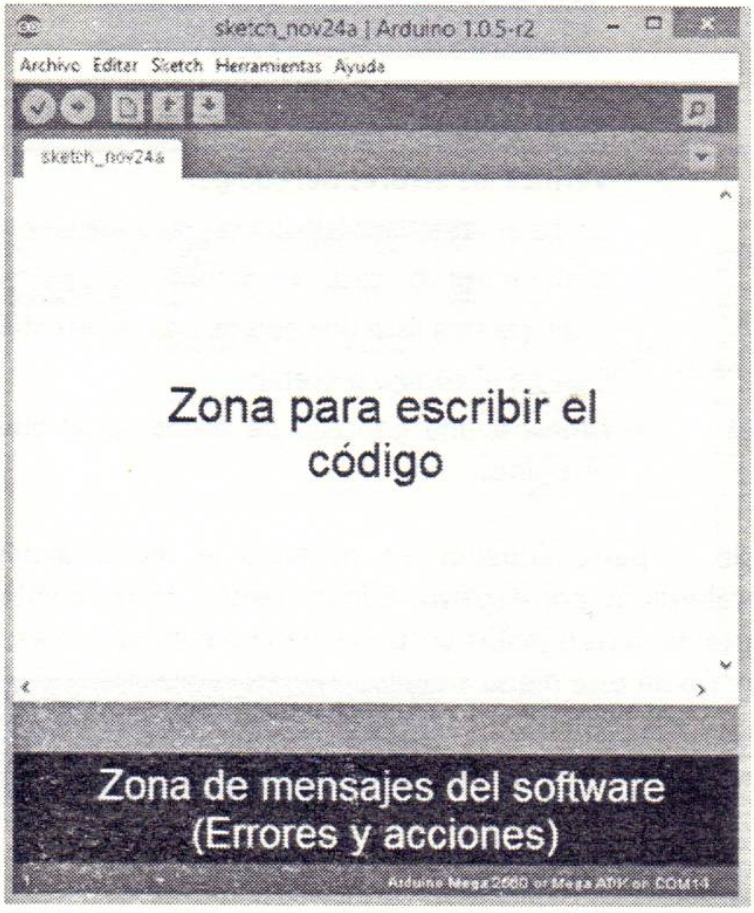
4) Ya estás preparado para trabajar y desarrollar proyectos en Arduino.

CONOCIENDO EL SOFTWARE ARDUINO

Para ejecutar el programa Arduino IDE, vamos a la dirección donde está instalado el software o en el escritorio.



Este es el Arduino IDE (Integrated Development Environment) y es donde vas a escribir sus sketches (programas) para subir a tu placa Arduino.



Cuando se abre el software, este se verá muy similar a la imagen anterior. Si estás utilizando Windows o Linux habrá algunas pequeñas diferencias, pero el IDE es más o menos lo mismo sin importar qué sistema operativo estés utilizando.

El IDE se divide en la barra de herramientas en la parte superior, el código o ventana de sketch en el centro y la ventana de salida de serie en la parte inferior.

Barra de herramientas

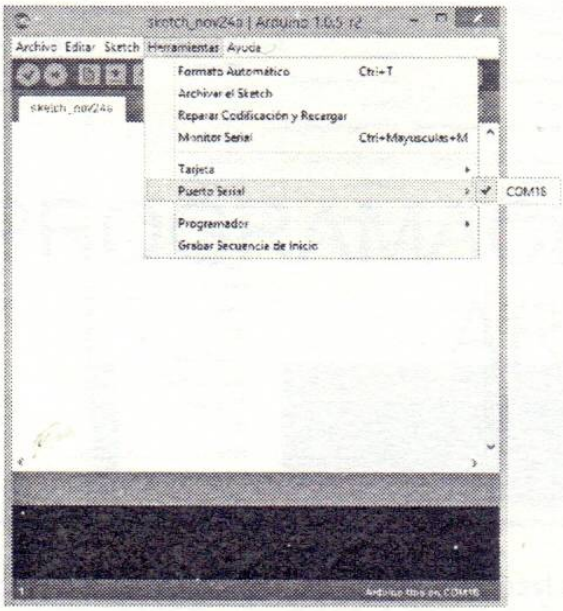
La barra de herramientas consta de 5 botones (Verificar, Cargar, Nuevo, Abrir y Guardar). También hay un botón adicional en el extremo derecho (Monitor Serial).

N°	Descripción de partes
Verificar	Verifica los errores del código.
Cargar	Carga el sketch o código a la placa Arduino.
Nuevo	Crea un nuevo sketch en blanco.
Abrir	Muestra una lista de códigos en tu sketchbook
Guardar	Guarda el código o sketch
Monitor Serial	Muestra una pantalla de datos serial enviados desde el Arduino

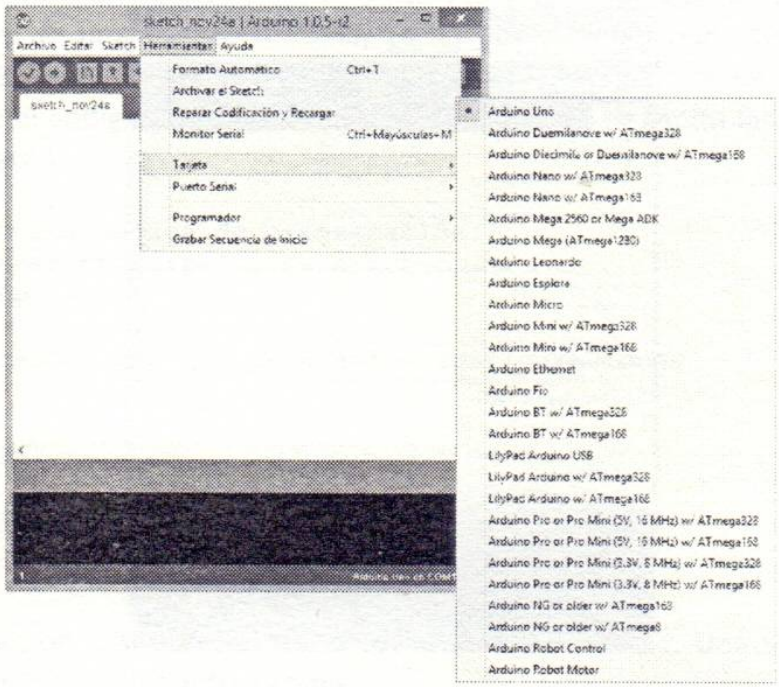
A lo largo de la parte superior se muestra el menú archivo con menús desplegables encabezados por Archivo, Editar, Sketch, Herramientas y Ayuda. Los botones de la barra de herramientas proporcionan un cómodo acceso a las funciones más utilizadas dentro de este menú archivo.

Cargar mi primer ejemplo

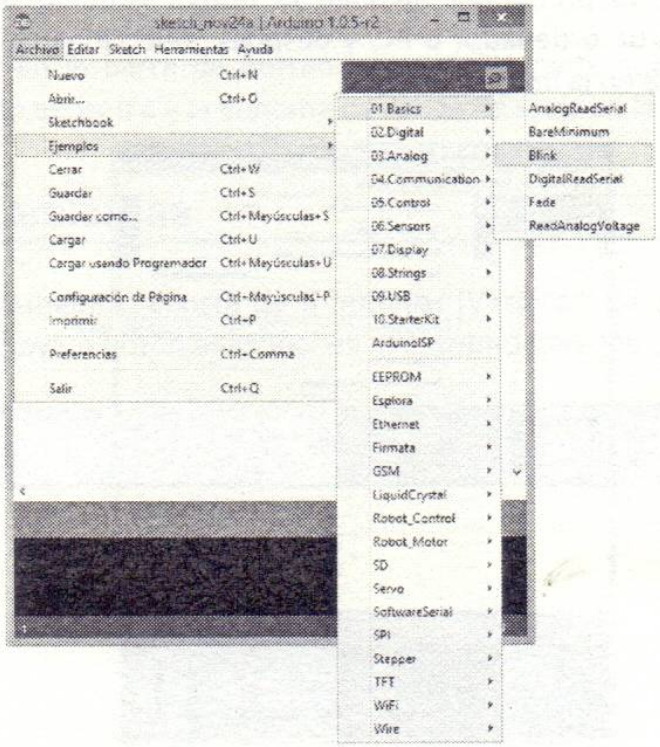
Para programar tu primer ejemplo en la tarjeta Arduino primero tienes que conectar tu placa a un ordenador o PC, y después seleccionar el puerto COM de la placa como se indica en la imagen inferior.



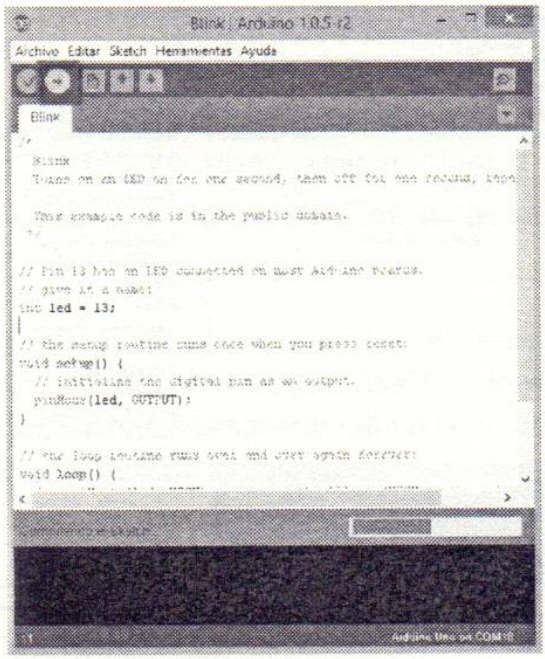
Después de ello, selecciona la placa de desarrollo Arduino UNO o la que dispongas.



Luego selecciona el código de ejemplo ubicado en la barra Archivo y elige "**ejemplos/Basics**" y selecciona el ejemplo código "**Blink**".



Por último, haz clic en la herramienta Cargar para grabar el código a la placa Arduino UNO.



2 PROGRAMACIÓN EN ARDUINO

ESTRUCTURA BÁSICA DE PROGRAMACIÓN

La estructura básica de programación de Arduino es bastante simple y divide la ejecución en dos partes: `setup()` y `loop()`. La función `Setup()` constituye la preparación del programa y `loop()` es la ejecución. En la función `Setup()` se incluye la declaración de variables y se trata de la primera función que se ejecuta en el programa. Esta función se ejecuta una única vez y es empleada para configurar las funcionalidades de una tarjeta Arduino, como por ejemplo para determinar si un puerto digital es de entrada o salida, inicializar la comunicación serial, etc. La función `loop()` incluye el código a ser ejecutado continuamente (leyendo las entradas de la placa, salidas, etc.).

```
/* Estructura básica de programación.
   Funciones principales de Arduino. */
void setup()
{
    // Preparación del programa y configuraciones
}

void loop()
{
    // Ejecución del programa principal
}
```

Toda instrucción o sentencia acaba con punto y coma (;). Una sentencia en su forma más simple es solo punto y coma.

Los comentarios se indican con doble barra oblicua (`//`). Esta provocará que el compilador ignore el texto solo hasta el fin de la línea.

Al igual que la programación en C se pueden introducir bloques de comentarios con (`/* ... */`). La configuración barra oblicua - asterisco (`/*`) es usada para crear un bloque de comentarios. El compilador ignora todo texto después de barra oblicua - asterisco (`/*`), incluso si este engloba múltiples líneas, hasta la configuración asterisco - barra oblicua (`*/`).

Las llaves "`{ }`" son usadas para delinear el principio y el fin del contenido de una función. Las llaves también son usadas para indicar cuándo una serie de sentencias van a ser tratadas como un bloque simple.

FUNCIONES

Una función es un bloque de código que tiene un nombre y un conjunto de instrucciones que son ejecutados cuando se llama a la función. Son funciones **`setup()`** y **`loop()`** de las que ya se ha hablado. Las funciones de usuario pueden ser escritas para realizar tareas repetitivas y para reducir el tamaño de un programa. Las funciones se declaran asociadas a un tipo de valor (p. ej.: el tipo de dato `int`) que devolverá la función un dato numérico tipo entero. Si la función no devuelve ningún valor, entonces se colocará la palabra `void`, que significa "**función vacía**". Después de declarar el tipo de dato que devuelve la función se debe escribir el nombre de la función y entre paréntesis se escribirán, si es necesario, los parámetros que se deben pasar a la función para que se ejecute.

La forma estándar de una función es:

```
Tipo Nombre_Funcion(parametro_1, parametro_2,...)
{
   Sentencia_1;
   Sentencia_2;
   ...
   Sentencia_x;
}
```

El tipo de una función (**`Tipo`**) o sus parámetros (`parametro_1, parametro_2,...`) pueden ser algún tipo de variable válido, por ejemplo: `int`, `char`, `float` o `String`. El tipo `void` es un tipo válido y es usado para indicar un parámetro o valor retornado de tamaño cero. Entonces, las funciones pueden ser declaradas como lo siguiente:

```
void Mi_Funcion_1()
{
   // Función sin retorno
}

int Mi_Funcion_2()
{
   // Función que retorna un valor entero.
}
```

Funciones de retorno

En algunos casos, una función es designada para realizar una tarea y retornar un valor o estado de una tarea realizada. El control de la palabra `return` es usado para indicar un punto de salida en una función, o en el caso de una función tipo de valor no-vacío, para un valor que va a ser retornado durante su ejecución.

En una función tipo **`void`**:

```
void suma(int x, int y)
{
   int z = x + y;  // Esta sentencia siempre se ejecutará.
   return;
   int w = z + 1;  // Esta sentencia nunca será ejecutada.
}
```

La función retornaría solo ejecutando la primera sentencia `int z = x + y`, y la siguiente sentencia después de `return` no se ejecutaría. Por lo tanto, la función retornaría al programa principal sin devolver un valor.

Para llamar a la función `suma` se usa el siguiente programa:

```
void setup()
{
   // No hay código aquí.
}
void loop()
{
   int X = 8; // Ponemos en X el valor de 8.
   int Y = 4; // Ponemos en Y el valor de 4.
   suma(X, Y);/* Sentencia que llama a la función suma() con valores
                 de X = 8, Y = 4, y retorna el valor pero no es
                 usado el cálculo de la suma de dos números. */
}
```

En una función cuyo tipo no es `void`, la palabra de control `return` solo enviará la ejecución de vuelta a la sentencia de llamada de la función. Además, `return` colocará

el valor de la expresión en la sentencia de llamada. La siguiente función es de tipo `float`. Por lo tanto, un `float` será declarado en la sentencia de llamada cuando `return` sea ejecutado:

```
/* Función para el cálculo del área de un círculo */
float area(int radio)
{
   float pi = 3.14159;
   float A = pi * radio * radio;// Formula de Área: A = pi*radio^2
   return A;                    // Retorna el valor A en tipo float.
}
```

La capacidad para retornar un valor permite a una función ser usado como parte de una expresión, tal como una asignación. Por ejemplo, en el siguiente programa:

```
void setup()
{
   // No hay código aquí.
}

void loop()
{
   int R = 2;      // Ponemos en R el valor de 2.
   float A;
   A = area(R);    // Sentencia que llama a la función area()
                      con un valor de R o Radio igual a 2, y
                      asigna su valor de retorno a A. */
}
```

La función `area()` asignará a la variable A el valor calculado del área de un círculo en tipo de dato `float`. Por lo tanto, el resultado debería ser de `12,56636`.

VARIABLES Y CONSTANTES

Una variable debe ser declarada y opcionalmente asignada a un determinado valor antes de su uso. En la declaración de una variable se indica el tipo de dato que almacenará (`boolean, byte, int, float, long, char`).

```
boolean Mi_Variable_Boleana = false;
byte Mi_Variable_byte = 10;
int Mi_Variable_Entero = 1000;
float Mi_Variable_flotante = 3.14;
long Mi_Variable_Larga = 2e8;
char Mi_variable_Caracter = 'A';
```

Una variable puede ser declarada en el inicio del programa antes de `setup()`, localmente a una determinada función e incluso dentro de un bloque como pueda

ser un bucle (p. eje., el bucle `for`). El sitio en el que la variable es declarada determina el ámbito de la misma.

Variables globales

Una variable global es aquella que puede ser empleada en cualquier función del programa. Estas variables deben ser declaradas al inicio del programa (antes de la función `setup()`).

```
int v;      //v es visible en todo el programa. Es variable global.
void setup()
{
   v = 1000; // Se asigna en un principio el valor de 1000.
}
void loop() {
   v = v + 1;//La variable es modificada conforme corre el programa.
}
```

Las variables globales pueden ser localizadas en todas las funciones de un programa, y pueden ser modificadas por cualquier función y retener sus valores para ser usadas en otras funciones.

Variables locales

Una variable local es aquella que puede ser empleada en una sola única función. Estas variables deben ser declaradas dentro de una función o estructuras de control del programa.

```
void setup()
{
  int a = 10;  // Variable visible solo para la función void step().
  for (int i = 0; i < 10; i++) // Estructura de control
  {
     int c = a*i;// Variables (c,i) visibles solo para el bucle for.
  }
}

void loop() {
  for (int i = 20; i < 0;i--)  // Estructura de control
  {
     int f = f+i; // Variables visibles únicamente en el bucle for.
  }
}
```

Estas variables no son accesibles desde otras funciones, lo cual quiere decir que su alcance es limitado para las funciones en las que son declaradas. Una variable local

puede ser usada en múltiples funciones sin conflictos, ya que el compilador ve cada una de esas variables como parte de esa única función.

Constantes

Las constantes son un tipo de variables de valor fijo que no pueden ser modificadas mientras se ejecuta un programa. Podemos clasificar las constantes en grupos:

1) **Constantes booleanas.** Estas constantes definen dos tipos de verdad y falsedad en el lenguaje Arduino: `true` y `false`.

   La constante `false` es la más fácil de las dos para definir. `false` es definido como 0.

   La constante `true` es a menudo definido como 1, lo cual es cierto, pero `true` tiene una amplia definición. Cualquier número entero que es distinto de cero es `true`, en un sentido booleano. Así -1, 2, y -200 son todos definidos como `true`, también, en un sentido booleano.

2) **Niveles lógicos en puertos.** Al leer o escribir en un puerto digital solo hay dos valores posibles: `HIGH` y `LOW`.

   El significado de `HIGH` (en referencia a un puerto) es algo diferente dependiendo de si un puerto se establece como una entrada o salida. Cuando un puerto es configurado como entrada, al momento de leer su valor, Arduino lo reportará como `HIGH` (si es mayor a 3 voltios para placas de 5 voltios). Cuando un puerto es configurado como salida, se escribe `HIGH`, el puerto de salida estará en 5 voltios (para placas de 5 voltios).

   El significado de `LOW` también tiene un significado diferente dependiendo de si un puerto está configurado como entrada o salida. Cuando un puerto es configurado como entrada, en el momento de leer su valor, Arduino lo reportará como `LOW` (si es menor a 3 voltios para placas de 5 voltios). Cuando un puerto es configurado como salida, se escribe `LOW`, el puerto de salida estará en 0 voltios (para placas de 5 voltios).

3) **Modos de puertos digitales.** Los puertos digitales pueden ser usados como: entrada o `INPUT`, entrada en pull-up o `INPUT_PULLUP`, y salida u `OUTPUT`.

Configurar los puertos de Arduino como `INPUT` se dice que están en un estado de alta impedancia, es decir, equivale a colocar una resistencia en serie equivalente a 100 MΩ delante del puerto. Esto los hace útiles para la lectura de un sensor o interruptor.

Los puertos digitales de Arduino tienen resistencias pull-up internas (resistencias que conectan a energía internamente) que se puede acceder a través de `INPUT_PULLUP` para evitar usar resistencias externas. Si una resistencia pull-up es usada, el puerto de entrada estará en `HIGH` cuando un interruptor esté abierto y `LOW` cuando el interruptor esté cerrado.

Los puertos configurados como `OUTPUT` se dice que están en un estado de baja impedancia, esto quiere decir que pueden proporcionar una cantidad de corriente a otros circuitos.

El lenguaje de Arduino maneja otro tipo de constantes llamadas constantes enteras. Estas constantes se usan directamente en un sketch. Por defecto son tratados como `int` pero pueden ser modificadas a través de los formateadores U y L.

1) **Decimal.** Esta constante es de base 10, es decir, usa la matemática de sentido común con la que se está familiarizado. Constantes sin otros prefijos se asumen que están en formato decimal.

```
A = 10;  //igual que 10 decimal.
```

2) **Binaria.** Esta constante es de base 2. Solo los valores 0 y 1 son admitidos y se indican con el prefijo "B". Este formato funciona en 8 bits, es decir, de 0 a 255 valores decimales.

```
A = B10;  //igual que 2 decimal.
```

3) **Octal.** Esta constante es de base 8. Solo los valores 0 a 7 son válidos. Valores octales se indican con el prefijo "O".

```
A = O10;  //igual que 8 decimal.
```

4) **Hexadecimal.** Esta constante es de base 16. Los valores válidos son del 0 al 9 y letras de la A a la F: A tiene el valor de 10, B es 11, C es 12, D es 13, E es 14, y F es 15. los valores hexadecimales se indican con el prefijo "0x".

```
A = 0x10;  //igual que 16 decimal.
B = 0xBA;  //igual que 186 decimal.
```

Los formateadores U y L pueden realizar el cambio de un formato a otro, por ejemplo:

1) Una "u" o "U" fuerza una constante a un formato sin signo.

```
A = 20U;
```

2) Una "l" o "L" fuerza una constante a un formato de longitud.

```
A = 250000L;
```

3) Una "ul" o "UL" fuerza una constante a un formato de longitud sin signo.

```
A = 32767UL;
```

TIPO DE DATOS

Las variables y constantes son almacenadas en memorias de espacio limitado dentro del microcontrolador de la tarjeta Arduino, por lo que es necesario saber cuánto espacio de memoria dejar para cada variable sin desperdiciar memoria innecesariamente. Consecuentemente, un desarrollador debe saber cómo declarar las variables, especificando tanto el tamaño de la variable como el tipo de la variable.

A continuación se presenta una lista de tipo de datos comúnmente utilizados en el entorno Arduino con descripción y tamaño de memoria:

Tipo de dato	Descripción	Rango
`boolean`	Almacena un valor entero de 8 bits. Lógica booleana verdad/falso	`True` y `false`
`byte`	Almacena un valor entero sin signo de 8 bits	0 a 255
`char`	Almacena un valor tipo carácter con signo de 8 bits.	128 a 127
`unsigned char`	Almacena un valor tipo carácter sin signo de 8 bits	0 a 255
`short`	Almacena un valor entero con signo de 16 bits	-32768 a 32767
`int`	Almacena un valor entero con signo de 16 bits	-32768 a 32767

Tipo de dato	Descripción	Rango
`unsigned int`	Almacena un valor entero sin signo de 16 bits	0 a 65535
`word`	Almacena un valor entero sin signo de 16 bits (sola para Arduino Due y Zero 32 bits)	0 a 65535 (0 a 4294967296)
`long`	Almacena un valor entero con signo de 32 bits	-2147483648 a 2147483647
`unsigned long`	Almacena un valor entero sin signo de 32 bits	0 a 4294967296
`float`	Almacena un valor flotante con signo de 32 bits	$-3{,}4028235E^{+38}$ a $3{,}4028235E^{+38}$
`double`	Almacena un valor flotante de doble precisión con signo de 32 bits	$-3{,}4028235E^{+38}$ a $3{,}4028235E^{+38}$

Arrays

Un array (o matriz) es un conjunto de valores que pueden ser accedidos con un número de índice (el primer valor del índice es 0), es decir, cualquier valor puede ser accedido haciendo uso del nombre de la matriz y el número índice.

La estructura general para declarar un array es la siguiente:

```
int Mi_Array[] = {valor_0, valor_1, valor_2,..., valor_N-1};
```

Los valores de un array pueden ser manejados por el índice. El índice puede variar entre 0 y la longitud del array declarado menos 1.

```
Mi_Array[0] = valor_0;
Mi_Array[1] = valor_1;
Mi_Array[2] = valor_2;
...
Mi_Array[N-1] = valor_N-1;
```

Un array puede ser declarado como cualquier otra variable o constante, tipo de dato y tamaño, por ejemplo:

```
int Mi_matriz[5];                    //Variable tipo int de 6 enteros.
int PuertosPWM[] = {3,5,6,9,10,11};//Variable tipo int de 6 enteros.
int Pos_Neg[5] = {-2,-1,0,1,2};      //Variable tipo int de 5 enteros.
char texto1[5] = {'H','O','L','A'};//Variable tipo char de 4 caract.
char texto2[5] = {"HOLA"};           //Variable tipo char de 4 caract.
```

Un array puede ser declarado sin asignar valores a cada posición antes de ser utilizado como en la variable `Mi_matriz` del ejemplo anterior.

Es posible declarar arrays como la variable `PuertosPWM` sin definir un tamaño explícito. El compilador cuenta los elementos y crea el array de tamaño apropiado.

Finalmente, es posible declarar arrays con tamaño y asignar valores, como en las variables `Pos_Neg`, `texto1` y `texto2`.

Observa que cuando declaramos un array tipo `char`, como en `texto1` y `texto2` se requiere un elemento más en su inicialización, para mantener el carácter nulo "`\0`" del código ASCII 0 (vea el Apéndice B). Esto permite saber dónde termina un array en la memoria del microcontrolador, de lo contrario formaría parte de los bytes de memoria, que no son parte del array.

Esto quiere decir que un array tipo `char` necesita tener un carácter más que el texto que lo contiene. Aunque no es necesario agregar el carácter nulo, automáticamente, un array tipo `char` quedaría de la siguiente forma:

```
char texto1[5] = {'H','O','L','A','\0'};
```

Para leer un elemento de un array, basta escribir el nombre y la posición:

```
Mi_puerto = PuertosPWM[1]; /* Mi_puerto es igual a 5 que está en la
                              Posición 1 del array PuertosPWM. */
```

Igualmente, es posible asignar valores a una posición específica de un array declarado:

```
Mi_Array[0] = 3; // Asigna el valor 3 a la posición 0.
texto[2] = 'A';  // Asigna el carácter A a la posición 2.
```

Arrays multidimensionales

Arduino soporta arrays multidimensionales. Cuando un array multidimensional es declarado, este debe ser considerado como arrays de arrays. Esos arrays pueden ser armados para tener dos, tres, cuatro, o más dimensiones.

Un típico array bidimensional de enteros sería declarado como:

```
int Mi_Array[5][10];  // [filas] [Columnas]
```

En memoria, los elementos del array serían almacenados en filas secuenciales como esto:

```
Mi_Array[0][0], Mi_Array[0][1], Mi_Array[0][2], ... Mi_Array[0][9],
Mi_Array[1][0], Mi_Array[1][1], Mi_Array[1][2], ... Mi_Array[1][9],
Mi_Array[2][0], Mi_Array[2][1], Mi_Array[2][2], ... Mi_Array[2][9],
Mi_Array[3][0], Mi_Array[3][1], Mi_Array[3][2], ... Mi_Array[3][9],
Mi_Array[4][0], Mi_Array[4][1], Mi_Array[4][2], ... Mi_Array[4][9]
```

Cuando declaramos un array bidimensional, el diseño es el mismo, filas secuenciales:

```
int Mi_matriz[5][3] = { 1,  2,  3,
                        4,  5,  6,
                        7,  8,  9,
                        10, 11, 12,
                        12, 14, 15 };

char Mi_teclado[4][4] = { {'1','2','3','A'},
                          {'4','5','6','B'},
                          {'7','8','9','C'},
                          {'*','0','#','D'} };
```

OPERACIONES ARITMÉTICAS

Empleando variables, valores constantes o componentes de un array pueden realizarse operaciones aritméticas y se puede utilizar el operador "casting" para conversión de tipos, por ejemplo:

```
int a = (int)3.5;
```

Además, pueden hacerse las siguientes asignaciones:

```
x ++;     // igual que x = x + 1, o incrementa x en + 1
x --;     // igual que x = x - 1, o decremento x en -1
x += y;   // igual que x = x + y, o incrementa x en +y
x -= y;   // igual que x = x - y, o decremento x en -y
x *= y;   // igual que x = x * y, o multiplica x por y
x /= y;   // igual que x = x / y, o divide x por y
```

Para su utilización en sentencias condicionales u otras funciones, Arduino permite utilizar los siguientes operadores de comparación:

```
x == y;   // x es igual a y
x != y;   // x no es igual a y
x < y;    // x es menor que y
x <= y;   // x es menor que o igual a y
x > y;    // x es mayor que y
x >= y;   // x es mayor que o igual a y
```

Los operadores lógicos son usualmente una forma de comparar dos expresiones y devolver un **VERDADERO** o **FALSO** dependiendo del operador. Existen tres operadores lógicos, **AND** (&&), **OR** (||) y **NOT** (!), que a menudo se utilizan en sentencias condicionales:

```
// Lógica AND:
if (x > 0 && x < 5)//Cierto solo si las dos expresiones son ciertas.

// Lógica OR:
if (x > 0 || y > 0)//Cierto si cualquiera de las expresiones
es cierta.

// Lógica NOT:
if (!x > 0)//Cierto solo si la expresión es falsa.
```

SENTENCIAS DE CONTROL

Las sentencias de control son usadas para controlar el flujo de ejecución de un programa.

El lenguaje de Arduino permite realizar sentencias de control: `if`, `if... else`, `for`, `while`, `do... while`, y `switch/case`. Su uso es similar a las funciones correspondientes en C.

If (Si condicional)

La sentencia `if` se usa para determinar si una condición se ha cumplido.

Un `if` tiene el siguiente formato:

```
if (expresión)
{
  Sentencia_1;
  Sentencia_2;
  ...
}
```

o

```
if (expresión)
  Sentencia;
```

Por ejemplo:

```
if (A == B)
{
  // Si A es igual a B, ejecuta este bloque.
}
```

Si el resultado de la expresión es **VERDADERO**, entonces la sentencia o bloque de sentencias es ejecutado. Por otra parte, si el resultado de la expresión es **FALSO**, entonces la sentencia o bloque de sentencias es omitido.

if... else (Si ... No...)

La sentencia `if... else` se usa para dirigir la dirección de un programa basado en la evaluación de una sentencia condicional, es decir, si no se cumple haz este otro.

Un `if... else` tiene el siguiente formato:

```
if (expresión)
{
  Sentencia_1;
  Sentencia_2;
  ...
}
else
{
  Sentencia_3;
  Sentencia_4
  ...
}
```

o

```
if (expresión)
  Sentencia_1;
else
  Sentencia_2;
```

Por ejemplo:

```
if (A > B)
{
  // Si A es mayor que B, ejecuta este bloque
}
else {
  // Si A no es mayor que B, ejecuta esta bloque.
}
```

La sentencia `else` añade la característica especial para el flujo de programa de tal manera que una sentencia o un bloque de sentencias asociado con `else` será ejecutado solo si el resultado de la expresión es **FALSO**. El bloque de sentencias será omitido si el resultado de la expresión es **VERDADERO**.

Una técnica de programación típica es colocar `if... else` en cascada para crear un árbol de selección:

```
if (expression_1)
  Sentencia_1;
else if (expression_2)
  Sentencia_2;
else if (expression_3)
```

```
    Sentencia_3;
else
    Sentencia_4;
```

La estructura de sentencias `if... else` seleccionará y ejecutará una sola sentencia. Si cualquiera de las expresiones es **VERDADERO** ejecutará su sentencia y el resto de expresiones serán omitidas.

Bucle for

Un bucle `for` es generalmente usado para ejecutar una sentencia o un bloque de sentencias a un número determinado de veces.

Un `for` tiene el siguiente formato:

```
for (expre_1; expre_2; expre_3)
{
  Sentencia_1;
  Sentencia_2;
  ...
}
```

o

```
for (expre_1; expre_2; expre_3)
  Sentencia;
```

Por ejemplo:

```
for(int a=0; a < 10; a++){
  // Ejecuta la sentencia 10 veces cuando "a" es menor a 10
}
```

La primera expresión (`expre_1`) corresponde a la inicialización de una variable local y se ejecuta una sola vez. La segunda expresión (`expre_2`) corresponde a la condición de control y es usado para determinar hasta cuándo tiene que permanecer ejecutándose el bucle `for`. La tercera expresión (`expre_3`) es usada para satisfacer la segunda expresión.

Cuando se ejecuta un programa, `expre_1` es ejecutado, `expre_2` es evaluado y si el resultado de la expresión es **VERDADERO**, entonces las sentencias dentro del bucle son ejecutadas. Durante el fin de cada ejecución dentro del bucle, `expre_3` es ejecutado, y el flujo del programa vuelve a ejecutar todas las sentencias del bucle, cuando `expre_2` es evaluado de nuevo. Si `expre_2` es **FALSO**, el bucle es completamente anulado y no ejecutará las sentencias dentro del bucle.

Bucle while

Un bucle `while` es un bucle de ejecución continua mientras se cumpla la expresión colocada dentro de la cabecera del bucle. Para salir del bucle, la expresión dentro de la cabecera tendrá que cambiar.

Un `while` tiene el siguiente formato:

```
while (expresión)
{
  Sentencia_1;
  Sentencia_2;
  ...
}
```

o

```
while (expresión)
  Sentencia;
```

Por ejemplo:

```
while(A==B){
  // Ejecuta la sentencia repetitivamente cuando A es igual a B
}
```

Para ejecutar el bucle `while`, primero evalúa la expresión. Si la expresión es **VERDADERO**, entonces las sentencias dentro del bucle son ejecutadas, pero si la expresión es **FALSO**, entonces el bucle es completamente anulado y no ejecutará las sentencias dentro del bucle.

Bucle do... while

Un bucle `do... while` es muy parecido al bucle `while`, excepto que la expresión es evaluada después de que el bucle ha sido ejecutado una sola vez. Eso quiere decir que los bucles `do... while` son siempre ejecutados una vez antes de determinar si el bucle será nuevamente ejecutado.

Un `do... while` tiene el siguiente formato:

```
do
{
  Sentencia_1;
  Sentencia_2;
  ...
} while (expresión)
```

o

```
do
  Sentencia;
while (expresión)
```

Por ejemplo:

```
do{
  /* Ejecuta el bucle una primera vez, y si A es igual a B vuelve a
     ejecutar la sentencia. */
}while(A==B);
```

Después de la primera ejecución del bucle `do... while`, la expresión es evaluada. Si la expresión es **VERDADERO**, entonces el flujo del programa retorna a ejecutar nuevamente todo el bucle, pero si la expresión es **FALSO**, el bucle es anulado y no retorna a ejecutar las sentencias dentro del bucle.

Sentencia switch/case

La sentencia `switch/case` se usa para ejecutar un caso que contiene una sentencia o un bloque de sentencias seleccionado por el valor de una expresión.

La sentencia `switch/case` posee la palabra clave `break`. Esta es usualmente usada para finalizar cada caso. Sin la sentencia `break`, la sentencia `switch` continuará la ejecución de los siguientes casos hasta hallar un `break`, o hasta el fin de la sentencia `switch`.

Un `switch/case` tiene el siguiente formato:

```
switch (expresión)
{
  case constante_1:
    Sentencia_1;
  break;
  case constante_2:
    Sentencia_2;
  break;
  ...
  case constante_N:
    Sentencia_N;
  break;
  default:
    Sentencia_0;
  break;
}
```

Por ejemplo:

```
switch (variable)
{
  case 1:
    // Ejecuta esta sentencia cuando la variable es igual a 1.
  break;
  case 2:
```

```
    // Ejecuta esta sentencia cuando la variable es igual a 2.
  break;
  default:
    /* Ejecuta esta sentencia cuando la variable no cumple con
    ninguno de los casos anteriores. */
  break;
}
```

Durante la ejecución de un programa, la expresión es evaluada y su valor es comparado con las constantes de cada uno de los casos (`constante_1, constante_2, ... constante_N`). Si se cumple uno de los casos, ejecuta las sentencias y después lo finaliza para posteriormente retornar al programa principal, y evaluar nuevamente la expresión.

Las constantes para todos los casos deben ser valores de tipo entero o de tipo carácter.

ENTRADAS Y SALIDAS DIGITALES Y ANALÓGICAS

La función `pinMode(puerto, modo)` es usada dentro de la función `setup()` para configurar un puerto en el modo `INPUT` u `OUTPUT`. Ejemplo: `pinMode(13, OUTPUT)` configura el puerto digital 13 como salida. Los puertos de Arduino funcionan por defecto como entradas, de forma que no necesitan declararse explícitamente como entradas empleando `pinMode()`.

La función `digitalRead(puerto)` lee el valor desde un puerto digital específico. Esta función devuelve un valor `HIGH` o `LOW`. El puerto digital puede ser especificado con una variable o una constante (0 – 13), por ejemplo: `v = digitalRead(2)`. En este ejemplo la variable `v` almacena los valores de `HIGH` o `LOW` del puerto digital 2.

La función `digitalWrite(puerto, valor)` introduce un nivel alto (`HIGH`) o bajo (`LOW`) en el puerto digital especificado. El puerto puede ser especificado con una variable o una constante (0 – 13). Ejemplo: `digitalWrite(10, HIGH)`.

La función `analogRead(puerto)` lee el valor desde el puerto analógico especificado con una resolución de 10 bits. Esta función solo funciona en los puertos analógicos (0 – 5). El valor resultante es un entero de 0 a 1023. Los puertos analógicos a diferencia de los digitales no necesitan declararse previamente como `INPUT` u `OUTPUT`, por ejemplo: `A = analogRead(A0)`. En este ejemplo se usa el puerto analógico A0 para lectura de valores y almacenados en la variable `A`.

La función `analogWrite(puerto, valor)` escribe un valor pseudo-analógico usando modulación por ancho de pulso (PWM) en un puerto de salida marcado como PWM. Esta función está activa para los puertos digitales 3, 5, 6, 9, 10, 11 en Arduino UNO, por ejemplo: `analogWrite(3, 255)`. En este ejemplo marca el puerto digital 3 como salida PWM en 255.

FUNCIONES DE TIEMPO

La función `delay(ms)` realiza una pausa en el programa la cantidad de tiempo en milisegundos (ms) especificada en el parámetro.

La función `delayMicroseconds(us)` realiza una pausa en el programa la cantidad de tiempo en microsegundos (us) especificada en el parámetro. Un segundo en milisegundos es 1000, un milisegundo en microsegundos es 1000, por lo tanto, un segundo en microsegundos es 1000000.

La función `millis()` devuelve la cantidad de milisegundos que lleva la placa Arduino ejecutando el programa actual como un valor **`unsigned`** `long`. Después de 9 horas el contador vuelve a 0 (cero).

La función `micros()` devuelve la cantidad de microsegundos después de empezar a ejecutar el programa actual en la tarjeta Arduino con un valor **`unsigned long`**. Después de aproximadamente 70 minutos el contador vuelve a 0 (cero).

FUNCIONES DE MATEMÁTICAS

Las funciones **`min`**`(x,y)` y **`max`**`(x,y)` devuelven el mínimo y el máximo respectivamente de entre dos números. Donde `x` es el primer valor e `y` es el segundo valor.

La función **`abs`**`(x)` calcula el valor absoluto de un número.

La función **`constrain`**`(x,a,b)` delimita un número dentro de un rango. Donde `x` es el número, `a` es el número más bajo del rango y `b` es el número más alto del rango.

La función **`map`**`(x,a,b,x,y)` mapea un número de un rango a otro. Donde `x` es el número, `a` y `b` son el rango original del número, y, `x` e `y` son el rango a convertir.

La función **`sqrt`**`(x)` devuelve la raíz cuadrada de un número.

FUNCIONES DE GENERACIÓN ALEATORIA

La función `randomSeed(x)` inicializa el generador de números pseudo-aleatorios, haciendo que se inicie en un punto arbitrario en su secuencia aleatoria. Esta secuencia, aunque muy larga, y aleatoria, es siempre la misma. Esta función especifica un valor como el punto de inicio para la función `random()`. Este parámetro debe ser realmente aleatorio y para ello puede emplearse la función `millis()` o incluso `analogRead()` para leer ruido eléctrico desde una entrada analógica.

Las funciones `random(max)` y `random(min, max)` devuelven un valor aleatorio entre el rango especificado.

FUNCIONES DE TRANSFERENCIA DE DATOS

`Serial``.begin(velocidad)`. Abre un puerto serial y especifica la velocidad de transmisión. La velocidad típica para comunicación con el ordenador es de 9600 baudios/segundo aunque se pueden soportar otras velocidades.

`Serial``.println(dato)`. Imprime datos al puerto serial seguido por un retorno de línea automático. Este comando tiene la misma forma que **`Serial`**`.print()` pero este último sin el salto de línea al final. Este comando puede emplearse para realizar la depuración de programas. Para ello puede mandarse mensajes de depuración y valores de variables por el puerto serial. Posteriormente, desde el entorno de programación de Arduino, activando el `"Serial Monitor"` se puede observar el contenido del puerto serial, y, por lo tanto, los mensajes de depuración. Para observar correctamente el contenido del puerto serial se debe tener en cuenta que el `"Serial Monitor"` y el puerto serial han de estar configurados a la misma velocidad (para configurar la velocidad del puerto serial se hará con el comando **`Serial`**`.begin(velocidad)`).

`Serial``.write(data)`. Escribe datos binarios en el puerto serial. Este dato es enviado como un byte en una serie de bytes. Para enviar caracteres que representan los dígitos de un número se utiliza la función **`Serial`**`.print()` en su lugar.

`Serial``.read()`. Lee o captura un byte (un carácter) desde el puerto serial. Devuelve -1 si no hay ningún carácter en el puerto serial.

`Serial``.available()`. Devuelve el número de caracteres disponibles para leer desde el puerto serial.

RESUMEN DE ESTRUCTURAS, VARIABLES Y FUNCIONES

Estructura

Funciones principales:
`setup()` (*inicialización*)
`loop()` (*bucle*)

Estructuras de control:
`if` (*Si condicional*)
`if...else` (*Si...no*)
`for` (*bucle con contador*)
`switch/case` (*comparador múltiple*)
`while` (*bucle por comparación booleana*)
`do...while` (*bucle por comparación booleana*)
`break` (*salida del bloque del código*)
`continue` (*continuación en bloque de código*)
`return` (*devuelve valor a programa*)

Sintaxis:
`;` (*punto y coma*)
`{}` (*llaves*)
`//` (*comentario en una línea*)
`/*...*/` (*comentarios en múltiples líneas*)

Operaciones aritméticas:
`=` (*asignación*)
`+` (*suma*)
`-` (*resta*)
`*` (*multiplicación*)
`/` (*división*)
`%` (*resto*)

Variables

Constantes:
`HIGH` & `LOW`
`INPUT` & `OUTPUT`
`True` & `false`

Constantes numéricas:
`const int`
`const long`
`u` o `U` (*forzar valores sin signo*)
`l` o `L` (*forzar valores a formato de 32bits*)
`ul` o `UL` (*Forzar valores a formato sin signo de 32bits*)

Tipos de datos:
`boolean` (*booleano*)
`char` (*carácter*)
`byte` (*variable de 8 bits*)
`int` (*entero*)
`unsigned int` (*entero sin signo*)
`long` (*entero 32bits*)
`unsigned long` (*entero 32bits sin signo*)
`float` (*en coma flotante*)
`double` (*en coma flotante de 32bits*)
`string` (*cadena de caracteres*)
`array` (*cadena*)
`void` (*vacío*)

Conversión:
`char()`

Funciones

E/S Digitales:
`pinMode()`
`digitalWrite()`
`digitalRead()`

E/S Analógicas:
`analogRead()`
`analogWrite()` (*PWM modulación por ancho de pulso*)

E/S Avanzadas:
`tone()`
`noTone()`
`shiftOut()`
`pulseIn()`

Tiempo:
`millis()`
`micros()`
`delay()`
`delayMicroseconds()`

Matemáticas:
`min()` (*mínimo*)
`max()` (*máximo*)
`abs()` (*valor absoluto*)
`constrain()` (*limita*)
`map()` (*cambia valor de rango*)
`pow()` (*eleva a un número*)
`sq()` (*eleva al cuadrado*)
`sqrt()` (*raíz cuadrada*)

Trigonometría:
`sin()` (*seno*)
`cos()` (*coseno*)

Operaciones comparativas:

== (*igual a*)
!= (*distinto de*)
< (*menor que*)
> (*mayor que*)
<= (*menor o igual que*)
>= (*mayor o igual que*)

Operaciones booleanas:

&& (*y*)
|| (*o*)
! (*negación*)

Operaciones de composición:

++ (*incrementa*)
-- (*decremento*)
+= (*composición suma*)
-= (*composición resta*)
*= (*composición multiplicación*)
/= (*composición división*)

```
byte()
int()
long()
float()
```

```
tan() (tangente)
```

Números aleatorios:

```
randomSeed()
random()
```

Comunicación serial:

```
Serial.begin()
Serial.read()
Serial.available()
Serial.print()
Serial.println()
Serial.write()
```

3 CODEBENDER

¿QUÉ ES CODEBENDER?

Codebender es un entorno de desarrollo de Arduino en la nube. Permite guardar, abrir, compartir, cargar, etc., proyectos en la nube, y trabajar con proyectos locales.

codebender

Vasilis Georgitzikis, también conocido como Tzikis, creó la aplicación **Codebender** para el desarrollo de sketch Arduino en la nube, con todas las disponibilidades de Arduino IDE como ejemplos, bibliotecas integradas, documentación y cargar códigos a cualquier placa Arduino desde un navegador.

Permite a cualquier usuario programar su Arduino directamente desde un navegador web, utilizando HTML5 y un pequeño plugin. La programación de una placa Arduino se puede realizar mediante un cable USB, o de forma remota a través de la red (Arduino Ethernet o Arduino Ethernet Shield).

Codebender permite en su plataforma el intercambio, como compartir, clonar y modificar códigos publicados, por lo tanto Codebender permite:

- Compartir tu trabajo con el mundo.
- Cargar un sketch existente a tu dispositivo.
- Clonar un sketch existente y modificarlo para tus necesidades.

- Código de inserción en tu sitio web, blog, o tutoriales.

www.codebender.cc

CREAR UNA CUENTA EN CODEBENDER

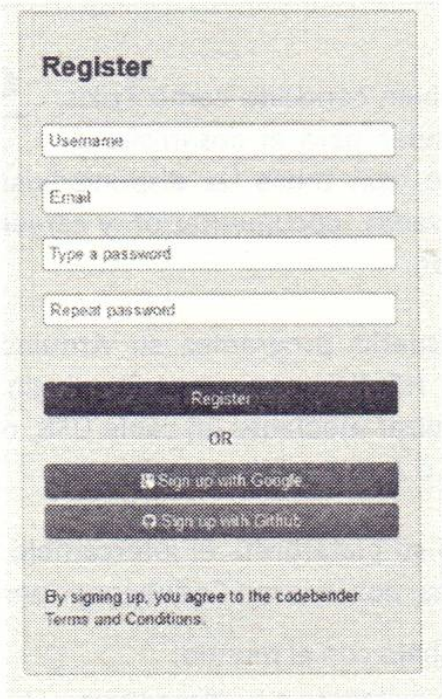
Para entrar en la plataforma web es importante crear una cuenta o registrarse como en cualquier sitio web de servicios online, correo, foros, etcétera.

Para registrar una cuenta es necesario seguir estos pasos:

1) Para crear una cuenta se puede introducir un nombre de usuario, un correo electrónico y una contraseña, o ingresar con una cuenta de **Google** o **Github**.

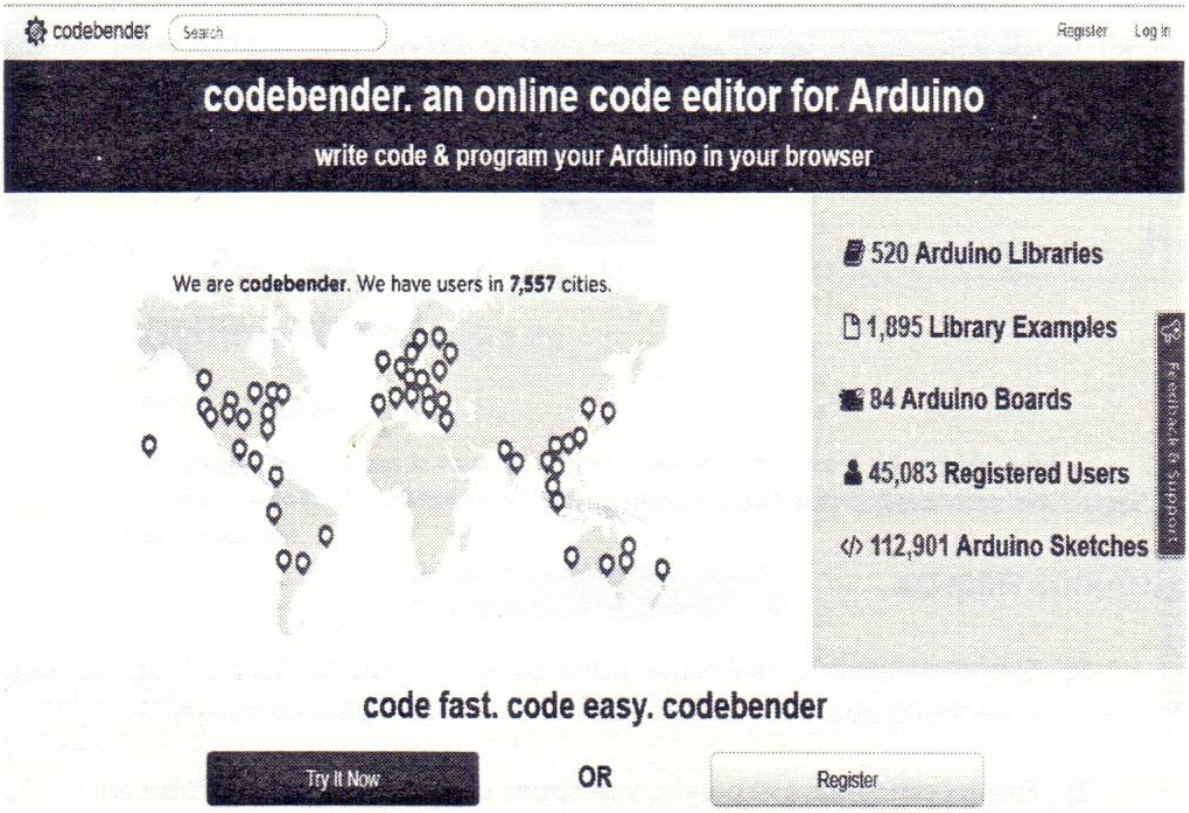
2) Una vez registrado, Codebender enviará un correo a la cuenta de correo electrónico con un link de activación para confirmar la cuenta.

3) Si el registro se hace con una cuenta de Google o Github, solo se necesita iniciar sesión y autorizar Codebender.



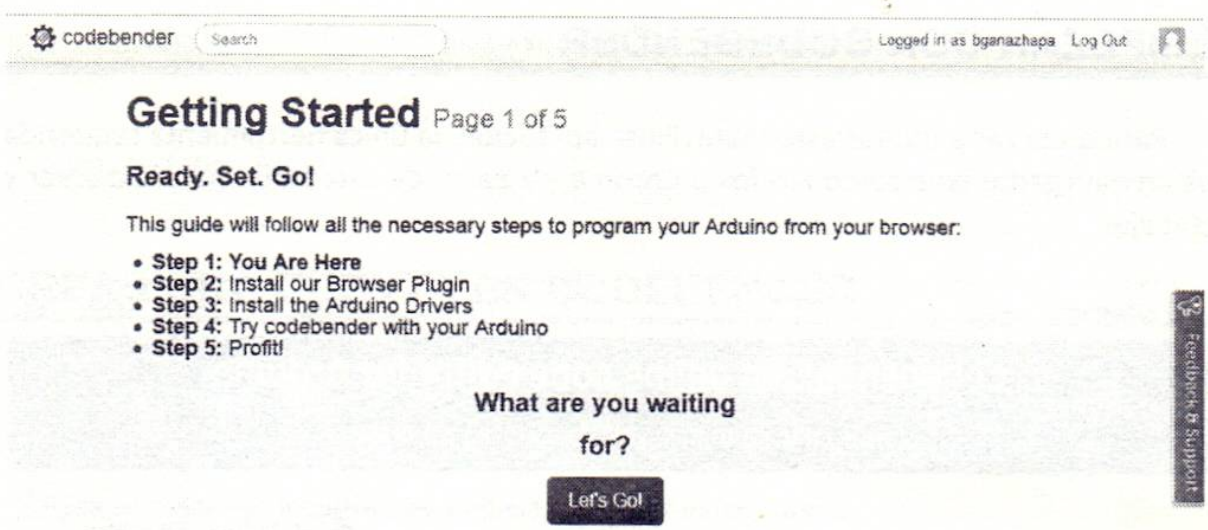
EMPEZAR CON CODEBENDER

Para empezar a utilizar esta maravillosa aplicación, la única herramienta requerida es un navegador web como Firefox o Chrome y a partir de esta se empezará a crear y trabajar.



Instalar Plugin Codebender

Después del registro, Codebender se dirigirá directamente a la página principal. Un mensaje aparecerá para informar de que es necesario seguir la guía "**Getting Started**" de Codebender. Haz clic en "**Let's Go!**".



Seguir el paso "**Install our Browser Plugin**".

USUARIOS FIREFOX

1) Si eres un usuario de Firefox, pulsa en el link "**Add to Firefox**". Un mensaje emergente aparecerá... solo haz clic en "**Allow**" para continuar.

2) Firefox descargará el plugin, y después una ventana abrirá. Pulsa en "**Install Now**" para continuar.

3) Después de finalizar con la instalación del plugin, Firefox deberá ser restaurado.

USUARIOS CHROME

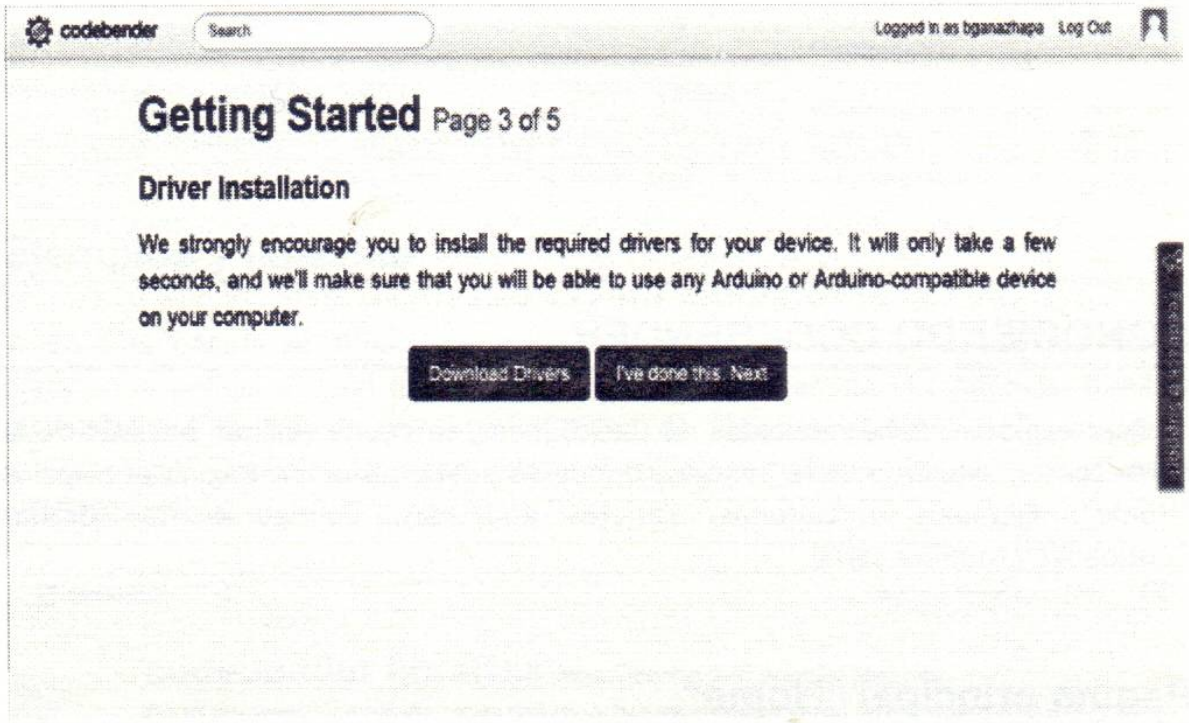
1) Si eres un usuario de Windows 8/8.1, haz clic en el link "**Add to Windows**" para descargar el plugin. Un archivo Codebendercc.msi emergerá en la carpeta de descargas. Haz doble clic en el archivo para instalar el plugin.

2) Si eres un usuario de Chrome en Linux, Mac OS X, u otras versiones de Windows, pulsa en el enlace "Add to Chrome" para llegar a la tienda web de Chrome. Ahí, haz clic en el botón "**+FREE**" para instalar el plugin.

3) Después de finalizar con la instalación del plugin, Chrome deberá ser restaurado.

Instalar drivers

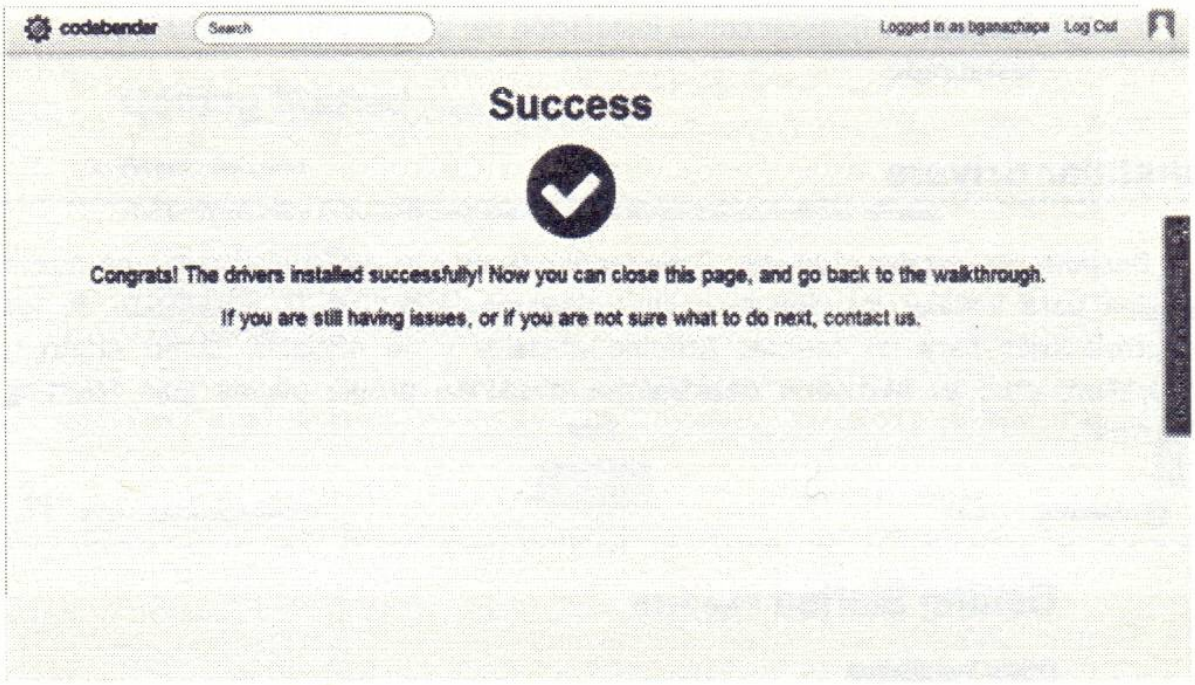
Después de instalar el plugin, Codebender se dirigirá directamente a una nueva página para instalar los drivers o controladores. Antes de la instalación de los controladores para las tarjetas Arduino oficiales y no oficiales (como Adafruit, SparkFun, etc.) es necesario descargarlos desde su propia página guía "Getting Started".



Haz clic en "**Download Drivers**" y automáticamente guardará un archivo zip en la dirección de descarga de tu ordenador.

Una vez finalizada la descarga, haz clic en el archivo zip y después ejecuta el archivo de instalación de los controladores.

Despues de instalar satisfactoriamente los controladores, una nueva página de éxito emergerá, "**Success**".

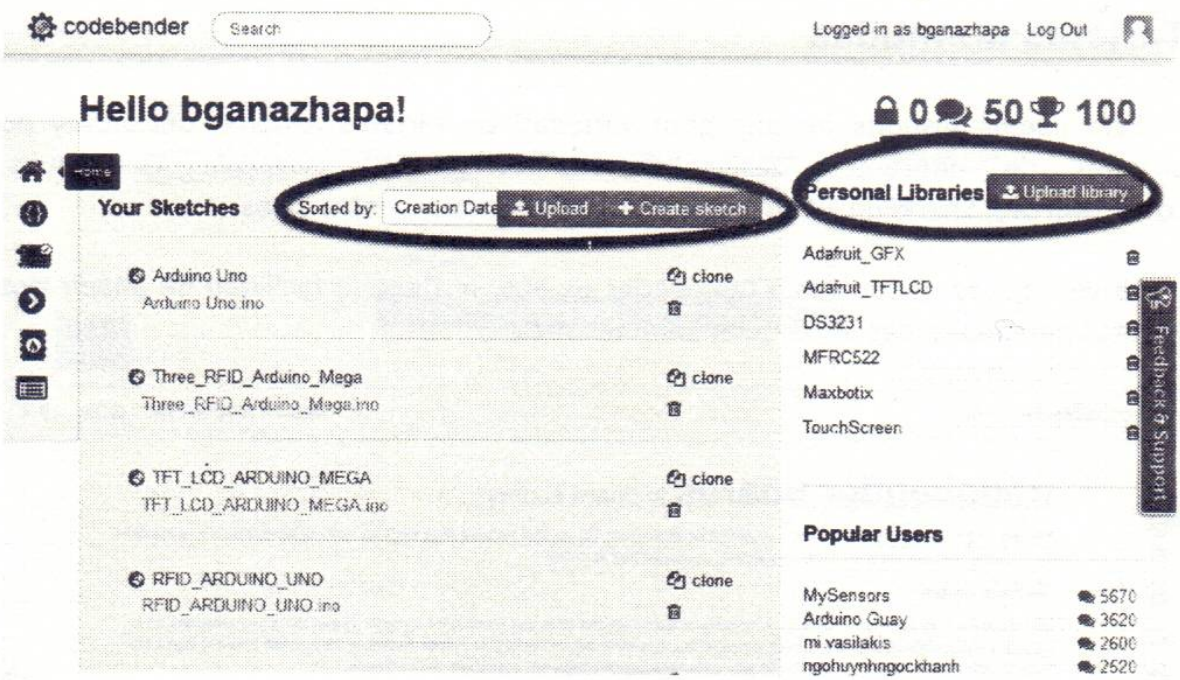


CONOCIENDO CODEBENDER

Para explorar las herramientas de Codebender, se puede realizar a través de la barra central izquierda de la pantalla, donde se puede observar: Página principal o "Home", Ejemplos y Librerías, Tarjetas admitidas, Configuraciones, Grabar Bootloader y Monitor serial.

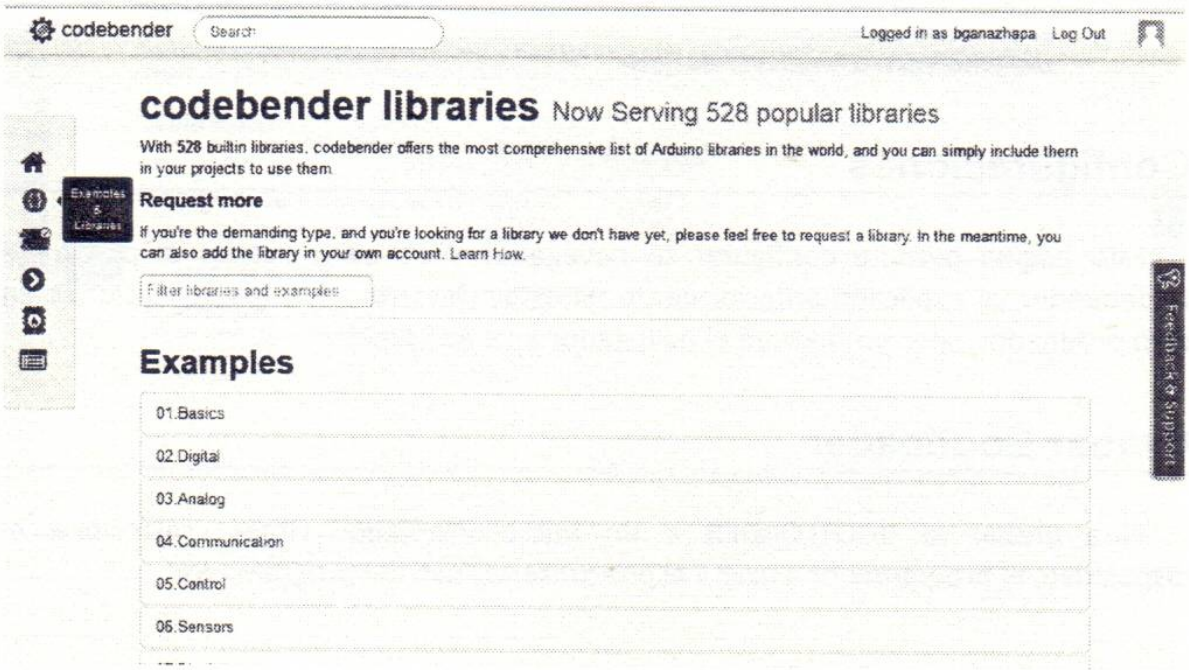
Página principal "Home"

En la página principal "Home" es donde se alojarán los sketch (programas) en la web y en donde se podrán crear o subir nuevos proyectos o sketch y subir librerías personalizadas por el usuario.



Ejemplos y librerías

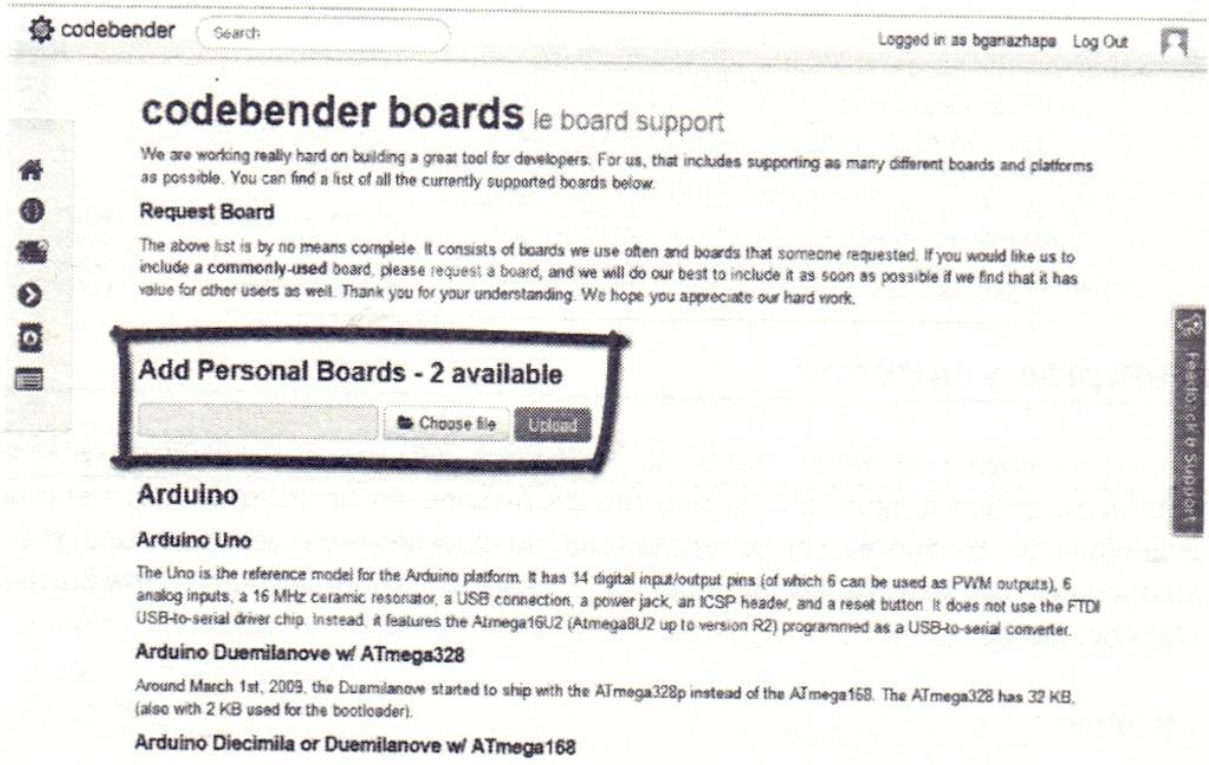
En esta página se alojan todos los ejemplos y las librerías dispuestos por la plataforma online al igual que el entorno de Arduino en un PC. Además, tiene la disponibilidad de contar con una infinidad de librerías externas para una gran variedad de dispositivos adaptables a cualquier tarjeta Arduino clasificadas alfabéticamente.



Tarjetas admitidas

Esta página dispone de una gran variedad de tarjetas Arduino oficiales y no oficiales de diferentes desarrolladores, como por ejemplo: TinyCircuits, LowPowerLab, SparkFun Electronics, Adafruit Industries, entre otros.

La ventaja de contar con Codebender es que se tiene la facilidad de añadir dos tarjetas personalizadas a cualquier característica.

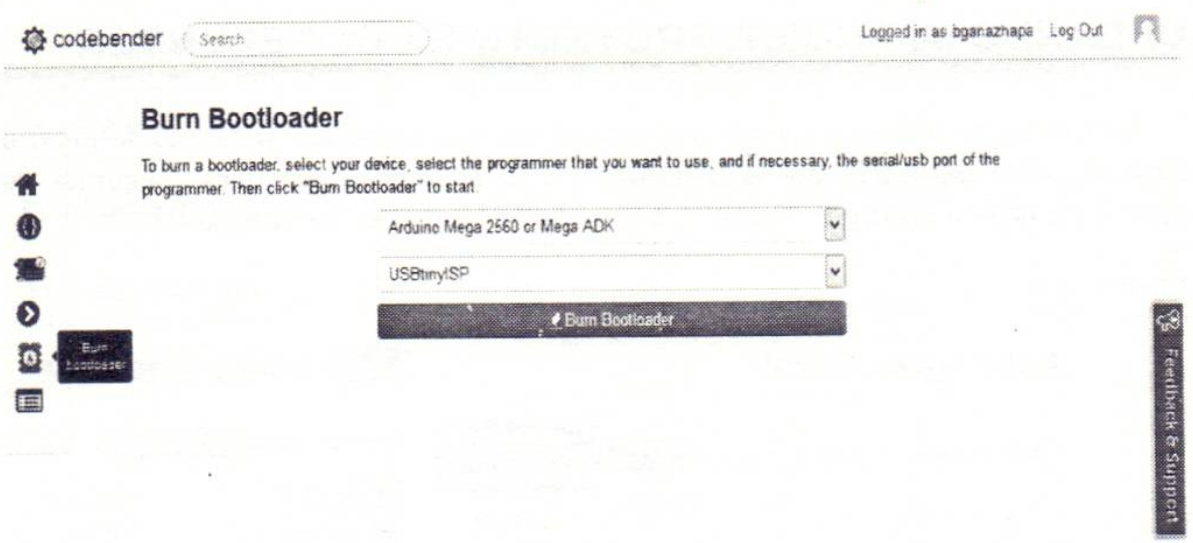


Configuraciones

Esta página permite configurar tu navegador Web para instalar el plugin de Codebender ya explicado anteriormente. Si estás desarrollando tus aplicaciones en otro ordenador, solo configurará el navegador en esta sección.

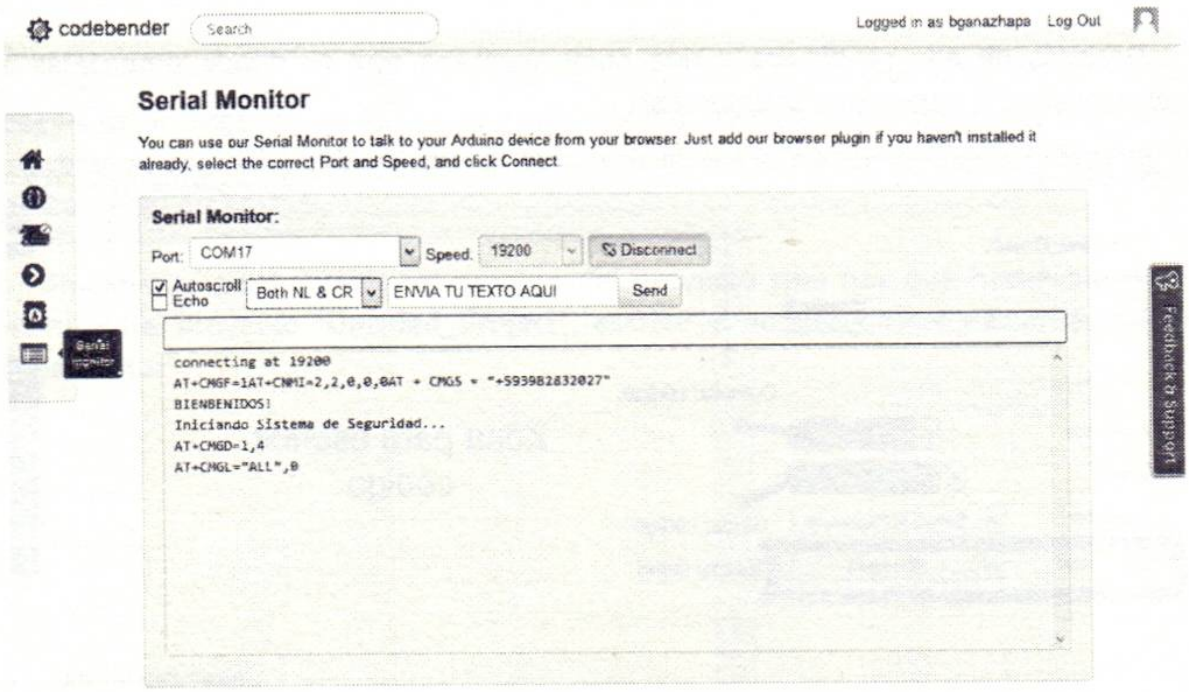
Grabar Bootloader

Para grabar el BOOTLOADER a un microcontrolador virgen, selecciona el dispositivo, el programador a usar y el puerto serial/usb del programador.



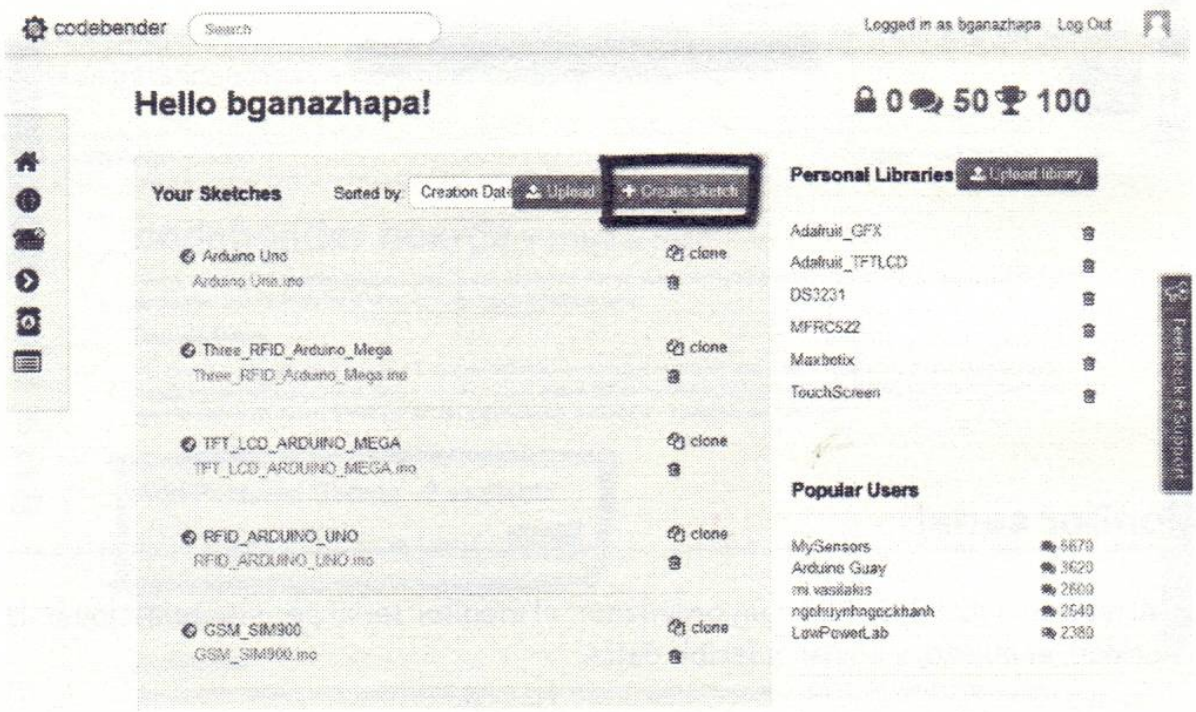
Monitor serial

Al igual que Arduino IDE en un ordenador, el monitor serial permite seleccionar la velocidad, el puerto, y enviar y recibir datos.

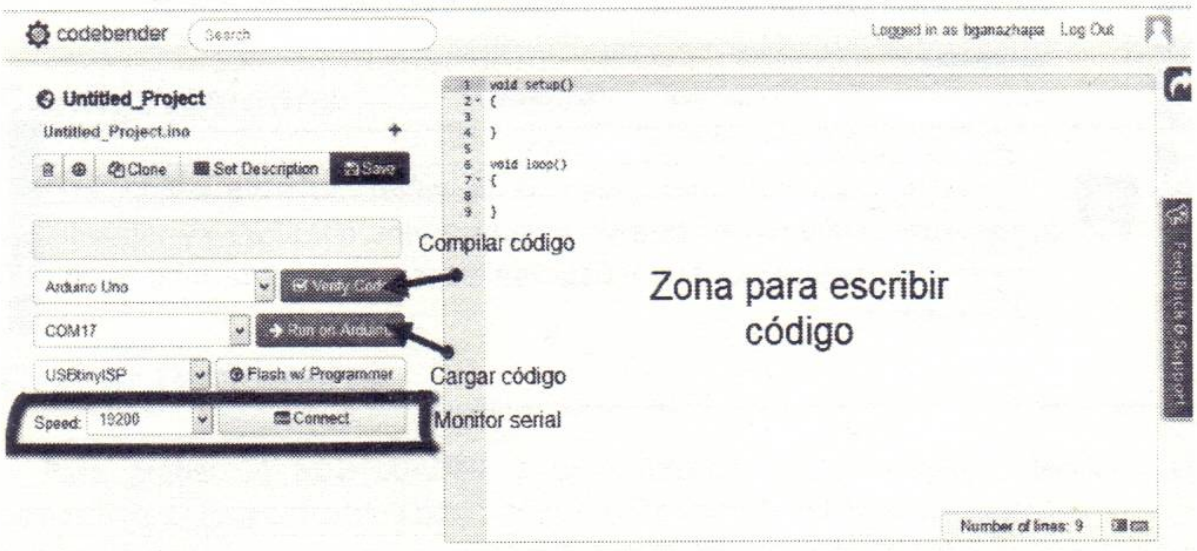


CREANDO MI PRIMER PROYECTO EN CODEBENDER

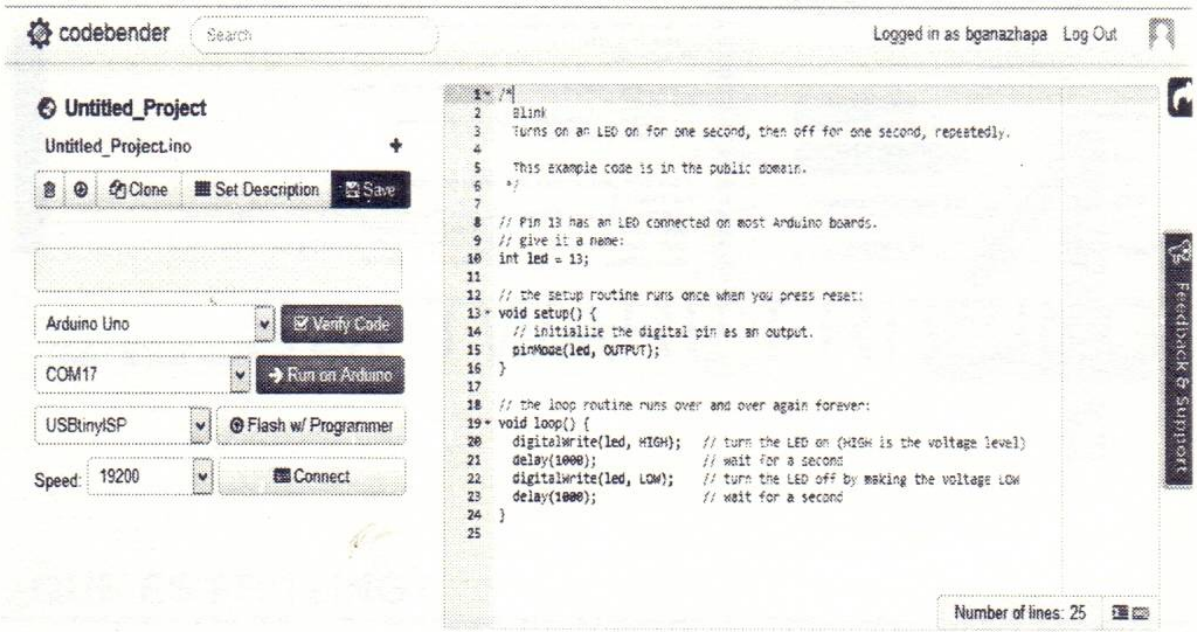
Para crear tu primer proyecto primero tienes que conectar tu tarjeta Arduino o cualquier otra permitida por la plataforma a tu ordenador, después asegúrate de estar en la página principal "Home" y hacer clic en la opción "**Create sketch**".



Después de ello se dirigirá a una nueva ventana que es donde empezarás a desarrollar tu programa en Codebender.



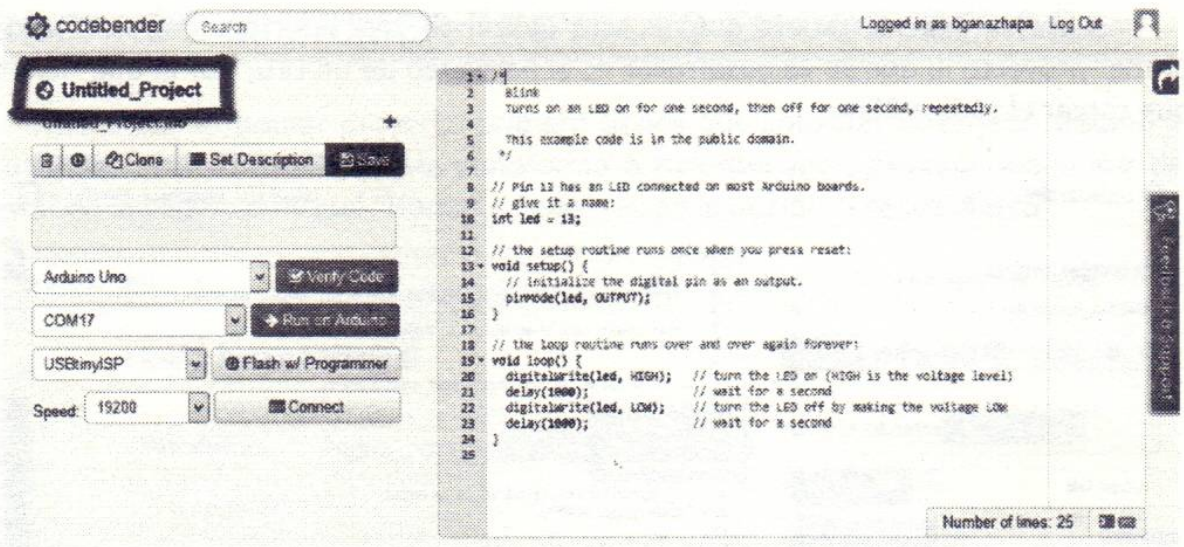
Después se elige la tarjeta que se está utilizando y el puerto serial. Para este primer proyecto lo que se va desarrollar es el parpadeo de un LED, por lo que tienes que cargar el código Blink.



Pulsa en "**Verify Code**" para compilar el programa y estar seguros de no cometer ningún error de sintaxis. Por último pulsa "**Run on Arduino**" para grabar el programa a la tarjeta Arduino o las que dispone la plataforma.

Codebender tiene la disponibilidad de contar con el monitor serial en la misma página del navegador, en la cual se desarrolla el proyecto, por lo que no es necesario cambiar de página. También tiene la opción de guardar, clonar, configurar, eliminar y descargar el programa.

Finalmente para cambiar el nombre del proyecto solo hay que hacer clic en el nombre del proyecto "**Untitled_Project**", escribir el nuevo nombre y guardar con la tecla "**Enter**".

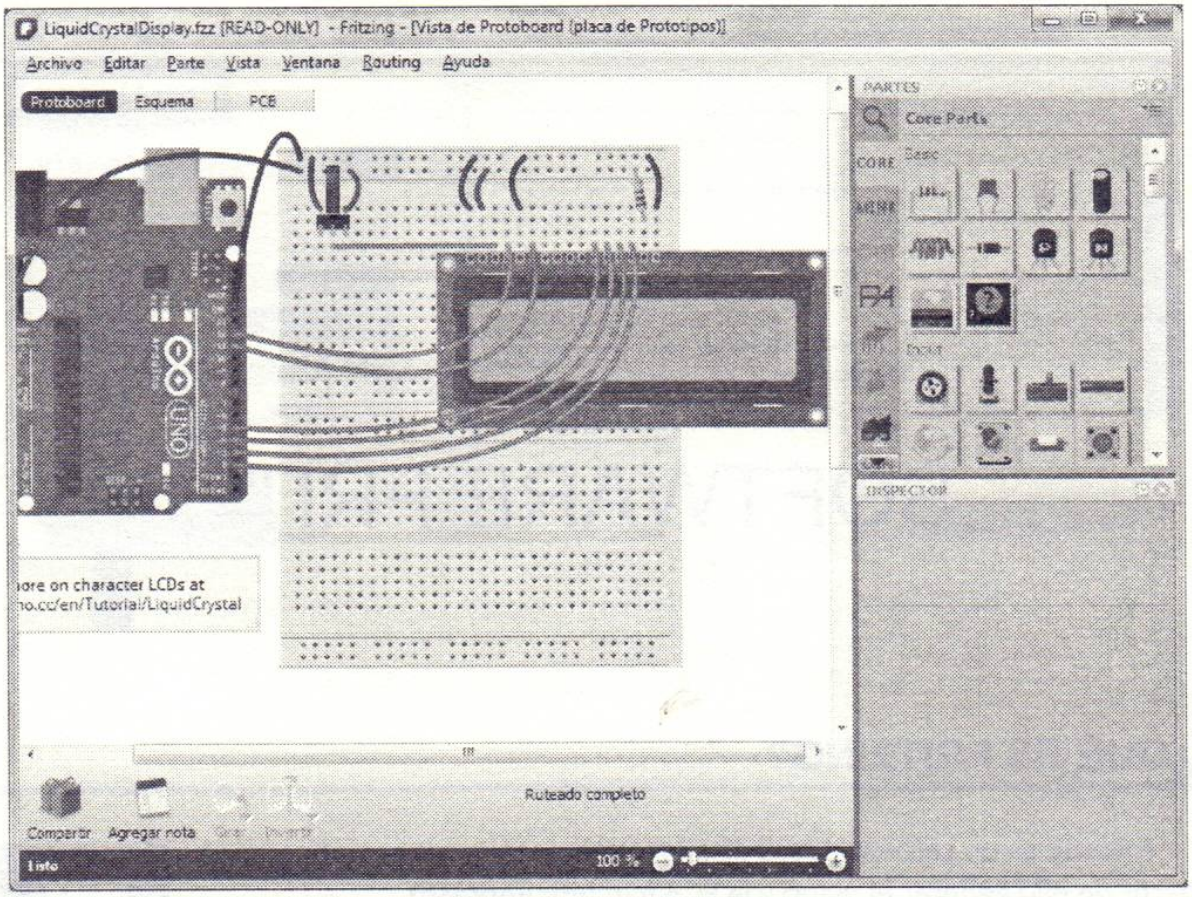


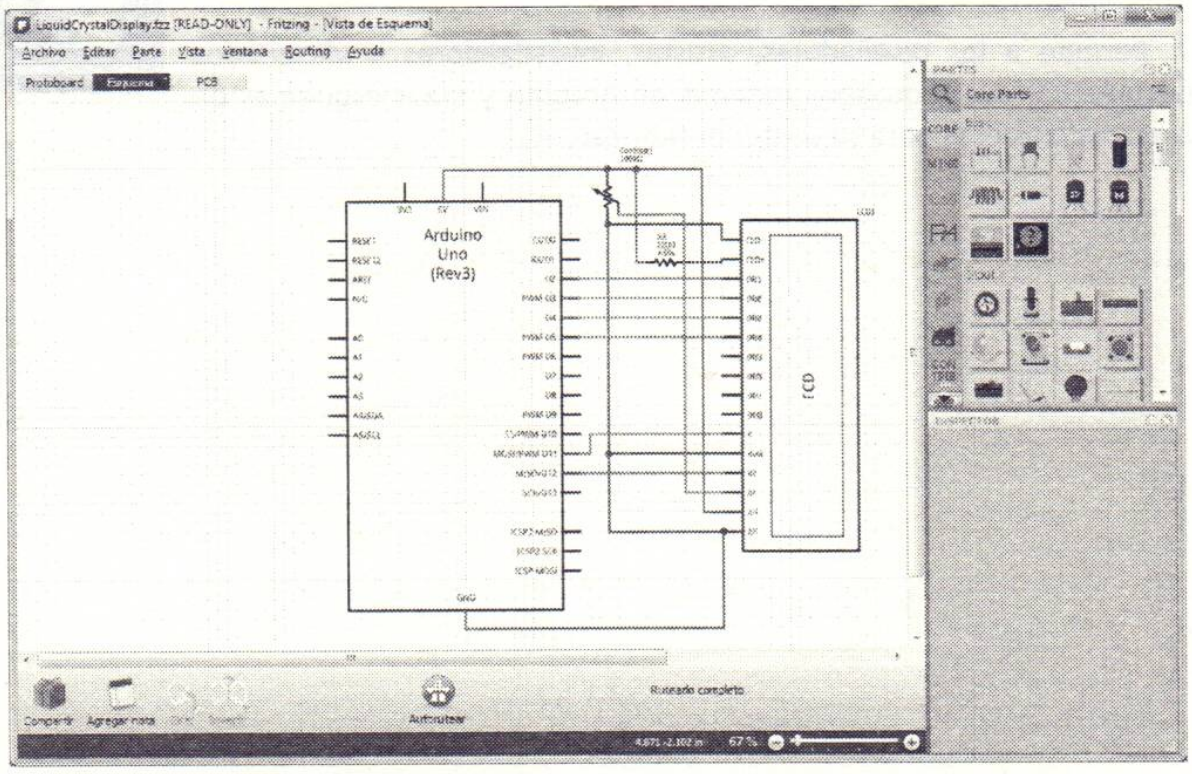
4 SOFTWARE FRITZING

¿QUÉ ES FRITZING?

Fritzing es un software para diseñar los montajes en protoboard y a partir de ello generar el plano y la PCB (circuito impreso).

Permite a los diseñadores, artistas, investigadores y aficionados documentar sus prototipos basados en Arduino y crear esquemas de circuitos impresos para su posterior fabricación.





Fritzing dispone de una gran variedad de componentes electrónicos, dispositivos, tarjeras Arduino, shields, sensores y dispositivos de diferentes fabricantes como: Parallax, SparkFun, etc.

EMPEZANDO A DISEÑAR EN FRITZING

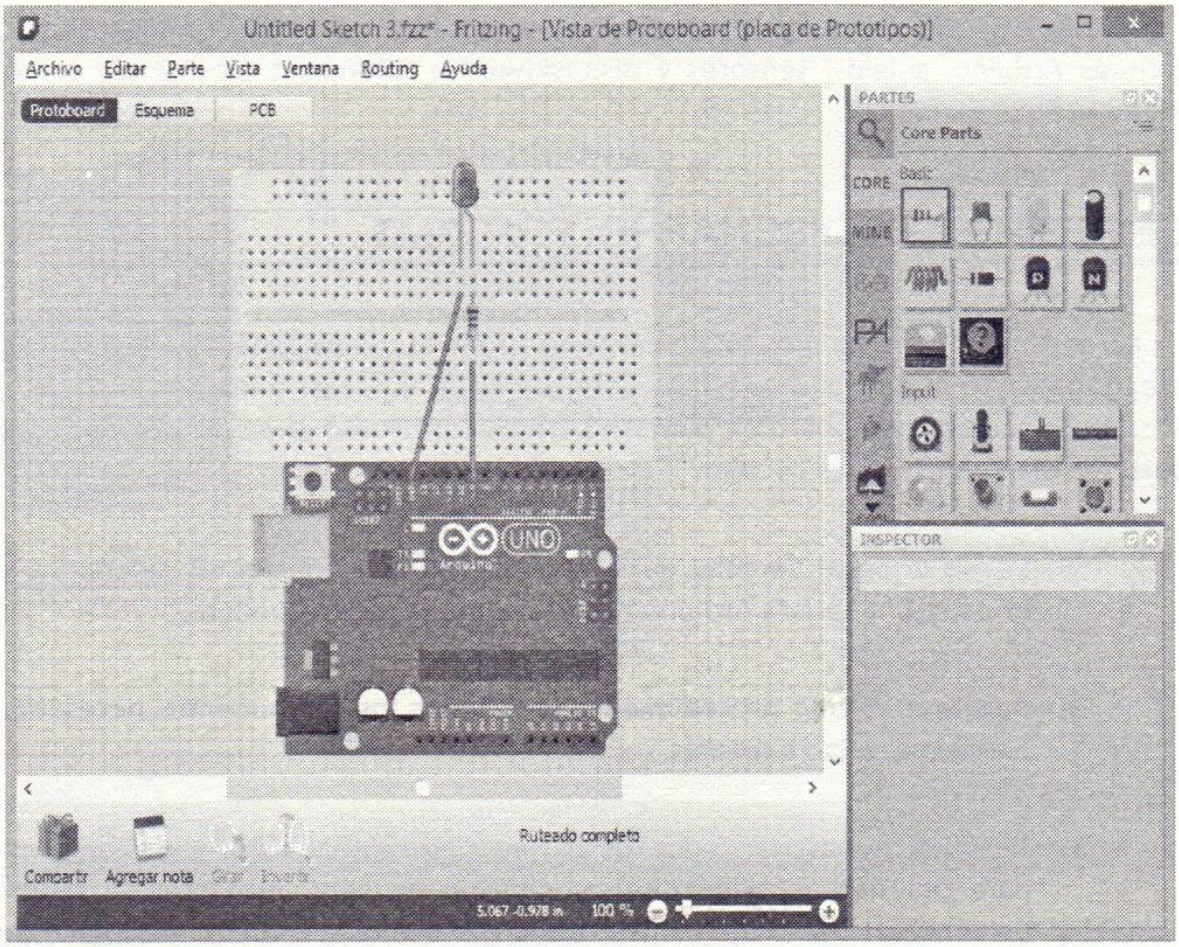
Para comenzar con un proyecto en Fritzing, es necesario disponer de un circuito y asegurarse de que funcione correctamente. Antes de empezar a diseñar el circuito, Fritzing brinda un protoboard en pantalla para comenzar con el nuevo proyecto, y a partir de él se construirá el circuito.

1) Desde la parte derecha de la pantalla, en el menú **Partes** selecciona y arrastra una tarjeta Arduino hasta la ventana del Protoboard.

2) Realiza el mismo procedimiento para todos los componentes necesarios para el proyecto a diseñar.

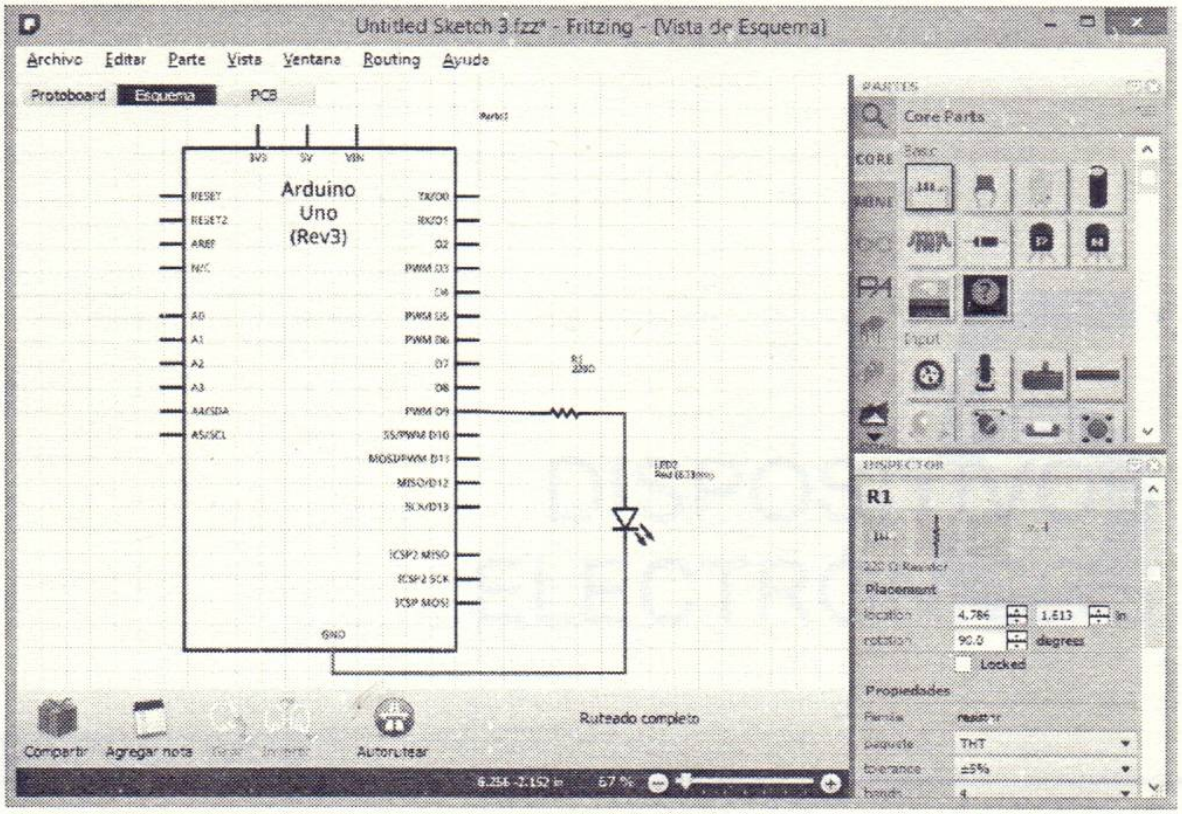
3) Después de obtener todos los componentes del circuito, se las organiza para obtener un buen diseño, y desde el menú **Inspector** se puede personalizar los componentes, como por ejemplo, cambiar el tamaño de un LED y su color.

4) Para empezar a cablear y conectar, simplemente selecciona y arrastra desde un pin de un componente hacia otro, por ejemplo: el conector GND de Arduino hasta el pin negativo de un LED.

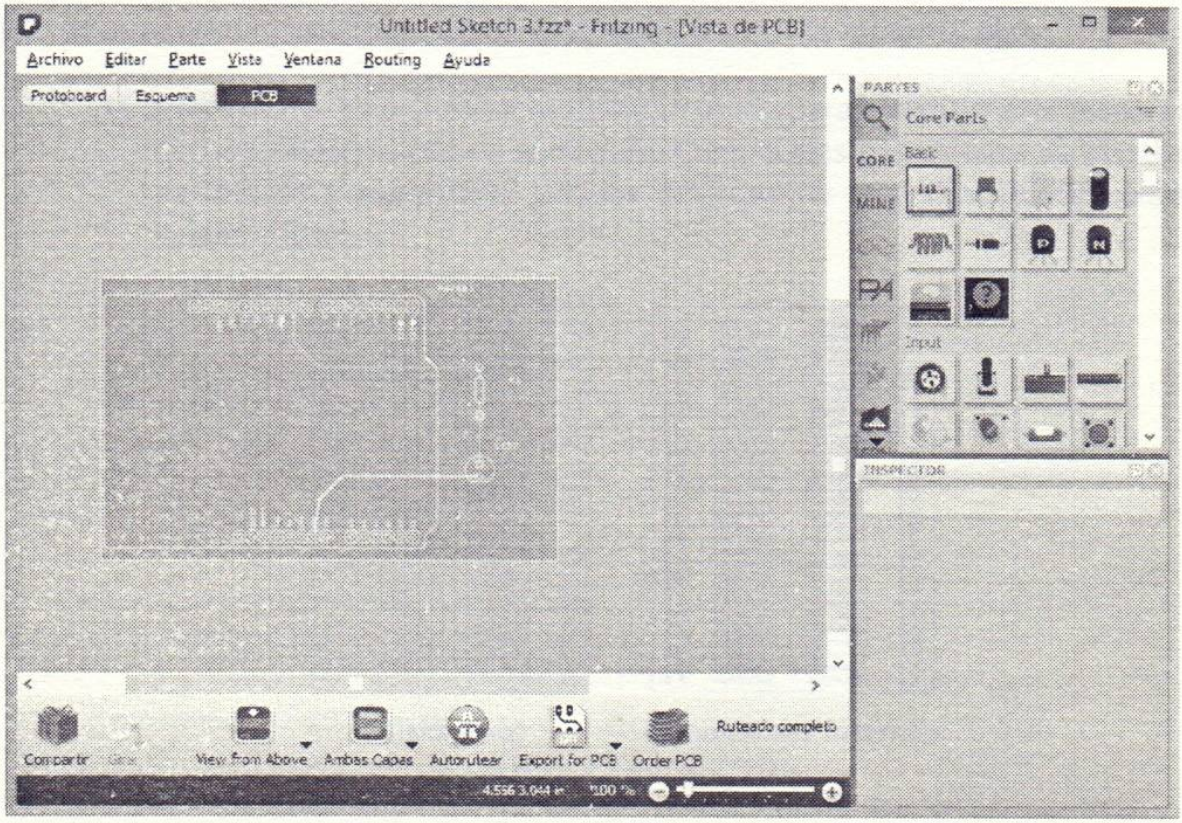


5) Puedes doblar los cables solo con añadir puntos de curvatura. Simplemente arrastra en cable hacia donde lo quieras doblar.

6) En la ventana Esquema puedes modificar el diseño para una mejor presentación.



7) En la ventana PCB puedes modificar el circuito para su fabricación.

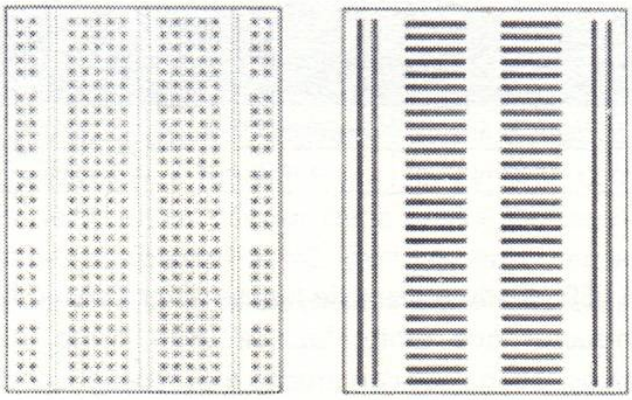


5 DISPOSITIVOS ELECTRÓNICOS

DISPOSITIVOS

Este capítulo integra una gran variedad de dispositivos electrónicos para desarrollar prácticas y aplicaciones en el entorno Arduino, por ejemplo: un Monitor LCD de temperatura y alarmas, tonos y melodías, control de motores, etc.

1) **Protoboard**. Es una placa reutilizable usada para construir prototipos de circuitos electrónicos sin soldadura. Compuestas por bloques de plástico perforados y numerosas láminas delgadas de una aleación de cobre, estaño y fósforo.

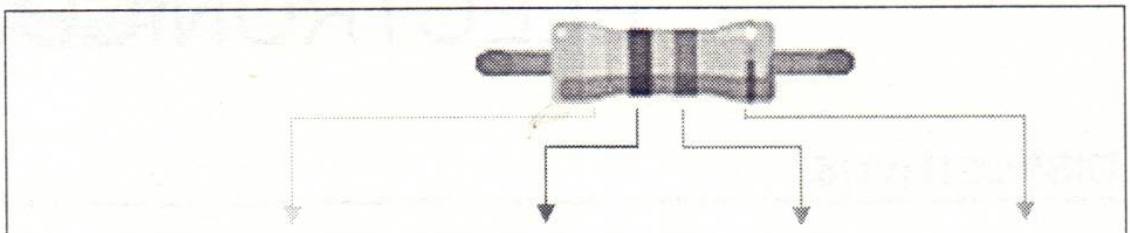


Vista Real **Conexiones internas**

2) Resistencia. Es un material formado por carbón y otros elementos resistivos para disminuir la corriente que fluye a través de un conductor. La corriente máxima en un resistor viene condicionada por la máxima potencia que puede disipar su cuerpo. El valor de la resistencia eléctrica se obtiene leyendo las cifras como un número de una, dos o tres cifras; se multiplica por el multiplicador y se obtiene el resultado en ohmios (Ω). Para consultar el cálculo de una resistencia a través de los bandas de colores, ver el Apéndice A. Código estándar de colores en resistencias.

Componente

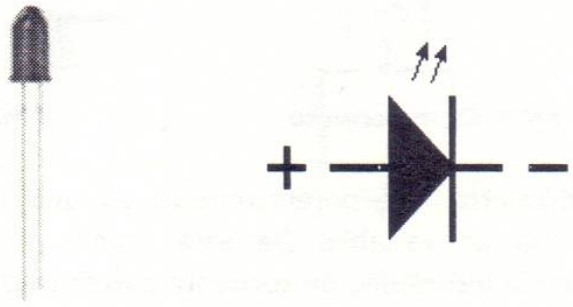
Símbolo



Color	Primera banda	Segunda banda	Tercera banda	Tolerancia
Oro			x 0.1Ω	± 5%
Plata			x 0.01Ω	± 10%
Negro	0	0	x 1Ω	
Marron	1	1	x 10Ω	± 1%
Rojo	2	2	x 100Ω	± 2%
Naranja	3	3	x 1000Ω	
Amarillo	4	4	x 10000Ω	
Verde	5	5	x 100000Ω	
Azul	6	6	x 1000000Ω	
Violeta	7	7	x 10000000Ω	
Gris	8	8		
Blanco	9	9		

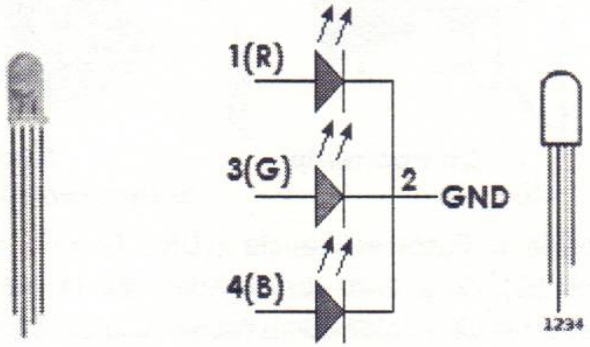
3) **LED**. Un LED (diodo emisor de luz, también "diodo luminoso") es un diodo semiconductor que emite luz. Se usan como indicadores en muchos dispositivos, y cada vez con mucha más frecuencia en iluminación. Los LED presentan muchas ventajas sobre las fuentes de luz incandescente como un consumo de energía mucho menor, mayor tiempo de vida, menor tamaño, gran durabilidad y fiabilidad.

El LED tiene una polaridad, un orden de conexión, y al conectarlo al revés se puede quemar, revisa los dibujos de la parte superior para conocer a qué corresponde el positivo y el negativo.



Componente **Símbolo**

4) **LED RGB**. Un LED RGB es un LED que incorpora en su mismo encapsulado tres LED, es RGB porque R (red, rojo), G (green, verde) y B (blue, azul), así se pueden formar miles de colores ajustando de manera individual cada color. Los tres LED están unidos por el negativo o cátodo.



Componente **Símbolo**

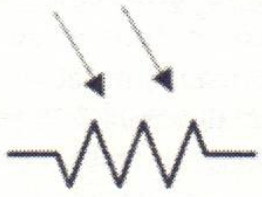
5) **Pulsador**. Un botón o pulsador se utiliza para activar alguna función. Los botones son por lo general activados al ser pulsados. Un botón de un dispositivo electrónico funciona por lo general como un interruptor eléctrico, es decir, en su interior tiene dos contactos, si es un dispositivo NA (normalmente abierto) o NC (normalmente cerrado), con lo que al pulsarlo se activará la función inversa de la que en ese momento esté realizando.

Componente **Símbolo**

6) **Potenciómetro**. Un potenciómetro es una resistencia cuyo valor de resistencia es variable. De esta manera, indirectamente, se puede controlar la intensidad de corriente que fluye por un circuito si se conecta en paralelo, o controlar el voltaje al conectarlo en serie. Son adecuados para su uso como elemento de control en los aparatos electrónicos. El usuario acciona sobre ellos para variar los parámetros normales de funcionamiento. Por ejemplo, el volumen de una radio.

Componente **Símbolo**

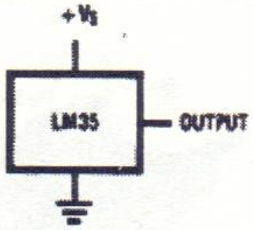
7) **Fotocelda** o **Fotorresistencia** (LDR). Es una resistencia, cuyo valor en ohmios (Ω) varía ante los cambios de la luz incidente. Una fotocelda presenta un bajo valor de su resistencia ante la presencia de luz y un alto valor de resistencia ante la ausencia de luz. Pueden encontrarse en muchos artículos de consumo, como, por ejemplo, en cámaras, medidores de luz, relojes con radio, alarmas de seguridad o sistemas de encendido y apagado del alumbrado público de las calles.



Componente **Símbolo**

8) **LM35**. El LM35 es un sensor de temperatura con una precisión calibrada de 1 °C. Su rango de medición abarca desde -55 °C hasta 150 °C. La salida es lineal y cada grado centígrado equivale a 10 mV.

Componente



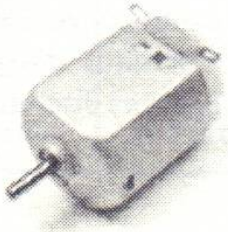
Símbolo

9) **Piezo Speaker (Altavoz)**. Es un dispositivo utilizado para reproducir sonido desde un equipo electrónico. También es llamado altoparlante, bocinar, loudspeaker. Convierte las ondas eléctricas en energía mecánica y esta se convierte en energía acústica. Técnicamente, es un transductor electroacústico que convierte una señal eléctrica en sonido.

Componente

Símbolo

10) **Motor DC**. El motor de corriente continua (DC) es una máquina que convierte la energía eléctrica en mecánica, provocando un movimiento rotatorio. Esta máquina de corriente continua es una de las más versátiles. Su fácil control de posición, paro y velocidad la han convertido en una de las mejores opciones en aplicaciones de control y automatización de procesos. Por ejemplo, los puedes encontrar en la tracción de los carros de juguetes de pilas o en las llantas de los robots.

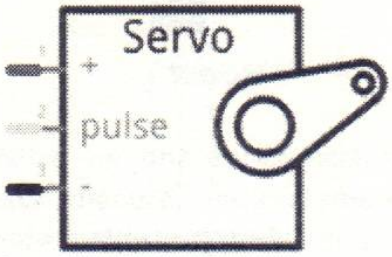


Componente

Símbolo

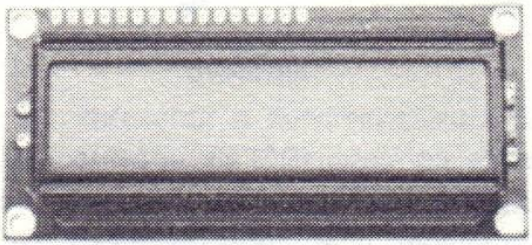
11) **Servomotor**. En un dispositivo similar a un motor DC que tiene la capacidad de ubicarse en cualquier posición dentro de su rango de operación y mantenerse estable en tal posición. Este dispositivo tiene la capacidad de ser controlado en velocidad y posición.

Dispositivo



Configuración de Pines

12) **Pantalla LCD**. Una pantalla de cristal líquido o LCD (siglas en 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.



Dispositivo

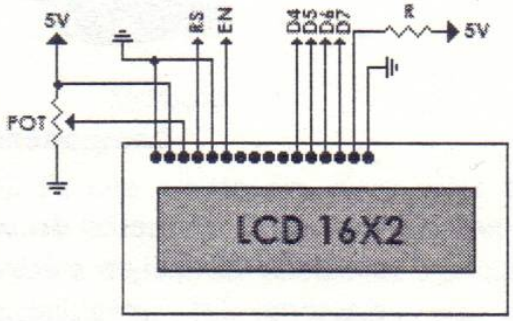
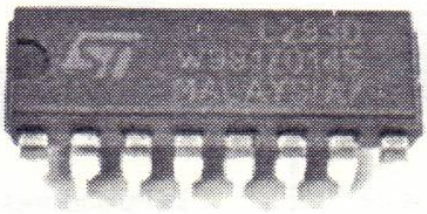
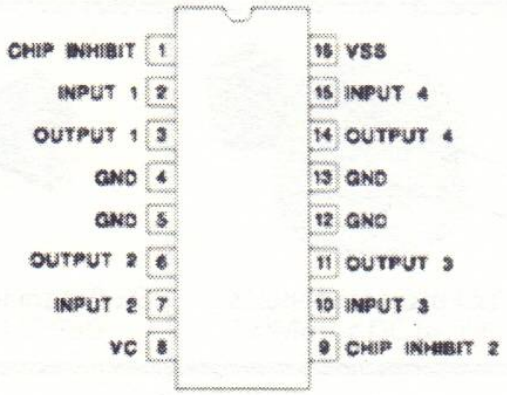


Diagrama de conexión

13) **Puente H L293D**. Un puente H es un circuito electrónico que permite a un motor eléctrico DC girar en ambos sentidos, avanzar y retroceder. 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.



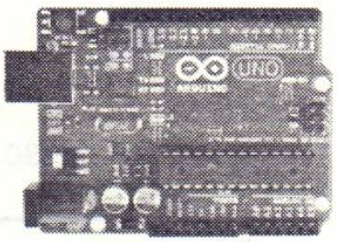
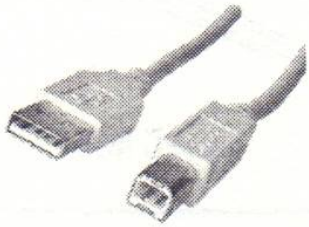
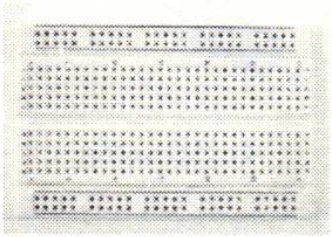
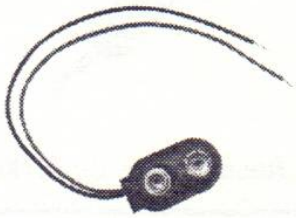
Dispositivo

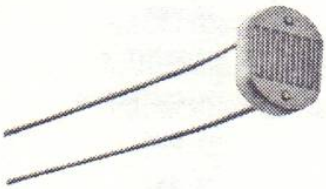
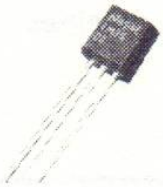
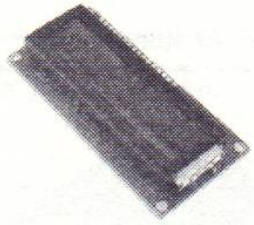


Configuración de Pines

MATERIALES NECESARIOS

Para el desarrollo de las prácticas, se dispone de un conjunto de componentes electrónicos para iniciarse en el mundo de Arduino. Diseñada para principiantes, cuenta con todo lo necesario para desarrollar programas que ayudan al aprendizaje y manejo de Arduino.

 Arduino UNO	 Cable USB tipo AB	 Protoboard
 1x Batería de 9 voltios	 1x Porta-baterías de 9V	 Cables

 1x Foto-resistencias	 3x Potenciómetros de 10 KΩ	 3x Pulsadores
 1x Sensor de temperatura LM35	 4x LED rojo	 4x LED amarillo
 4x LED verde	 2x LED RGB	 12x Resistencias de 330 Ω
 3x Resistencias de 1 kΩ	 4x Resistencias de 4,7KΩ	 1x Display LCD 16x2

 1x Speaker	 1x Puente-H L293D	 1x Motor DC (6-9V)
 1x Servomotor		

6 PRÁCTICAS

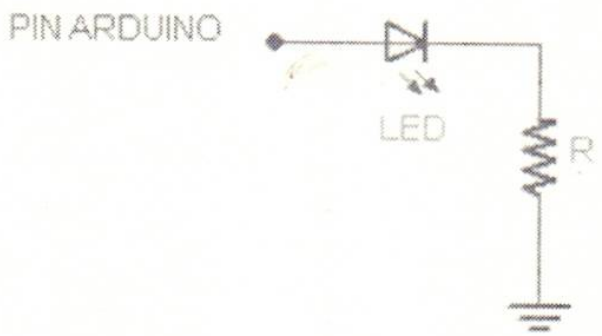
PRIMERA PRÁCTICA (PUERTOS DIGITALES DE SALIDA)

En esta primera práctica se aprenderá cómo encender diodos LED, cómo desarrollar el encendido de LED por desplazamiento y cómo simular un semáforo en tiempos cortos. Además, se aprenderá un poco de electrónica básica, declaración de variables, asignación de puertos y programación en C, y lo más importante se utilizarán funciones básicas y principales del entorno Arduino.

1.1 LED intermitente

En este primer ejemplo realizaremos lo que se ha hecho en un capítulo anterior "**Cargar mi primer ejemplo**", es decir, parpadear un LED "**Blink**". Sin embargo, conectaremos un diodo LED al puerto 10 de nuestro Arduino.

El circuito para este ejemplo es muy sencillo, simplemente se trata de una resistencia y un diodo LED conectados en serie como se puede apreciar en el circuito.



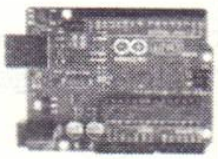
Para cerrar el circuito, el pin positivo del diodo LED va conectado a un puerto digital de la tarjeta Arduino y el otro extremo de la resistencia va conectado a tierra o GND. El valor de la resistencia es de 330 Ω para un diodo LED rojo, y se calcula a través de la fórmula de resistencia.

$$R = \frac{V_{5V} - V_{LED}}{I}$$

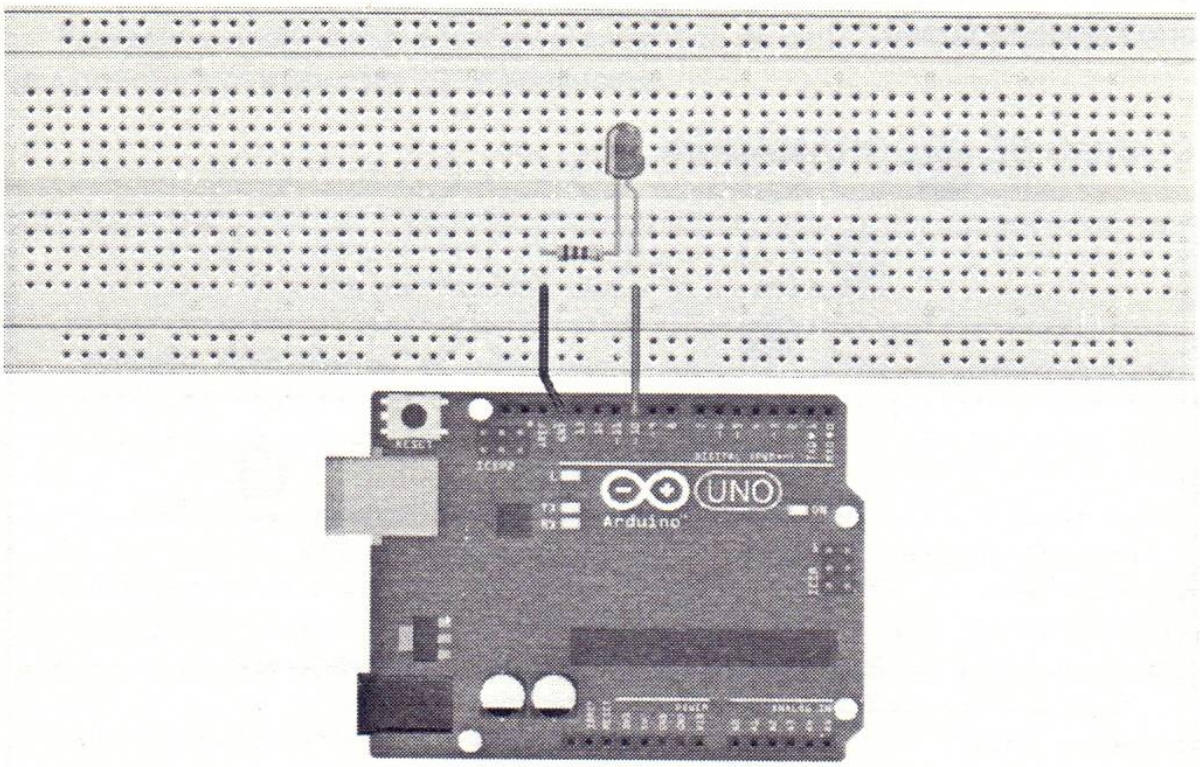
Donde V_{5V} es el voltaje de la tarjeta Arduino, V_{LED} es la caída de voltaje en el diodo LED, e I es la corriente del diodo LED.

El voltaje de alimentación para este circuito es de 5 V. Por lo tanto, el voltaje de cada pin digital de la tarjeta Arduino es de 5 V. El diodo se enciende cuando el voltaje es 5 V y se apaga cuando es 0 V.

MATERIALES

				
Arduino UNO	Protoboard	1 LED rojo	1 Resistencia de 330 Ω	Cables

CIRCUITO



DESCRIPCIÓN GENERAL DE LA PROGRAMACIÓN

Para el desarrollo del programa declararemos el puerto 10 de nuestra placa Arduino como una variable `int`:

```
int ledPin = 10;
```

Dentro de la función principal `void` **`setup`**`()` configuramos el puerto 10 como salida. Es necesario hacer funcionar el diodo LED con una señal digital de 0 V (LED apagado) y 5 V (LED encendido):

```
pinMode(ledPin, OUTPUT);
```

Dentro de la función principal `void` **`loop`**`()` escribimos en el puerto digital los estados lógicos para que empiece a parpadear el LED. Para encender el LED en el puerto digital se escribe `HIGH` para el estado lógico 1 (5 V), y para apagar el LED se escribe `LOW` para el estado lógico 0 (0 V):

```
digitalWrite(ledPin, HIGH);    // Encender LED
delay(1000);                   // Retardo de 1 segundo.
digitalWrite(ledPin, LOW);     // Apagar LED
delay(1000);                   // Retardo de 1 segundo.
```

La función de tiempo `delay(1000)` permite retardar el programa un segundo. Esta función permite que el LED durante un segundo se mantenga encendido y por otro segundo apagado.

CÓDIGO DE PROGRAMACIÓN

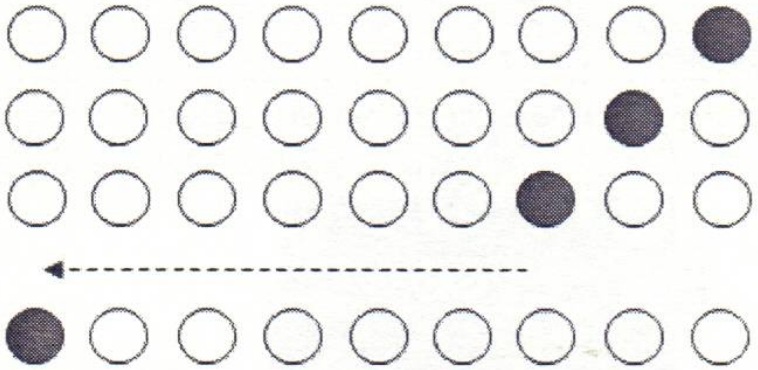
```
/*****************************************************************
*                Practica  1.1 - LED Intermitente                *
*****************************************************************
Este programa permite realizar el parpadeo de un led cada segundo.
*****************************************************************/
int ledPin = 10;                // Declaramos el Puerto digital 10.

void setup() { // Se ejecuta cada vez que el Arduino se inicia.
  pinMode(ledPin, OUTPUT);      // Configuramos el PIN 10 como salida
}

void loop() { // Bucle repetitivo principal del programa.
  digitalWrite(ledPin, HIGH); // Encendemos el LED
  delay(1000);                  // Colocamos un retardo para mantener
encendido el led durante 1 segundo.
  digitalWrite(ledPin, LOW);  // Apagamos el LED
  delay(1000);                  // Colocamos un retardo para mantener
apagado el led durante 1 segundo.
}
```

Ahora hacemos clic en **Verificar** en la parte superior del entorno Arduino para asegurarnos de que no hay errores en el código, y por último pulsamos en el botón **Cargar** para subir el código a nuestra placa Arduino.

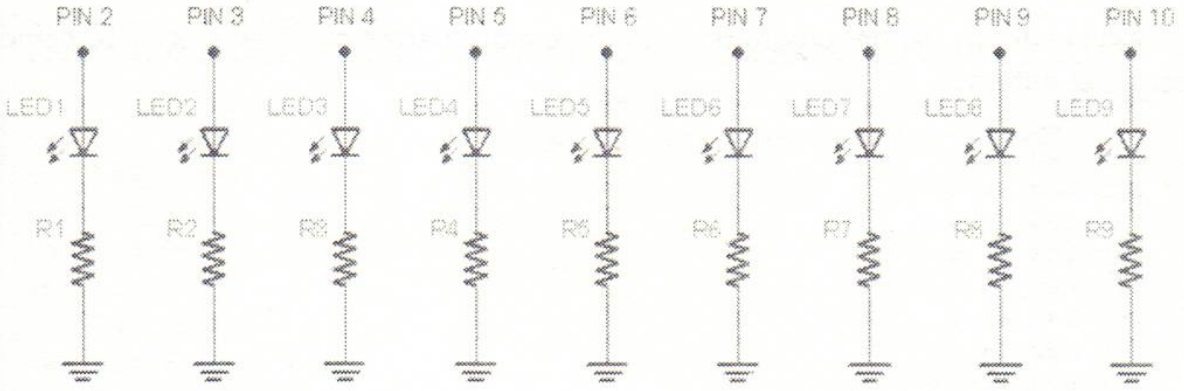
Al terminar con el programa tendremos como resultado un LED que parpadea cada segundo, es decir, que se mantiene encendido durante un segundo y apagado durante otro segundo.

1.2 Desplazamiento de LED

Para este ejemplo vamos a realizar el encendido de los LED por desplazamiento de una posición a otra por cada 200 milisegundos.



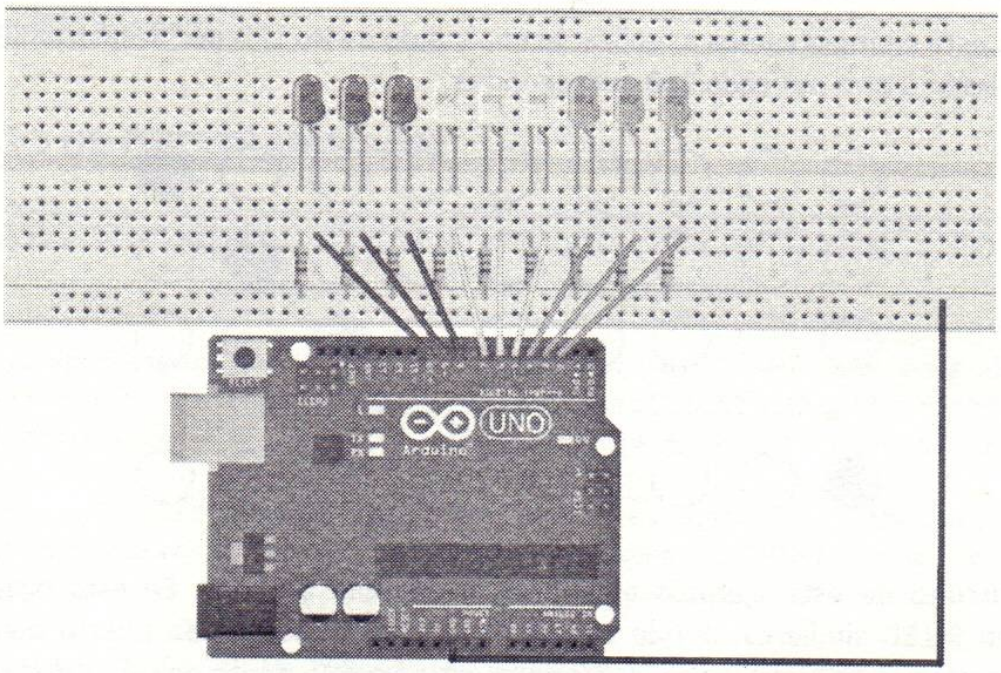
El circuito de este ejemplo es similar al ejemplo anterior. En esta ocasión se armarán 9 LED similares, donde cada LED va conectado a cada puerto digital del Arduino y todas las resistencias van conectadas a un solo punto de referencia o GND.



MATERIALES

				
Arduino UNO	Protoboard	9 LED (rojo, verde y azul)	9 Resistencias de 330 Ω	Cables

CIRCUITO



DESCRIPCIÓN GENERAL DE LA PROGRAMACIÓN

Para el desarrollo del programa declaramos los puertos digitales 2, 3,..., 10 como variables enteras:

```
int Led_2= 2;
int Led_3= 3;
int Led_4= 4;
int Led_5= 5;
int Led_6= 6;
int Led_7= 7;
int Led_8= 8;
int Led_9= 9;
int Led_10= 10;
```

Dentro de la función principal `void setup()` configuramos todos los puertos como salidas:

```
pinMode(Led_2, OUTPUT);
pinMode(Led_3, OUTPUT);
pinMode(Led_4, OUTPUT);
pinMode(Led_5, OUTPUT);
pinMode(Led_6, OUTPUT);
pinMode(Led_7, OUTPUT);
pinMode(Led_8, OUTPUT);
pinMode(Led_9, OUTPUT);
pinMode(Led_10, OUTPUT);
```

Para el encendido de los LED por desplazamiento dentro de la función principal `void loop()` escribimos en cada uno de los puertos digitales el estado lógico 1 (5 V)

HIGH durante cada 200 milisegundos. Durante cada desplazamiento, el LED anterior se apaga con un estado lógico 0 (0 V) LOW, por ejemplo:

```
/* Durante este segmento de código enciende el LED 5 y apaga el
   LED 4. En un segmento de código anterior el LED 4 fue encendido.
*/
digitalWrite(Led_4, LOW);       // LED 4 apagado
digitalWrite(Led_5, HIGH);      // LED 5 encendido
delay(200);                     // Espera de 0,2 segundos

/* Durante los próximos 200ms se encenderá el LED 6 y se apagará el
   LED 5. */
digitalWrite(Led_5, LOW);       // LED 5 apagado
digitalWrite(Led_6, HIGH);      // LED 6 encendido
delay(200);                     // Espera de 0,2 segundos

...etc.
```

CÓDIGO DE PROGRAMACIÓN

```
/*****************************************************************
*              Práctica 1.2- Desplazamiento de LED               *
*****************************************************************
Este programa permite realizar el encendido de 9 LED por
secuencia de posición.
*****************************************************************/
int Led_2= 2;                 // Declaración de los puertos digitales
int Led_3= 3;                 // para 9 LED (desde D2 hasta D10).
int Led_4= 4;
int Led_5= 5;
int Led_6= 6;
int Led_7= 7;
int Led_8= 8;
int Led_9= 9;
int Led_10= 10;

void setup() {
  pinMode(Led_2, OUTPUT);     // Configuración de puertos digitales
  pinMode(Led_3, OUTPUT);     // como salidas.
  pinMode(Led_4, OUTPUT);
  pinMode(Led_5, OUTPUT);
  pinMode(Led_6, OUTPUT);
  pinMode(Led_7, OUTPUT);
  pinMode(Led_8, OUTPUT);
  pinMode(Led_9, OUTPUT);
  pinMode(Led_10, OUTPUT);
}
```

```
void loop() {
  digitalWrite(Led_10, LOW);      // LED 10 apagado
  digitalWrite(Led_2, HIGH);      // LED 2 encendido
  delay(200);                     // espera de 0,2 segundos

  digitalWrite(Led_2, LOW);       // LED 2 apagado
  digitalWrite(Led_3, HIGH);      // LED 3 encendido
  delay(200);                     // espera de 0,2 segundos

  digitalWrite(Led_3, LOW);       // LED 3 apagado
  digitalWrite(Led_4, HIGH);      // LED 4 encendido
  delay(200);                     // espera de 0,2 segundos

  digitalWrite(Led_4, LOW);       // LED 4 apagado
  digitalWrite(Led_5, HIGH);      // LED 5 encendido
  delay(200);                     // espera de 0,2 segundos

  digitalWrite(Led_5, LOW);       // LED 5 apagado
  digitalWrite(Led_6, HIGH);      // LED 6 encendido
  delay(200);                     // espera de 0,2 segundos

  digitalWrite(Led_6, LOW);       // LED 6 apagado
  digitalWrite(Led_7, HIGH);      // LED 7 encendido
  delay(200);                     // espera de 0,2 segundos

  digitalWrite(Led_7, LOW);       // LED 7 apagado
  digitalWrite(Led_8, HIGH);      // LED 8 encendido
  delay(200);                     // espera de 0,2 segundos

  digitalWrite(Led_8, LOW);       // LED 8 apagado
  digitalWrite(Led_9, HIGH);      // LED 9 encendido
  delay(200);                     // espera de 0,2 segundos

  digitalWrite(Led_9, LOW);       // LED 9 apagado
  digitalWrite(Led_10, HIGH);     // LED 10 encendido
  delay(200);                     // espera de 0,2 segundos
}
```

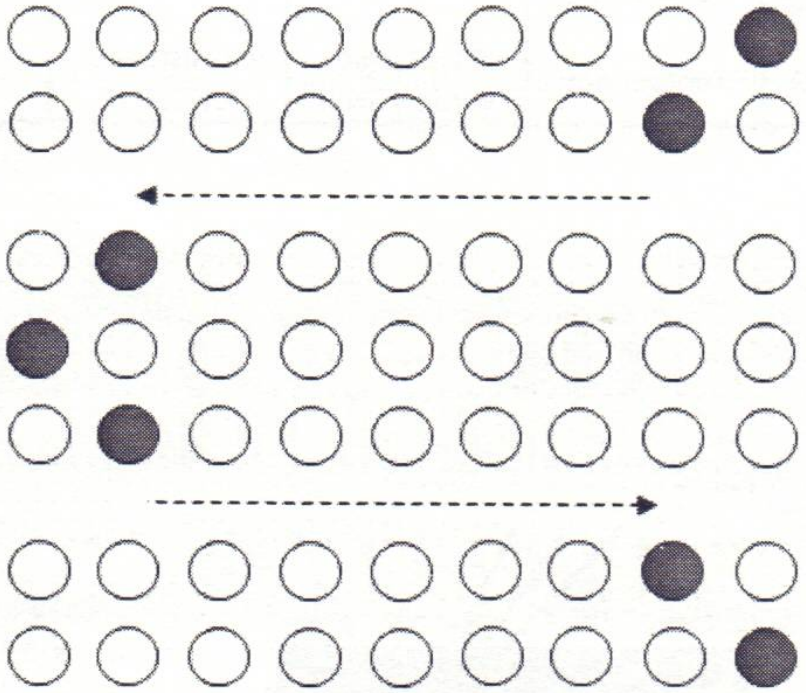
Ahora hacemos clic en **Verificar** en la parte superior del entorno Arduino para asegurarse de que no hay errores en el código, y por último pulsamos en el botón **Cargar** para subir el código a nuestra placa Arduino.

Cuando finalicemos con este programa lo que observaremos es el desplazamiento de encendido y apagado de los LED desde el puerto 2 hasta el puerto 10.

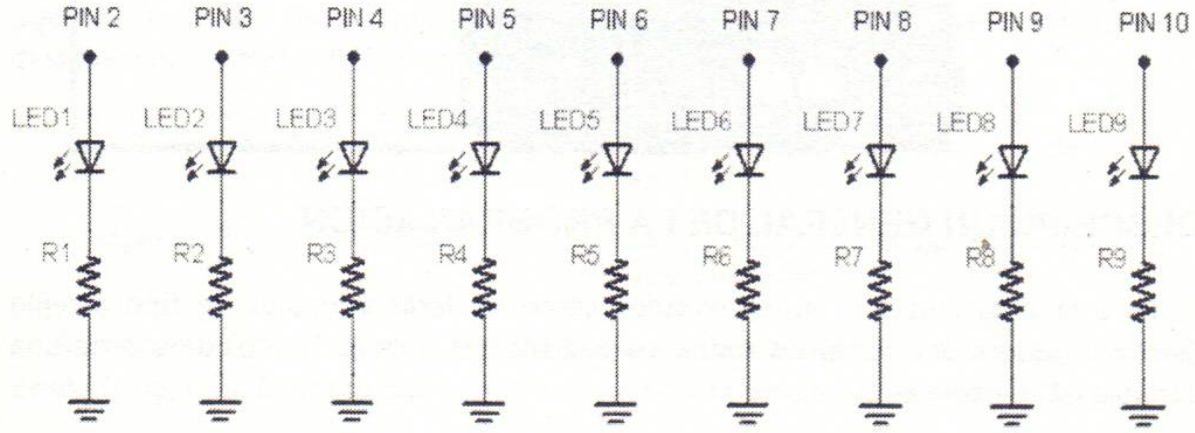
Si se ha realizado todo bien, el encendido de cada LED es un tiempo de 200 ms.

1.3 Luces del coche fantástico

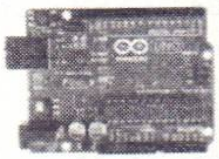
Para nuestro tercer programa vamos a realizar el efecto de encendido de luces del coche fantástico. Este ejemplo completa la secuencia del ejemplo anterior, es decir, realiza el desplazamiento por posición ascendente y descendente.



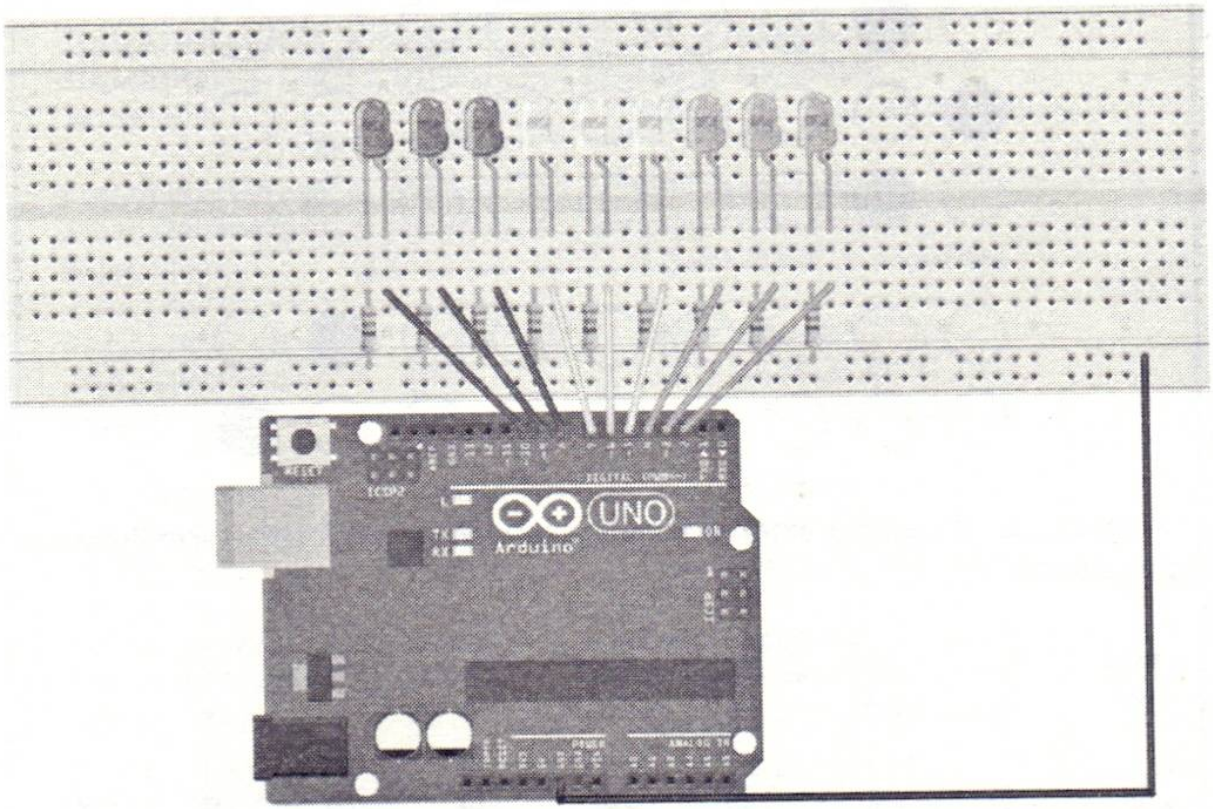
Al igual que el ejemplo anterior vamos a utilizar 9 diodos LED, y crearemos el mismo circuito.



MATERIALES

				
Arduino UNO	Protoboard	9 LED (rojo, verde y azul)	9 Resistencias de 330 Ω	Cables

CIRCUITO



DESCRIPCIÓN GENERAL DE LA PROGRAMACIÓN

Para la programación, aprenderemos cómo declarar variables de tipo arreglo "array". Primero declararemos todos los puertos digitales 2, 3,..., 10 dentro de una variable entera tipo array:

```
int ledsPin[] = {2, 3, 4, 5, 6, 7, 8, 9, 10};
```

Es importante recordar que las posiciones dentro de una variable array empiezan desde el 0, es decir, en la posición 0 se encuentra alojado el valor entero 2, en la

posición 1 el valor 3, en la posición 2 el valor 4, etc. Este modo de declaración nos permite reducir líneas de código.

Declaramos una variable entera para retardar el desplazamiento durante cada 65 milisegundos:

```
int ledDelay(65);
```

Declaramos una variable `int` para definir la dirección de desplazamiento de los LED (ascendente o descendente). Esta variable se inicializa en 1 para empezar con el desplazamiento en dirección ascendente:

```
int direction = 1;
```

Declaramos una variable `int` para el desplazamiento de los LED por posición:

```
int DireccionLed = 0;
```

Declaramos una variable `long` para el conteo del tiempo en milisegundos:

```
unsigned long Tiempo_de_cambio;
```

Este tipo de declaración de variables `unsigned` permite aprovechar todo el rango solo para valores positivos. Si `long` tiene un rango de -2,147,483,648 a 2,147,483,647, `unsigned long` tiene un rango de 4,294,967,295.

Dentro de la función principal `void setup()` configuramos todos los puertos como salidas a través del bucle `for (int x=0; x<10; x++)`, donde `x` es una variable que permite al bucle ejecutar 9 veces, es decir, permite configurar los 9 puertos a través de `x`. También inicializamos un tiempo para empezar el desplazamiento de los LED a través de la función `millis()`:

```
for (int x=0; x<9; x++) {
  pinMode(ledsPin[x], OUTPUT); // Configuración de puertos de
                               // salida.
}
Tiempo_de_cambio = millis();   // Leer tiempo inicial de
                               // desplazamiento.
```

Recordemos que el primer valor entero en la variable array `ledsPin[x]` empieza desde la posición 0, por lo que el bucle `for` empieza desde 0 y termina en 9.

Dentro de la función principal `void loop()` utiliza la función `millis()` para la lectura de tiempos en milisegundos. Dentro de la estructura lógica `if ((millis() - Tiempo_de_cambio) > ledDelay)` se realiza el desplazamiento de los LED durante

cada 65 ms, donde el valor de `Tiempo_de_cambio` es el último tiempo transcurrido por un ciclo de trabajo anterior. En el bucle `for` apagamos todos los LED durante la ejecución de la estructura lógica `digitalWrite(ledsPin[x], LOW)`.

Para el desplazamiento de los LED la operación `DireccionLed += direction` permite incrementar la posición de encendido, pero cuando este llega hasta 9 `if(DireccionLed == 9)` invierte la dirección de encendido de ascendente a descendente `direction = -1`. Cuando decrementa `DireccionLed += direction` la posición de encendido hasta llegar a 0 `if(DireccionLed == 0)` nuevamente invierte la dirección de encendido de descendente a ascendente `direction = 1`. Y nuevamente se repite el proceso inicial:

```
if ((millis() - Tiempo_de_cambio) > ledDelay) { /* Tiempo
      transcurrido es mayor que LedDelay, Define el tiempo para el
      encendido de cada LED según su posición y dirección. */
  for (int x=0; x<9; x++) {
     digitalWrite(ledsPin[x], LOW);          // Apagar todos los LED.
  }
  digitalWrite(ledsPin[DireccionLed], HIGH);// Encender LED.
  DireccionLed += direction;                 /* Incrementar cada
                                          posición de encendido de LED. */
  if (DireccionLed == 9) direction = -1;     /* Cambio descendente de
                                              la dirección de LED. */
  if (DireccionLed == 0) direction = 1;      /* Cambio ascendente de
                                              la dirección de LED. */
  Tiempo_de_cambio = millis();               /* Actualización de
                                              tiempo.  *!
}
```

Para este ejemplo, por primera vez utilizamos la secuencia repetitiva `for` para declarar los puertos como salida y apagar los LED, y evitarnos demasiadas líneas de código como en el ejemplo anterior. Y por primera vez usamos la función de tiempo `millis()` para leer el tiempo de ejecución del programa en milisegundos.

CÓDIGO DE PROGRAMACIÓN

```
/*****************************************************************
*            Práctica 1.3 - Luces del coche fantástico           *
*****************************************************************
Este programa permite realizar el efecto de luces interactivas del
coche fantástico.
*****************************************************************/

int ledsPin[] = {2, 3, 4, 5, 6, 7, 8, 9, 10};  // Declaramos un
vector de tamaño 9 para los puertos digitales (del 2 hasta 10).
```

```
int ledDelay(65);                                    // Declaramos un
retardo para cada cambio de LED.
int direction = 1;                                   // Declaramos la
dirección para una secuencia de encendido de LED.
int DireccionLed = 0;                                // Declaramos una
variable para el cambio de posición de LED para cada dirección.
unsigned long Tiempo_de_cambio;                      // Declaramos una
variable de conteo de tiempo para el cambio de LED.

void setup() {
  for (int x=0; x<9; x++) {
    pinMode(ledsPin[x], OUTPUT);                     // Configuración de
puertos de salida.
  }
Tiempo_de_cambio = millis();                         // Leemos un tiempo
inicial.
}

void loop() {
// si ha sido ledDelay ms desde el último cambio
if ((millis() - Tiempo_de_cambio) > ledDelay) {// Tiempo transcurrido
es mayor que LedDelay. Define el tiempo para el encendido de cada LED
según su posición y dirección.
   for (int x=0; x<9; x++) {
      digitalWrite(ledsPin[x], LOW);             // Apagamos todos los
LEDs.
   }
   digitalWrite(ledsPin[DireccionLed], HIGH);// Encendemos los LED.
   DireccionLed += direction;                        // Incrementamos cada
posición de encendido de LEDs.
   if (DireccionLed == 9) direction = -1;      // Cambio descendente de
la dirección de LEDs.
   if (DireccionLed == 0) direction = 1;       // Cambio ascendente de
la dirección de LEDs
   Tiempo_de_cambio = millis();              //  Actualización de tiempo.
 }
}
```

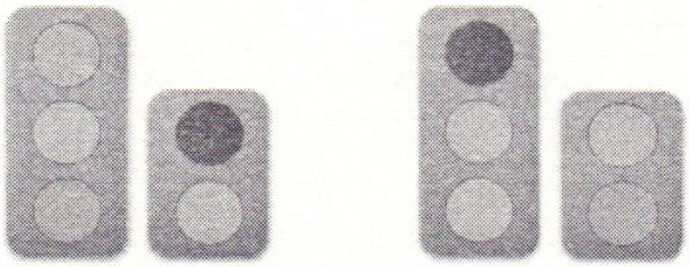
Ahora hacemos clic en el botón **Verificar** para identificar errores y por último **Cargar** el programa a nuestra placa Arduino.

Finalmente observaremos el encendido y apagado de LED por desplazamiento en ambos sentidos (ascendente y descendente) al igual que el coche fantástico.

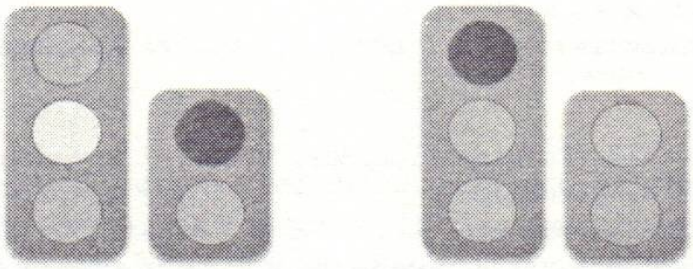
1.4 Semáforos de cruce

Para culminar con esta primera práctica, vamos a simular el funcionamiento de dos semáforos para vehículos y dos semáforos peatonales. En esta ocasión vamos a utilizar 4 diodos LED rojo, 4 LED verde y 2 LED amarillo.

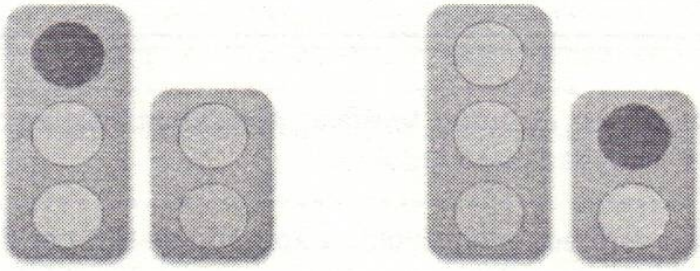
El funcionamiento de un sistema de semáforos de cruce de calles, en una instancia el primer semáforo para vehículos estará en luz verde y su semáforo peatonal en luz roja, mientras tanto el otro semáforo para vehículos estará en luz roja y su semáforo peatonal en luz verde durante tres segundos.



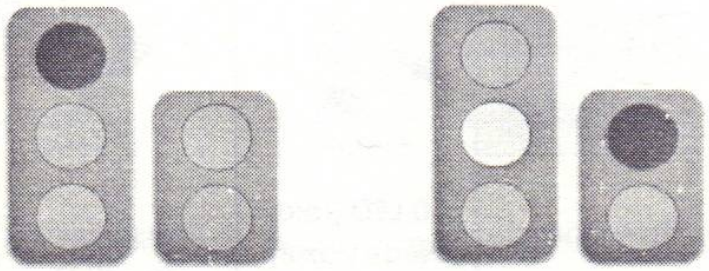
Durante el próximo segundo, la luz amarilla del primer semáforo para vehículos se encenderá, y las luces del resto de semáforos se mantienen iguales.



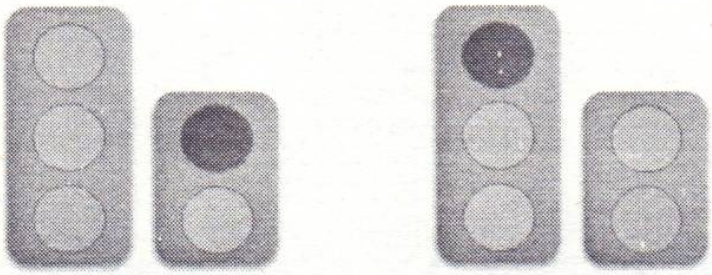
Durante los próximos tres segundos, el primer semáforo para vehículos estará en luz roja y su semáforo peatonal en luz verde, mientras tanto el otro semáforo de vehículos estará en luz verde y su semáforo peatonal en luz roja.



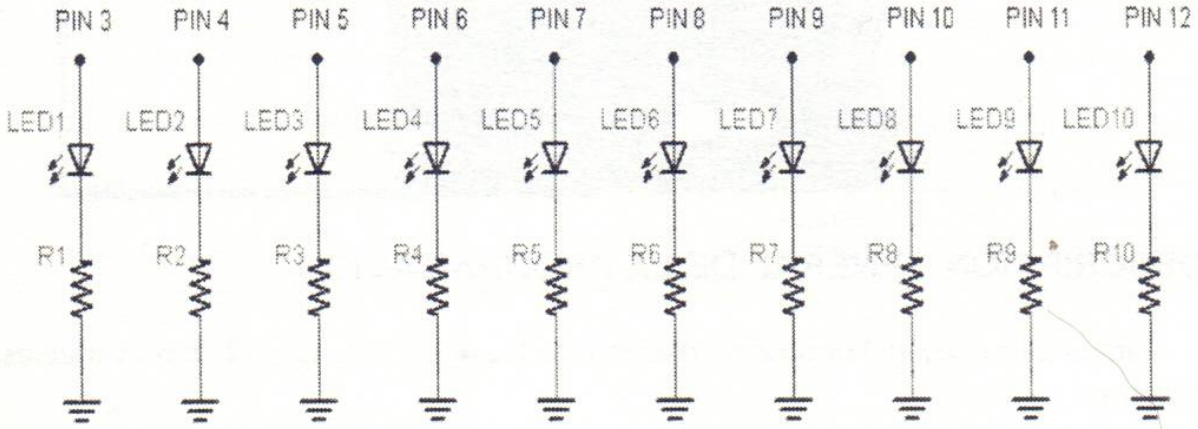
Después de otro segundo, el segundo semáforo para vehículos estará en luz amarilla y el resto de semáforos se mantendrán iguales.



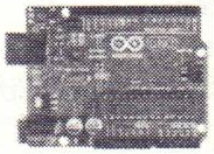
Finalmente, los semáforos volverán a estar en su estado inicial. El primer semáforo para vehículos estará en luz verde y su semáforo peatonal en luz roja, y mientras tanto el otro semáforo para vehículos estará en luz roja y su semáforo peatonal en luz verde. Nuevamente se repite la secuencia.



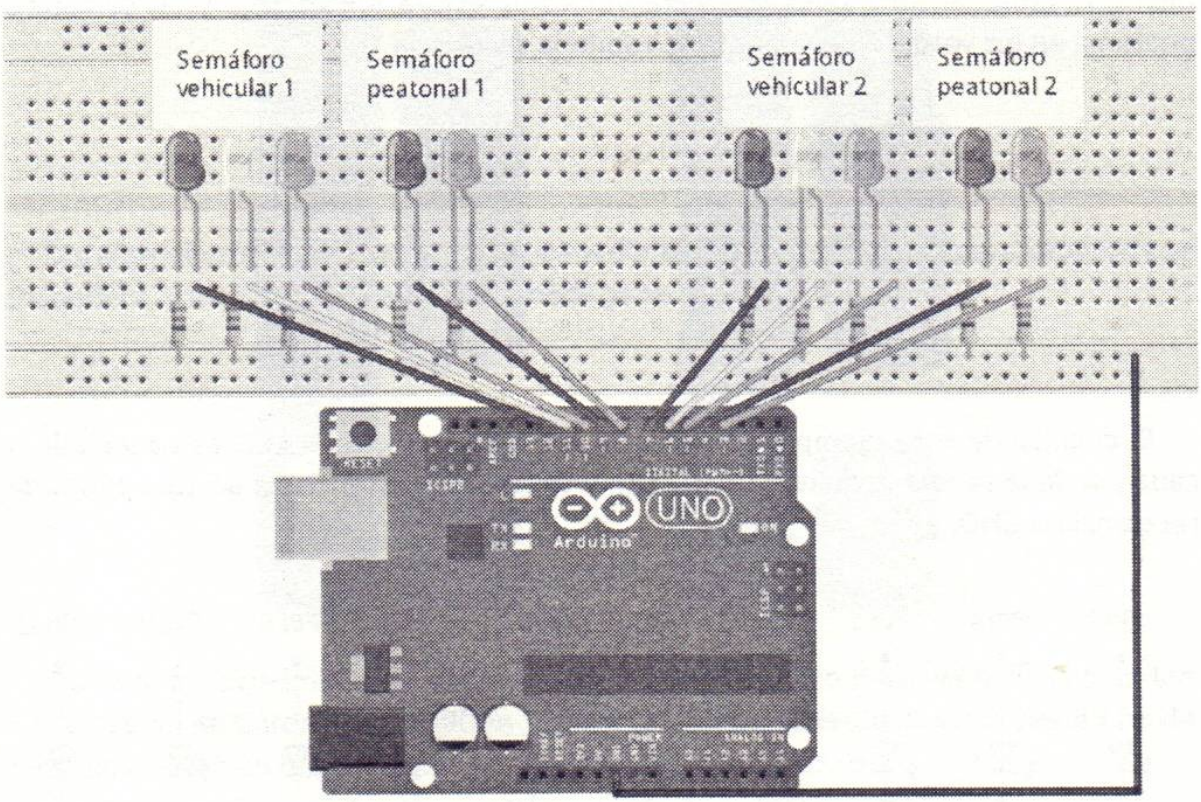
El circuito de este ejemplo es similar a los anteriores, cada LED es conectado a cada pin de la tarjeta Arduino (3, 4,..., 12), y todas las resistencias a un solo punto de referencia o GND.



MATERIALES

				
Arduino UNO	Protoboard	10 LED (rojo, verde y azul)	10 Resistencias de 330 Ω	Cables

CIRCUITO



DESCRIPCIÓN GENERAL DE LA PROGRAMACIÓN

Para el desarrollo del programa, declaramos los puertos 3, 4,..., 12 como variables enteras:

```
int Semaforo_Rojo_1    = 12;//Puerto 12 para luz roja semáforo 1.
int Semaforo_Amarillo_1= 11;//Puerto 11 para luz amarilla semáforo 1
int Semaforo_Verde_1   = 10;//Puerto 10 para luz verde semáforo 1.
```

```
int Peatonal_Rojo_1    = 9; //Puerto 9 para luz roja peatonal 1.
int Peatonal_Verde_1   = 8; //Puerto 8 para luz verde peatonal 1.

int Semaforo_Rojo_2    = 7; //Puerto 7 para luz roja semáforo 2.
int Semaforo_Amarillo_2= 6; //Puerto 6 para luz amarilla semáforo 2.
int Semaforo_Verde_2   = 5; //Puerto 5 para luz verde semáforo 2.

int Peatonal_Rojo_2    = 4; //Puerto 4 para luz roja peatonal 2.
int Peatonal_Verde_2   = 3; //Puerto 3 para luz verde peatonal 2.
```

Dentro de la función principal **setup**() configuramos los puertos como salidas digitales (OUTPUT) a través de la función pinMode:

```
pinMode(Semaforo_Rojo_1,      OUTPUT);
pinMode(Semaforo_Amarillo_1, OUTPUT);
pinMode(Semaforo_Verde_1,     OUTPUT);
pinMode(Peatonal_Rojo_1,      OUTPUT);
pinMode(Peatonal_Verde_1,     OUTPUT);
pinMode(Semaforo_Rojo_2,      OUTPUT);
pinMode(Semaforo_Amarillo_2, OUTPUT);
pinMode(Semaforo_Verde_2,     OUTPUT);
pinMode(Peatonal_Rojo_2,      OUTPUT);
pinMode(Peatonal_Verde_2,     OUTPUT);
```

En la función principal void **loop**() programamos todo el funcionamiento del sistema descrito anteriormente escribiendo en los puertos digitales los niveles lógicos correspondientes para encender o apagar los LED a través de la función digitalWrite. Para encender un LED en el puerto digital, se escribe HIGH para 5 V, y para apagar un LED se escribe LOW para 0 V, por ejemplo:

```
// Primer proceso de encendido de luces de los semáforos.
digitalWrite(Semaforo_Verde_1,   HIGH); /* LED semáforo verde 1
                                           encendido. */
digitalWrite(Peatonal_Rojo_1,    HIGH); /* LED peatonal rojo 1
                                           encendido. */
digitalWrite(Semaforo_Rojo_2,    HIGH); /* LED semáforo rojo 2
                                           encendido. */
digitalWrite(Peatonal_Verde_2,   HIGH); /* LED peatonal verde 2
                                           endencido. */
delay(3000);                           // Espera de 3 segundos.

... etc.
```

CÓDIGO DE PROGRAMACIÓN

```
/*********************************************************************
*                Práctica 1.4 - Luces de semáforos                   *
*********************************************************************
Este programa permite realizar el efecto de encendido de un
sistema de semáforos de cruce.
*********************************************************************/
int Semaforo_Rojo_1     = 12; // Asignación del puerto digital 12
para luz roja semáforo 1.
int Semaforo_Amarillo_1 = 11; // Asignación del puerto digital 11
para luz amarilla semáforo 1.
int Semaforo_Verde_1    = 10; // Asignación del puerto digital 10
para luz verde semáforo 1.

int Peatonal_Rojo_1     = 9;  // Asignación del puerto digital 9 para
luz roja peatonal 1.
int Peatonal_Verde_1    = 8;  // Asignación del puerto digital 8 para
luz verde peatonal 1.

int Semaforo_Rojo_2     = 7;  // Asignación del puerto digital 7 para
luz roja semáforo 2.
int Semaforo_Amarillo_2 = 6;  // Asignación del puerto digital 6 para
luz amarilla semáforo 2.
int Semaforo_Verde_2    = 5;  // Asignación del puerto digital 5 para
luz verde semáforo 2.

int Peatonal_Rojo_2     = 4;  // Asignación del puerto digital 4 para
luz roja peatonal 2.
int Peatonal_Verde_2    = 3;  // Asignación del puerto digital 3 para
luz verde peatonal 2.

void setup() {
  // Configuración de los puertos digitales como salida.
  pinMode(Semaforo_Rojo_1,     OUTPUT);
  pinMode(Semaforo_Amarillo_1, OUTPUT);
  pinMode(Semaforo_Verde_1,    OUTPUT);
  pinMode(Peatonal_Rojo_1,     OUTPUT);
  pinMode(Peatonal_Verde_1,    OUTPUT);
  pinMode(Semaforo_Rojo_2,     OUTPUT);
  pinMode(Semaforo_Amarillo_2, OUTPUT);
  pinMode(Semaforo_Verde_2,    OUTPUT);
  pinMode(Peatonal_Rojo_2,     OUTPUT);
  pinMode(Peatonal_Verde_2,    OUTPUT);
}
```

```
void loop() {
  digitalWrite(Semaforo_Verde_1,    HIGH); // Led semáforo verde 1
encendido.
  digitalWrite(Peatonal_Rojo_1,     HIGH); // Led peatonal rojo 1
encendido.
  digitalWrite(Semaforo_Rojo_2,     HIGH); // Led semáforo rojo 2
encendido.
  digitalWrite(Peatonal_Verde_2,    HIGH); // Led peatonal verde 2
encendido.
  delay(3000);                             // Espera de 3 segundos.

  digitalWrite(Semaforo_Verde_1,     LOW); // Led semáforo verde 1
apagado.
  digitalWrite(Semaforo_Amarillo_1,HIGH); // Led semáforo amarillo 1
encendido.
  delay(1000);                             // Espera de 1 segundo.

  digitalWrite(Semaforo_Amarillo_1, LOW); // Led semáforo amarillo 1
apagado.
  digitalWrite(Peatonal_Rojo_1,      LOW); // Led peatonal rojo 1
apagado.
  digitalWrite(Semaforo_Rojo_2,      LOW); // Led semáforo rojo 2
apagado.
  digitalWrite(Peatonal_Verde_2,     LOW); // Led peatonal verde 2
apagado.

  digitalWrite(Semaforo_Rojo_1,     HIGH); // Led semáforo rojo 1
encendido.
  digitalWrite(Peatonal_Verde_1,    HIGH); // Led peatonal verde 1
encendido.
  digitalWrite(Semaforo_Verde_2,    HIGH); // Led semáforo verde 2
encendido.
  digitalWrite(Peatonal_Rojo_2,     HIGH); // Led peatonal rojo 2
encendido.
  delay(3000);                             // Espera de 3 segundos.

  digitalWrite(Semaforo_Verde_2,     LOW); // Led semáforo verde 2
apagado.
  digitalWrite(Semaforo_Amarillo_2,HIGH); // Led semáforo amarillo 2
encendido.
  delay(1000);                             // Espera de 1 segundo.

  digitalWrite(Semaforo_Rojo_1,      LOW); // Led semáforo rojo 1
apagado.
```

```
  digitalWrite(Peatonal_Verde_1,     LOW); // Led peatonal verde 1
apagado.
  digitalWrite(Semaforo_Amarillo_2, LOW); // Led peatonal amarillo 2
apagado.
  digitalWrite(Peatonal_Rojo_2,      LOW); // Led peatonal rojo 2
apagado.
}
```

Hacemos clic en el botón **Verificar** en la parte superior del entorno Arduino para asegurar que no hay errores en el código, y después pulsamos en el botón **Cargar** para subir el código a nuestra placa Arduino.

Si todo está bien observaremos la secuencia de encendidos de los LED de acuerdo a la descripción del ejemplo:

1. Semáforo 1 Verde, Peatonal 1 Rojo, Semáforo 2 Rojo, Peatonal 2 Verde durante tres segundos.
2. Semáforo 1 Amarillo, Peatonal 1 Rojo, Semáforo 2 Rojo, Peatonal 2 Verde durante un segundo.
3. Semáforo 1 Rojo, Peatonal 1 Verde, Semáforo 2 Verde, Peatonal 2 Rojo durante tres segundos.
4. Semáforo 1 Rojo, Peatonal 1 Verde, Semáforo 2 Amarillo, Peatonal 2 Rojo durante un segundo.
5. Repite nuevamente la secuencia.

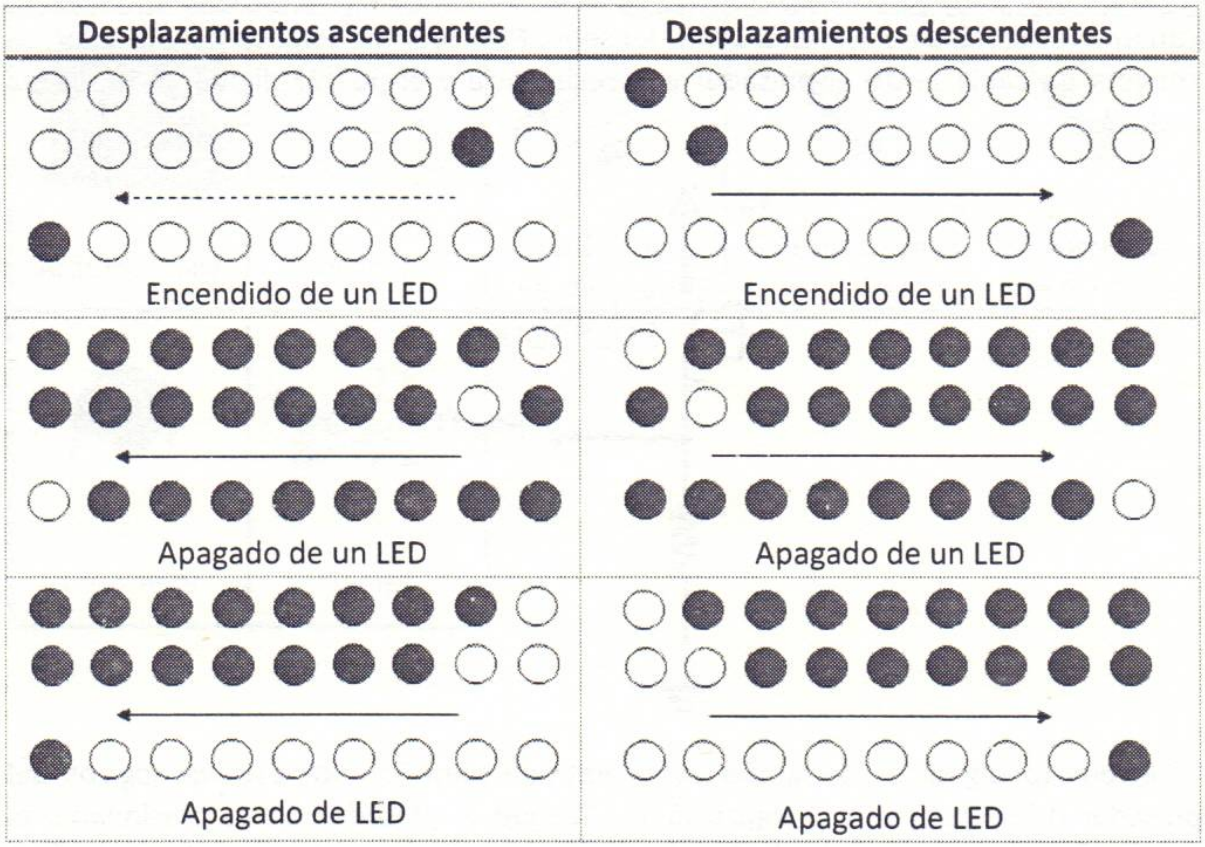
SEGUNDA PRÁCTICA (PUERTOS DIGITALES DE ENTRADA Y SEÑALES PWM)

En esta segunda práctica extenderemos la práctica anterior para interactuar por primera vez con la tarjeta Arduino, como por ejemplo, cambiar de estados a través de un pulsador, y a generar efectos luminosos combinados (rojo, verde y azul) en un solo dispositivo. Al igual que la práctica anterior se aprenderá un poco de electrónica básica, declaración de variables, operaciones matemáticas y programación en C/C++.

2.1 Órdenes de desplazamiento de LED

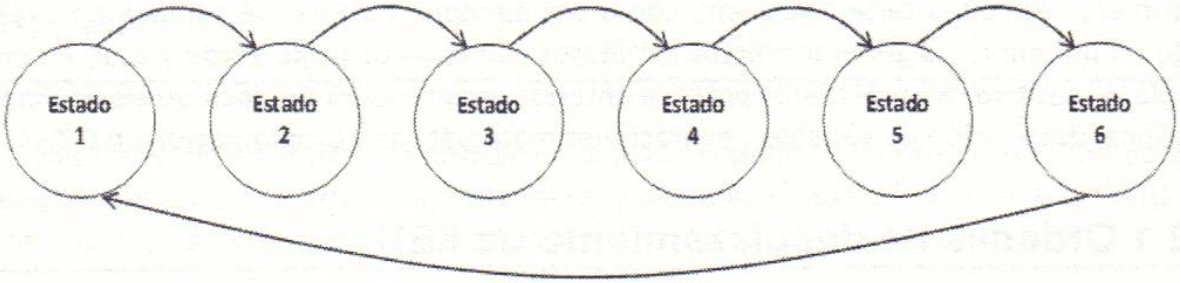
En este ejemplo realizaremos la ejecución de diferentes estilos de desplazamiento de LED por estados a través de un pulsador. Cuando se presiona un pulsador.

Estilos de desplazamiento:



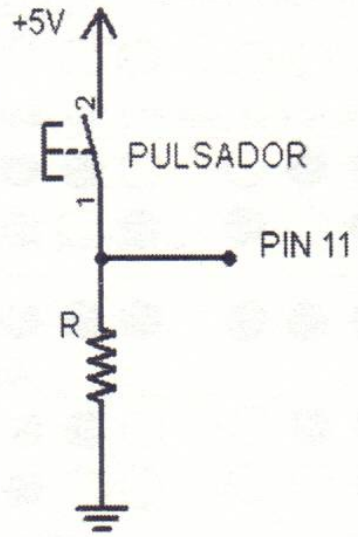
Cuando presionamos el pulsador cambiaremos de estado, en el primer estado está alojado el desplazamiento ascendente de encendido de un LED, en el segundo estado el desplazamiento descendente de encendido de un LED, en el tercer estado

el desplazamiento ascendente de apagado de un LED, en el cuarto estado el desplazamiento descendente de apagado de un LED, en el quinto estado el desplazamiento ascendente de apagado de LED, y en el sexto estado el desplazamiento descendente de apagado de LED.



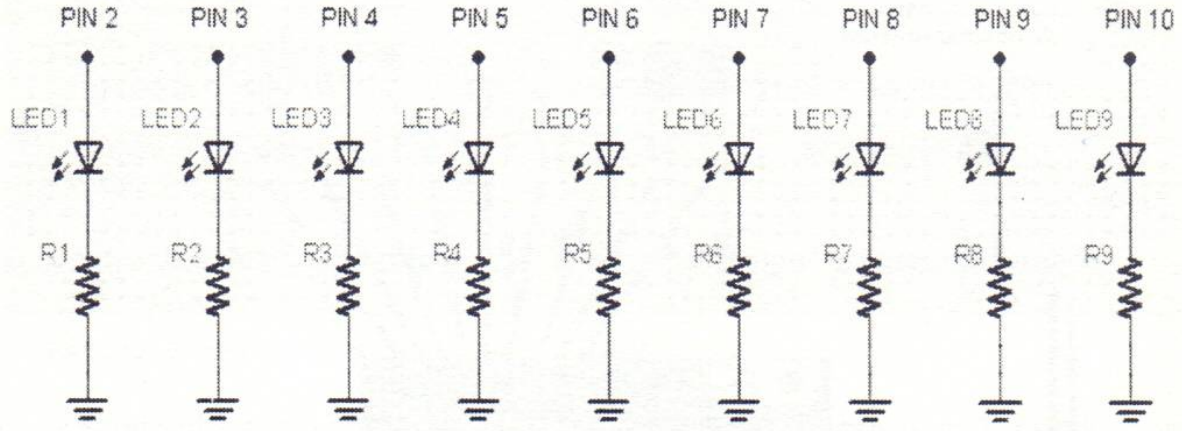
Para interactuar con la tarjeta Arduino y hacer que ejecute una acción, cambiamos el estado lógico de un pulsador, es decir, pulsamos para cambiar el estado de abierto a cerrado.

El circuito para un pulsador es muy sencillo, simplemente se conecta un pulsador con una resistencia de 4,7 kΩ a 10 kΩ en serie. Para leer los estados del pulsador, se conecta un cable entre el pulsador y la resistencia y el puerto digital de la tarjeta Arduino.

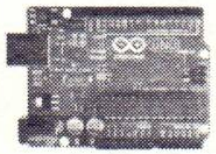


El puerto digital es declarado como entrada para leer los estados lógicos del pulsador 0 lógico (0 V) o 1 lógico (5 V). Cuando el pulsador no es presionado, el puerto digital lee un estado de 0 lógico, y cuando es presionado lee el estado de 1 lógico.

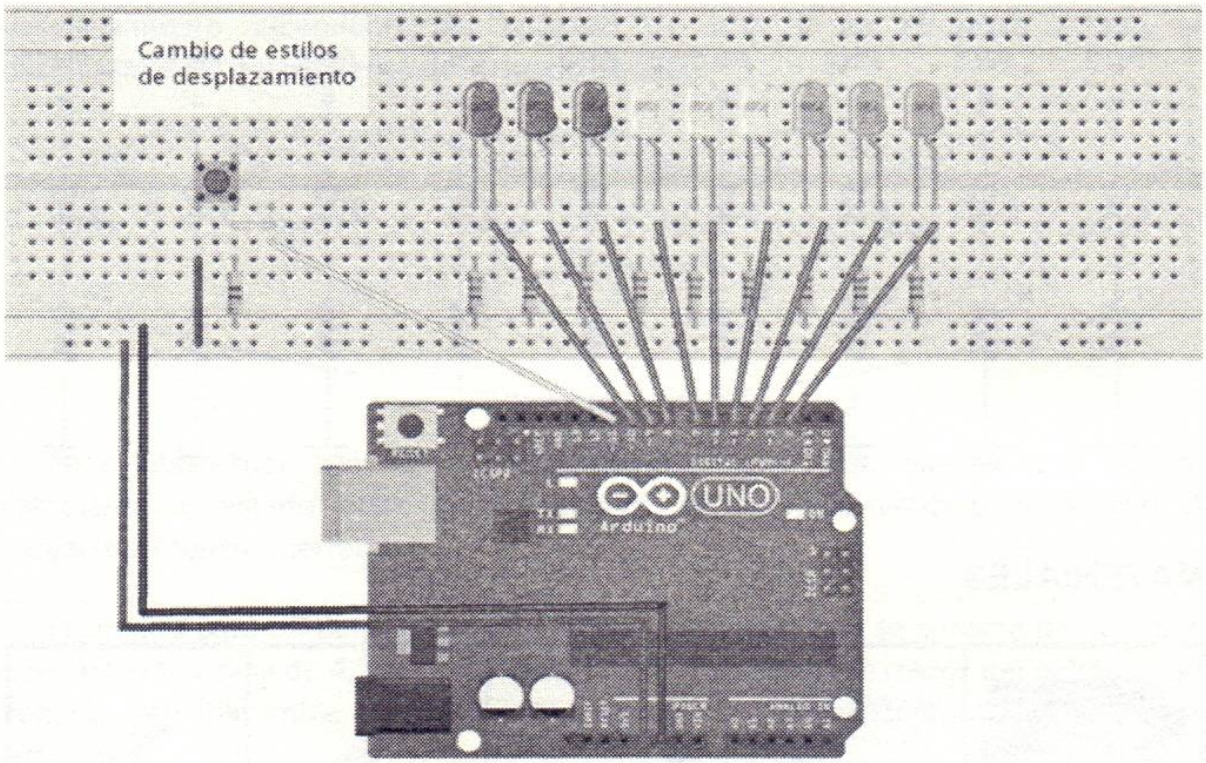
El circuito para los LED es similar a la práctica anterior. Todas las tierras van a un solo punto de referencia.



MATERIALES

 Arduino UNO	 Protoboard	 9 LED (rojo, verde y azul)	 9 Resistencias de 330 Ω	 1 Resistencia de 4,7 KΩ
 1 Pulsador	 Cables			

CIRCUITO



DESCRIPCIÓN GENERAL DE LA PROGRAMACIÓN

Para el encendido de LED mediante desplazamiento por posición durante un tiempo determinado, usamos la misma lógica de programación del ejemplo "**Luces del coche fantástico**".

Declararemos una variable array para todos los puertos digitales de salida:

```
int ledsPin[] = {2, 3, 4, 5, 6, 7, 8, 9, 10};
```

Declaramos la variable `int` para el puerto digital 11 de entrada, y declaramos una variable para la lectura de sus estados lógicos:

```
int PulsadorPin = 11;
int Estado_Pulsador = LOW;
```

Para cambiar de un estado a otro declaramos, la variable `int`. Cuando se presiona el pulsador, esta variable se incrementa en 1 para recorrer 6 estados:

```
int Estados = 0;
```

Declaramos una variable `int` para el desplazamiento de los LED por posición:

```
int Posicion = 0;
```

Dentro de la función principal `void setup()` configuramos todos los puertos como salidas para los LED a través del bucle `for`, y un puerto de entrada para el pulsador.

```
for (int x = 0; x < 9; x++){
  pinMode(ledsPin[x], OUTPUT); // Puertos digitales de salida.
}
pinMode(PulsadorPin, INPUT);    // Puerto 11 como entrada.
Tiempo_de_cambio = millis();
```

Dentro de la función principal `void loop()` leemos los estados del pulsador:

```
Estado_Pulsador = digitalRead(PulsadorPin);
```

Si el pulsador es un estado lógico 1 `HIGH`, incrementamos la variable Estados para recorrer los 6 estados del programa. Para los estados impares inicializamos la variable `Posicion = 0` para desplazamientos ascendentes, y para estados pares inicializamos `Posicion = 9` para desplazamientos descendentes:

```
if (Estado_Pulsador == HIGH){ //Leer el estado lógico 1 del pulsador
  Estados ++;      // Incrementamos al siguiente estado + 1;
  // Condición lógica para desplazamiento ascendente.
  if ((Estados == 1)||(Estados == 3)||(Estados == 5)) Posicion=0;
  // Condición lógica para desplazamiento descendente.
  else if ((Estados == 2)||(Estados == 4)||(Estados == 6))
  Posicion=9;
  else Estados = 1; // Reiniciamos el primer estado.
  delay(200);       // Retardo para eliminar rebotes en el pulsador.
}
```

La función `desplazamientos(Estados)` permite seleccionar el estado a ejecutar la sentencia de desplazamiento correspondiente a través de la variable `Estados`.

Dentro de nuestra propia función `void desplazamientos(int estado)` ejecutamos cada estado seleccionado por el pulsador a través de la variable `Estados`:

```
void desplazamientos(int estado){
if ((millis()-Tiempo_de_cambio) > ledDelay){ // Tiempo de
                          // desplazamiento por posición cada 65 ms.

  if (estado == 1){              // Primer estado de desplazamiento.
    for (int x = 0; x < 9; x++)
    digitalWrite(ledsPin[x], LOW); // Apagamos todos los LED.
    Posicion ++;                   // incremento de posición en 1.
    digitalWrite(ledsPin[Posicion-1], HIGH);
    if (Posicion >= 9) Posicion = 0;
  }
```

```
    else if (estado == 2){  // Segundo estado de desplazamiento.
      for (int x = 0; x < 9; x++)
      digitalWrite(ledsPin[x], LOW); // Apagamos todos los LED.
      Posicion --;  // decremento de posición en 1
      digitalWrite(ledsPin[Posicion], HIGH);
      if (Posicion <= 0) Posicion = 9;
    }

  ... etc.

    Tiempo_de_cambio = millis();      // Tiempo final de secuencia.
  }
```

Por primera vez somos capaces de usar la condición lógica `if else` para la ejecución de sentencias diferentes lógicas, y crear funciones propias para el diseño de código más ordenado y estructurado.

CÓDIGO DE PROGRAMACIÓN

```
/*****************************************************************
*       Práctica 2.1 - Órdenes de desplazamiento de LED          *
*****************************************************************
Este programa permite realizar la ejecución de diferentes estilos
de desplazamiento de LED por estados.
*****************************************************************/

int ledsPin[] = {2, 3, 4, 5, 6, 7, 8, 9, 10};  // Declaramos un
vector de tamaño 9 para los puertos digitales (2 hasta 10).
int PulsadorPin = 11; // Declaramos un puerto para el pulsador.
int Estado_Pulsador = LOW;  // Declaramos una variable para leer el
estado del pulsador.
int Estados = 0;  // Declaramos una variable para los diferentes
estados.
int Posicion;  // Declaramos una variable para el cambio de posición
de LED para cada dirección.
int ledDelay(65); // Declaramos un retardo para cada cambio de LED.
unsigned long  Tiempo_de_cambio;  // Declaramos una variable de
conteo de tiempo para el desplazamiento de LED.

void setup() {
  // Configuramos puertos digitales como salida (2 hasta 10).
  for (int x = 0; x < 9; x++){
    pinMode(ledsPin[x], OUTPUT);
  }
  pinMode(PulsadorPin, INPUT); // Configuración del puerto 11 como
entrada.
```

```
  Tiempo_de_cambio = millis();
}

void loop() {
  Estado_Pulsador = digitalRead(PulsadorPin); // Leemos el estado
lógico del pulsador.
  if (Estado_Pulsador == HIGH){  // Leemos el estado lógico 1 del
pulsador
    Estados ++;      // Incrementamos al siguiente estado + 1;
    if ((Estados == 1)||(Estados == 3)||(Estados == 5)) Posicion = 0;
// inicializamos la posición en 0 para desplazamiento ascendente.
    else if ((Estados == 2)||(Estados == 4)||(Estados == 6)) Posicion
= 9;  // inicializamos la posición en 9 para desplazamiento
descendente.
    else Estados = 1;  // Reiniciamos el primer estado.
    delay(200);      // Retardo para eliminar rebotes en el pulsador.
  }
  desplazamientos(Estados); // Llama a la función  para ejecutar cada
estado.
}

// Función de estados para los diferentes desplazamientos.
void desplazamientos(int estado){
  if ((millis()-Tiempo_de_cambio) > ledDelay){  // Tiempo de
desplazamiento por posición cada 65 ms.

    if (estado == 1){  // Primer estado de desplazamiento.
      for (int x = 0; x < 9; x++)digitalWrite(ledsPin[x], LOW); //
Apagamos todos los LED.
      Posicion ++;  // Incremento de posición en 1
      digitalWrite(ledsPin[Posicion-1], HIGH);
      if (Posicion >= 9) Posicion = 0;
    }

    else if (estado == 2){  // Segundo estado de desplazamiento.
      for (int x = 0; x < 9; x++)digitalWrite(ledsPin[x], LOW); //
Apagamos todos los LED.
      Posicion --;  // Decremento de posición en 1
      digitalWrite(ledsPin[Posicion], HIGH);
      if (Posicion <= 0) Posicion = 9;
    }

    else if (estado == 3){  // Tercer estado de desplazamiento.
      for (int x = 0; x < 9; x++)digitalWrite(ledsPin[x], HIGH); //
Encendemos todos los LED.
```

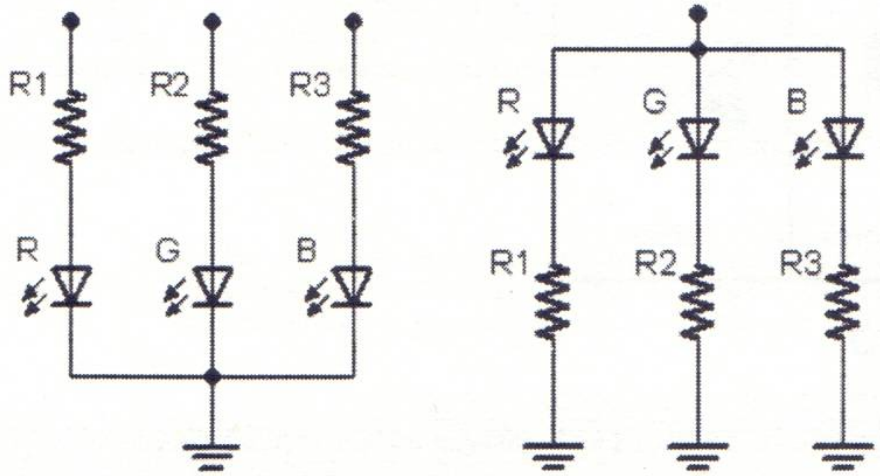
```
      Posicion ++;  // Incremento de posición en 1
      digitalWrite(ledsPin[Posicion-1], LOW);
      if (Posicion >= 9) Posicion = 0;
    }

    else if (estado == 4){  // Cuarto estado de desplazamiento.
      for (int x = 0; x < 9; x++)digitalWrite(ledsPin[x], HIGH); //
Encendemos todos los LED.
      Posicion --;  // Decremento de posición en 1
      digitalWrite(ledsPin[Posicion], LOW);
      if (Posicion <= 0) Posicion = 9;
    }

    else if (estado == 5){  // Quinto estado de desplazamiento.
      Posicion ++;  // Incremento de posición en 1
      digitalWrite(ledsPin[Posicion-1], LOW);
      if (Posicion > 9) {
        for (int x = 0; x < 9; x++)digitalWrite(ledsPin[x], HIGH); //
Apagamos todos los LED.
        Posicion = 0;
      }
    }

    else if (estado == 6){  // Sexto estado de desplazamiento.
      Posicion --;  // Decremento de posición en 1
      digitalWrite(ledsPin[Posicion], LOW);
      if (Posicion < 0) {
        for (int x = 0; x < 9; x++)digitalWrite(ledsPin[x], HIGH); //
Apagamos todos los LEDs.
        Posicion = 9;
      }
    }

    Tiempo_de_cambio = millis();  // Tiempo final de secuencia.
  }
}
```

Introducimos el código, **Verificamos** y **Cargamos**. Al ejecutar el programa no observaremos alguno de los desplazamientos desarrollados en el programa. Cuando presionamos una sola vez el pulsador observaremos el primer desplazamiento, este desplazamiento es parecido al ejemplo "**Desplazamiento de LED**" de la práctica anterior. Si nuevamente presionamos el pulsador, observaremos el desplazamiento descendente de encendido de un LED. Después de presionar hasta observar todos los desplazamientos programados, nuevamente volveremos al estado inicial, es decir, volveremos a observar el primer desplazamiento.

2.2 LED RGB interactivo con pulsadores

En este ejemplo crearemos efectos luminosos en un LED RGB con la combinación de colores rojo, verde y azul a través de tres pulsadores, donde cada pulsador corresponde a un color, respectivamente.

El diodo LED RGB es un dispositivo compuesto internamente por tres diodos LED de diferente color: Rojo (Red), Verde (Green) y Azul (Blue). Al variar la intensidad de corriente de cada LED se producen diferentes colores.



El LED RGB ánodo común, todos los cátodos o negativos van conectados a un solo punto de referencia o GND, y cada ánodo o positivo va conectado a una resistencia de 330 Ω.

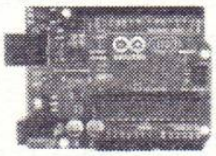
En esta práctica utilizaremos el LED RGB de cátodo común, cada ánodo o positivo del LED RGB es conectado a una resistencia de 330 Ω y los cátodos o negativos referenciados a un solo punto.

Cada pin del LED RGB va conectado a los puertos 9, 10 y 11 de la tarjeta Arduino (rojo = 11, verde = 10 y azul = 9). Se usan estos puertos para generar señales de salida en PWM para poder variar las intensidades de los colores.

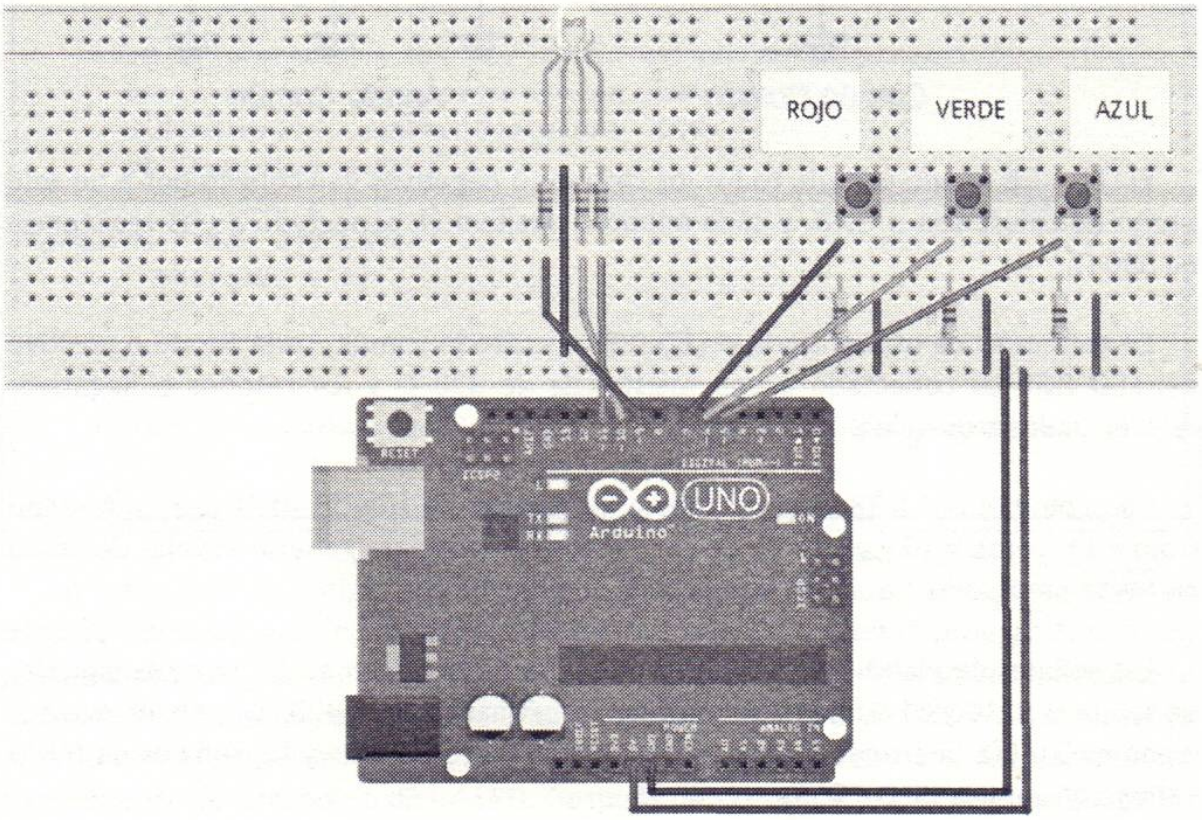
Las señales de PWM (Pulse Width Modulation) funcionan en los puertos digitales de salida 3, 9, 10 y 11 a una frecuencia de aproximadamente 490 Hz, y en los puertos digitales 5 y 6 a una frecuencia de aproximadamente 980 Hz, y las señales de PWM varían con valores de 0 a 255.

En el Arduino MEGA los puertos PWM son del puerto 2 al 13 y 44 al 46.

MATERIALES

 Arduino UNO	 Protoboard	 1 LED RGB	 3 Resistencias de 330 Ω	 3 Resistencias de 4,7 KΩ
 3 Pulsadores	 Cables			

CIRCUITO



DESCRIPCIÓN GENERAL DE LA PROGRAMACIÓN

En la programación declaramos tres variables para los puertos digitales de salida del LED RGB:

```
int LedRojo = 9;          // Declara Pin LED Rojo (9).
int LedVerde = 10;        // Declara Pin LED Verde (10).
int LedAzul = 11;         // Declara Pin LED Azul (11).
```

Declaramos tres variables para los pulsadores en los puertos de entrada 5, 6 y 7:

```
int RojoPulsador = 7; //Pulsador correspondiente al color rojo (7).
int VerdePulsador = 6;//Pulsador correspondiente al color verde (6).
int AzulPulsador = 5; //Pulsador correspondiente al color azul (5).
```

Declaramos tres variables para variar las intensidades de los colores del LED RGB de 0 a 255:

```
int Rojo = 0; // Variable para definir la intensidad de color rojo.
int Verde= 0; // Variable para definir la intensidad de color verde.
int Azul = 0; // Variable para definir la intensidad de color azul.
```

Dentro de la función principal `void setup()` configuramos los puertos digitales para el LED RGB como salida `OUTPUT` y los puertos digitales para los pulsadores como entradas `INPUT` a través de la función `pinMode`:

```
// Configuración de puertos digitales como salida (9 hasta 11).
pinMode(LedRojo, OUTPUT);
pinMode(LedVerde, OUTPUT);
pinMode(LedAzul, OUTPUT);
// Configuración de puertos digitales como entrada (5 hasta 7).
pinMode(RojoPulsador, INPUT);
pinMode(VerdePulsador, INPUT);
pinMode(AzulPulsador, INPUT);
```

Dentro la función principal `void loop()` realizamos la lectura de los estados lógicos de los pulsadores. Por ejemplo, si el estado lógico del pulsador para el color rojo es 1 (5 V) `if (digitalRead(RojoPulsador) == HIGH)`, la variable `Rojo ++` incrementa en 1 para variar la intensidad del color rojo hasta 255, pero cuando esta variable llega hasta 255 se reinicia a 0 en la condición lógica `if (Rojo > 255) Rojo = 0`. Para presentar esos datos, generamos señales PWM con la función `analogWrite(LedRojo, Rojo)` correspondiente al puesto de salida y señal a generar `(puerto, valor)`:

```
if (digitalRead(RojoPulsador) == HIGH) { // Si al pulsador
                    // correspondiente al color rojo está inactivo.
  Rojo ++;                      // Incrementa el valor de rojo en 1.
```

```
    if (Rojo > 255) Rojo = 0;          // Si rojo es mayor a 255, el color
                                       // rojo reinicia a 0.
  }
  analogWrite(LedRojo, Rojo);          // Presenta el valor del color
                                       // rojo.

  ... etc.
```

Realizamos la misma lógica de programación para los colores verde y azul.

Por primera vez somos capaces de generar una infinidad de colores mediante la función `analogWrite`. Esta función genera señales PWM en los puertos digitales de salida de la tarjeta Arduino UNO 3, 5, 6, 9, 10 y 11, y Arduino MEGA en los puertos 2 al 13 y 44 al 46.

CÓDIGO DE PROGRAMACIÓN

```
/*****************************************************************
*       Práctica  2.2 - LED RGB interactivo con pulsadores       *
*****************************************************************
Este programa permite realizar el efecto de luces interactivas
multicolor con pulsadores.
*****************************************************************/

int LedRojo = 9;           // Declara Pin LED Rojo (D9).
int LedVerde = 10;         // Declara Pin LED Verde (D10).
int LedAzul = 11;          // Declara Pin LED Azul (D11).

int RojoPulsador = 7;      // Declaro pulsador correspondiente al color
rojo (D7).
int VerdePulsador = 6;     // Declaro pulsador correspondiente al color
verde (D6).
int AzulPulsador = 5;      // Declaro pulsador correspondiente al color
azul (D5).

int Rojo = 0;              // Variable para definir la intensidad de
color rojo.
int Verde= 0;              // Variable para definir la intensidad de
color verde.
int Azul = 0;              // Variable para definir la intensidad de
color azul.
void setup() {
  // Configuración de puertos digitales como salida (9 hasta 11).
  pinMode(LedRojo, OUTPUT);
  pinMode(LedVerde, OUTPUT);
```

```
  pinMode(LedAzul, OUTPUT);
  // Configuración de puertos digitales como entrada (desde 5 hasta
7).
  pinMode(RojoPulsador, INPUT);
  pinMode(VerdePulsador, INPUT);
  pinMode(AzulPulsador, INPUT);
}

void loop(){

  if (digitalRead(RojoPulsador) == HIGH) { // Si el pulsador
correspondiente al color rojo está inactivo
    Rojo ++;                                // Incrementa el valor de
rojo en 1.
    if (Rojo > 255) Rojo = 0;               // Si rojo es mayor a 255,
el color rojo se reinicia a 0.
  }

  if (digitalRead(VerdePulsador) == HIGH) { // Si el pulsador
correspondiente al color verde está inactivo
    Verde ++;                               // Incrementa el valor de
verde en 1.
    if (Verde > 255) Verde = 0;             // Si verde es mayor a
255, el color verde se reinicia a 0.
  }

if (digitalRead(AzulPulsador) == HIGH) { // Si el pulsador
correspondiente al color azul está inactivo
    Azul ++;                                // Incrementa el valor de
azul en 1.
    if (Azul > 255) Azul = 0;               // Si azul es mayor a 255,
el color azul se reinicia a 0.
  }

  analogWrite(LedRojo, Rojo);               // Presenta el valor del
color rojo.
  analogWrite(LedVerde, Verde);             // Presenta el valor del
color verde.
  analogWrite(LedAzul, Azul);               // Presenta el valor del
color azul.
  delay(50);                                // Esperamos 0.1 segundos.
}
```

Ahora hacemos clic en **Verificar** para asegurarnos de no cometer errores en el código, y por último **Cargar** para subir el código a nuestra placa Arduino y podemos ver diferentes intensidades de colores en el LED RGB conforme presionamos los tres pulsadores, cada uno de ellos controla la intensidad de los colores (rojo, verde y azul). Cada vez que presionamos los pulsadores, estos incrementarán de 0 a 255 para obtener diferentes intensidades y combinaciones.

TERCERA PRÁCTICA (PUERTOS ANALÓGICOS)

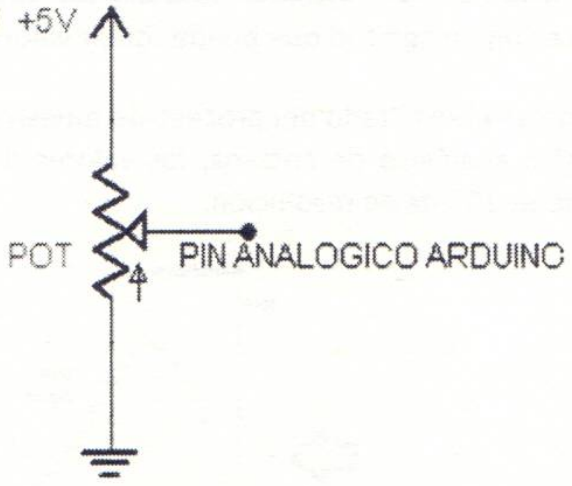
En esta práctica aprenderemos a adquirir señales analógicas de diferentes dispositivos como un potenciómetro o una LDR. Para leer señales analógicas externas se usan los puertos analógicos de ia tarjeta Arduino con una resolución de 10 bits, es decir, 1024 valores posibles para una señal analógica de 0 a 5 V.

Un sensor es un dispositivo capaz de detectar señales físicas y transformarlas en señales eléctricas (voltaje o corriente) como por ejemplo: temperatura, intensidad de luz, etc.

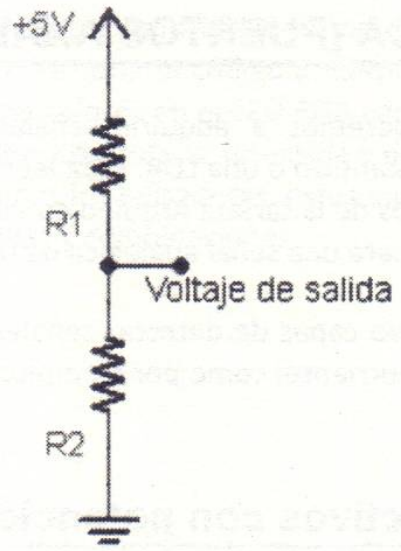
3.1 LED RGB interactivos con potenciómetros

Ahora vamos a realizar el mismo ejemplo anterior, que es generar efecto de multicolor a diferentes intensidades con dos LED RGB, con la diferencia de que utilizaremos tres potenciómetros para regular la intensidad de colores (rojo, verde y azul).

El circuito adicional para este ejemplo es el uso de un potenciómetro. Este se conecta a una fuente de voltaje y al variar su resistencia varía el nivel de voltaje en el pin central del dispositivo.



El circuito equivalente de un potenciómetro corresponde a dos resistencias conectadas en serie R1 y R2.



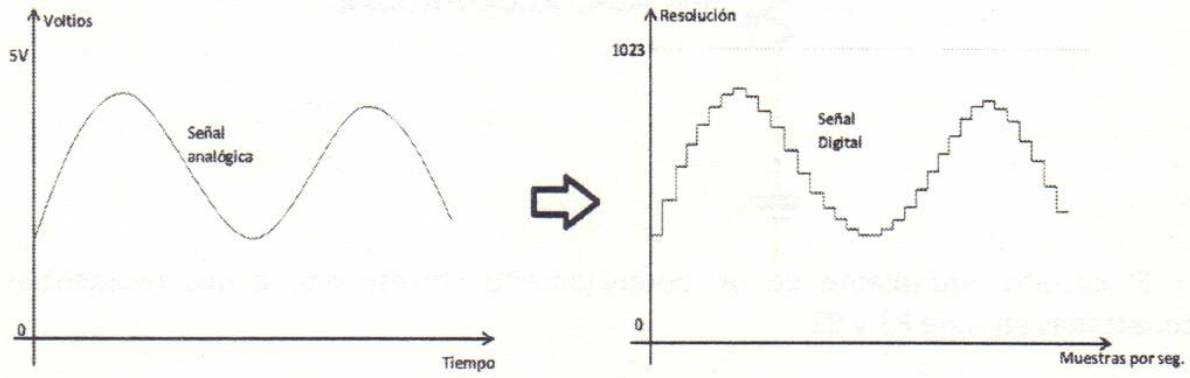
El nivel de voltaje de salida del potenciómetro se calcula con la fórmula de divisor de voltaje.

$$V_{salida} = \frac{R_1}{R_1 + R_2} V_{entrada}$$

Donde V_{salida} es el valor de la caída de tensión en la resistencia $R2$ y $V_{entrada}$ es el voltaje de alimentación del circuito, para Arduino es 5 V.

Una señal analógica es una magnitud que puede tomar valores entre 0 y 5 voltios.

Una señal digitalizada es el resultado del proceso de muestreo de cuantificación y codificación de una señal analógica de entrada, los valores digitales son N bits de resolución, para Arduino es 10 bits de resolución.

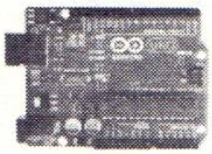


Arduino convierte una señal analógica a digital a través del ADC (Analog Digital Converter) que es un componente funcional interno. Un ADC es un dispositivo que convierte una señal analógica a digital con un número de bits.

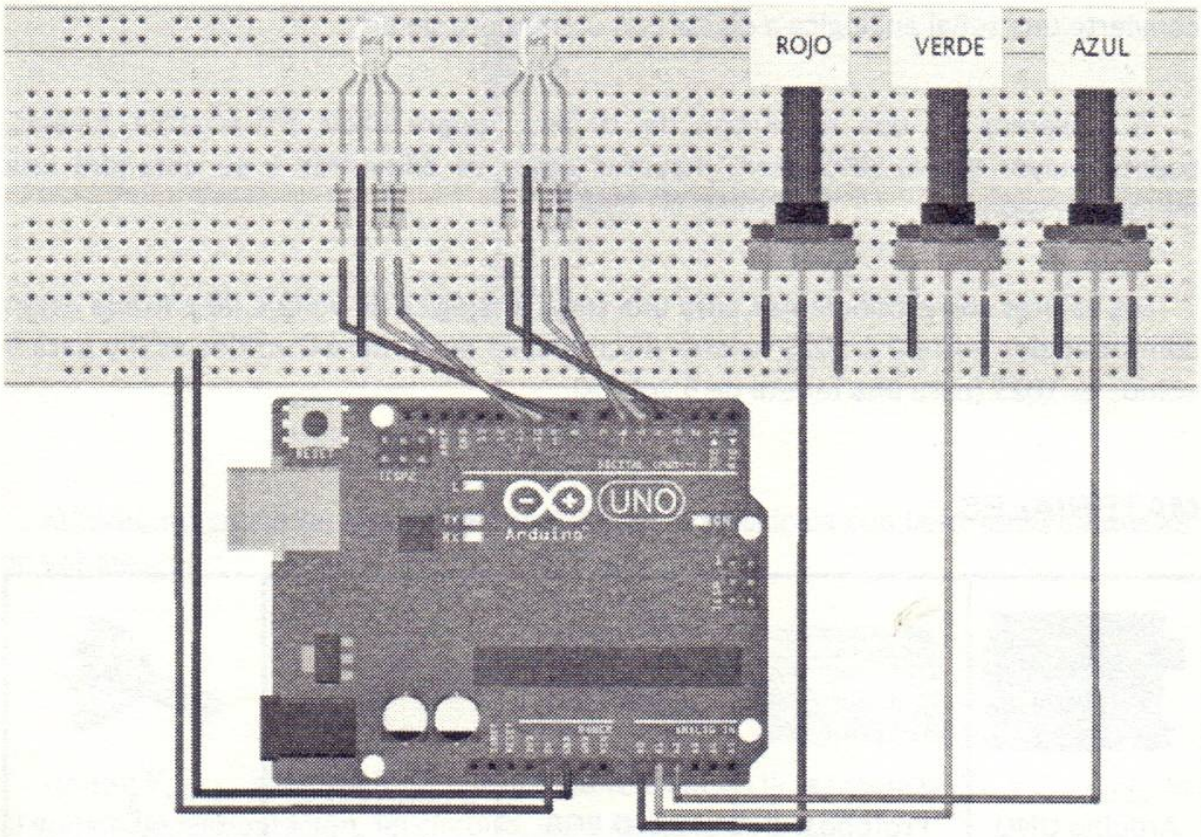
Si hablamos de una resolución de 10 bits, quiere decir $2^{10} = 1024$ niveles, cuanto mayor sea la resolución, mayores datos se adquirirán y se obtendrá una mejor precisión.

Los valores correspondientes para una señal analógica de voltaje de entrada están comprendidos entre 0 a 1023 valores discretos, es decir, para 0 voltios es 0 y para 5 voltios es 1023 (para una tarjeta de 5 voltios).

MATERIALES

 Arduino UNO	 Protoboard	 2 LED RGB	 6 Resistencias de 330 Ω	 3 potenciómetros
 Cables				

CIRCUITO



DESCRIPCIÓN GENERAL DE LA PROGRAMACIÓN

Para el programa declaramos variables `int` para los puertos de salida de los LED RGB:

```
//Declaración de puerto para el LED RGB 1.
int ledRojo_1 = 9;
int ledVerde_1 = 10;
int ledAzul_1 = 11;

//Declaración de puerto para el LED RGB 2.
int ledRojo_2 = 3;
int ledVerde_2 = 5;
int ledAzul_2 = 6;
```

Declaramos variables `int` para los puertos analógicos de entrada correspondientes a los tres potenciómetros:

```
int pinPotenciometro_1 = A0;    // Potenciómetro para color Rojo.
int pinPotenciometro_2 = A1;    // Potenciómetro para color Verde.
int pinPotenciometro_3 = A2;    // Potenciómetro para color Azul.
```

Declaramos variables para el almacenamiento de datos para cada una de las señales analógicas, y declaramos variables `int` para convertir los datos de las señales analógicas a intensidades correspondientes a cada color:

```
int pinPotenciometro_1 = A0; //Puerto analógico Potenciómetro Rojo.
int pinPotenciometro_2 = A1; //Puerto analógico Potenciómetro Verde.
int pinPotenciometro_3 = A2; //Puerto analógico Potenciómetro Azul.

/* Variables para el almacenamiento de datos analógicos
correspondientes a los potenciómetros. */
int valorPotenciomerto_1 = 0;
int valorPotenciomerto_2 = 0
int valorPotenciomerto_3 = 0
```

Dentro de la función principal `void setup()` configuramos los puertos de salida para los LED RGB:

```
pinMode(ledRojo_1,OUTPUT);    //El LED Rojo 1 como una salida
pinMode(ledVerde_1,OUTPUT);   //El LED Verde 1 como una salida
pinMode(ledAzul_1,OUTPUT);    //El LED Azul 1 como una salida

pinMode(ledRojo_2,OUTPUT);    //El LED Rojo 2 como una salida
pinMode(ledVerde_2,OUTPUT);   //El LED Verde 2 como una salida
pinMode(ledAzul_2,OUTPUT);    //El LED Azul 2 como una salida
```

Dentro de la función principal `void loop()` realizamos la lectura de las señales analógicas de los potenciómetros:

```
valorPotenciomerto_1 = analogRead(pinPotenciometro_1);  /* Leemos el
        valor del potenciómetro 1 correspondiente al color rojo. */
valorPotenciomerto_2 = analogRead(pinPotenciometro_2);  /* Leemos el
       valor del potenciómetro 1 correspondiente al color verde. */
valorPotenciomerto_3 = analogRead(pinPotenciometro_3);  /* Leemos el
        valor del potenciómetro 1 correspondiente al color azul. */
```

A través de la función `map` convertimos los datos analógicos de entrada a datos para señales PWM en los puertos digitales de salida, es decir, datos de 0 - 1023 los convertimos a 0 - 255:

```
// Mapeamos las datos analógicos de 0 - 1023 a 0 - 255.
intensidadRojo  = map(valorPotenciomerto_1, 0, 1023, 0, 255);
intensidadVerde = map(valorPotenciomerto_2, 0, 1023, 0, 255);
intensidadAzul  = map(valorPotenciomerto_3, 0, 1023, 0, 255);
```

Finalmente enviaremos estos datos a los puertos digitales de salida correspondientes a los pines de los LED RGB. Por ejemplo para el color rojo de un LED RGB es: `analogWrite(ledRojo_1, intensidadRojo)`, y para el otro LED RGB los

datos son inversos, por ejemplo para el color rojo es: `analogWrite(ledRojo_2, 255-intensidadRojo)`:

```
analogWrite(ledRojo_1, intensidadRojo);      /* LED ROJO 1. El valor
                    de 0 a 255 es obtenido del potenciómetro rojo. */
analogWrite(ledVerde_1, intensidadVerde);    /* LED VERDE 1. El valor
                   de 0 a 255 es obtenido del potenciómetro verde. */
analogWrite(ledAzul_1, intensidadAzul);      /* LED AZUL 1. El valor
                    de 0 a 255 es obtenido del potenciómetro azul. */

analogWrite(ledRojo_2, 255-intensidadRojo); /* LED ROJO 2. Le resta
      al total de (255) el valor obtenido del potenciómetro rojo. */
analogWrite(ledVerde_2, 255-intensidadVerde);/*LED VERDE 2. Le resta
     al total de (255) el valor obtenido del potenciómetro verde. */
analogWrite(ledAzul_2, 255-intensidadAzul); /* LED AZUL 2. Le resta
      al total de (255) el valor obtenido del potenciómetro azul. */
```

Por primera vez usamos la función `analogRead` para la lectura de datos analógicos, y la función `map` para convertir números de un intervalo a otro.

CÓDIGO DE PROGRAMACIÓN

```
/*****************************************************************
*    Práctica 3.1 - LED RGB interactivos con potenciómetros     *
*****************************************************************
Este programa permite realizar el efecto de luces interactivas
multicolor con potenciómetros.
*****************************************************************/

int ledRojo_1 = 9;              //Declara Pin LED Rojo 1 (D9).
int ledVerde_1 = 10;            //Declara Pin LED Verde 1 (D10).
int ledAzul_1 = 11;             //Declara Pin LED Azul 1 (D11).

int ledRojo_2 = 3;              //Declara Pin LED Rojo 2 (D3).
int ledVerde_2 = 5;             //Declara Pin LED Verde 2 (D5).
int ledAzul_2 = 6;              //Declara Pin LED Azul 2 (D6).

int pinPotenciometro_1 = A0;    //Declara Pin analógico para el
Potenciómetro correspondiente al Rojo.
int pinPotenciometro_2 = A1;    //Declara Pin analógico para el
Potenciómetro correspondiente al Verde.
int pinPotenciometro_3 = A2;    //Declara Pin analógico para el
Potenciómetro correspondiente al Azul.

int valorPotenciomerto_1 = 0;  //Variable para el almacenamiento de
valores analógicos de la salida del potenciómetro correspondiente al
```

```
color Rojo.
int valorPotenciomerto_2 = 0;  //   ||        ||        ||
||        ||             ||             ||             ||
||
int valorPotenciomerto_3 = 0;  //   ||        ||        ||
||        ||             ||             ||             ||
||

int intensidadRojo;           // Variable para  almacenar las
intensidades del color rojo.
int intensidadVerde;          // Variable para  almacenar las
intensidades del color verde.
int intensidadAzul;           // Variable para  almacenar las
intensidades del color azul.

void setup() {
  pinMode(ledRojo_1,OUTPUT);    //El LED Rojo 1 como una salida
  pinMode(ledVerde_1,OUTPUT);   //El LED Verde 1 como una salida
  pinMode(ledAzul_1,OUTPUT);    //El LED Azul 1 como una salida

  pinMode(ledRojo_2,OUTPUT);    //El LED Rojo 2 como una salida
  pinMode(ledVerde_2,OUTPUT);   //El LED Verde 2 como una salida
  pinMode(ledAzul_2,OUTPUT);    //El LED Azul 2 como una salida
}

void loop() {

   valorPotenciomerto_1 = analogRead(pinPotenciometro_1);  // Leemos
el valor del potenciómetro 1 correspondiente al color rojo.
   valorPotenciomerto_2 = analogRead(pinPotenciometro_2);  // Leemos
el valor del potenciómetro 1 correspondiente al color verde.
   valorPotenciomerto_3 = analogRead(pinPotenciometro_3);  // Leemos
el valor del potenciómetro 1 correspondiente al color azul.

   intensidadRojo  = map(valorPotenciomerto_1, 0, 1023, 0, 255);  //
mapeamos las datos analógicos de 0 - 1023 a 0 - 255.
   intensidadVerde = map(valorPotenciomerto_2, 0, 1023, 0, 255);
   intensidadAzul  = map(valorPotenciomerto_3, 0, 1023, 0, 255);

   analogWrite(ledRojo_1, intensidadRojo);       //LED ROJO 1. El
valor de 0 a 255 es obtenido del potenciómetro rojo.
   analogWrite(ledVerde_1, intensidadVerde);     //LED VERDE 1. El
valor de 0 a 255 es obtenido del potenciómetro verde.
   analogWrite(ledAzul_1, intensidadAzul);       //LED AZUL 1. El
valor de 0 a 255 es obtenido del potenciómetro azul.
```

```
   analogWrite(ledRojo_2, 255-intensidadRojo);    //LED ROJO 2. Le
resta al total de (255) el valor obtenido del potenciómetro rojo.
   analogWrite(ledVerde_2, 255-intensidadVerde); //LED VERDE 2. Le
resta al total de (255) el valor obtenido del potenciómetro verde.
   analogWrite(ledAzul_2, 255-intensidadAzul);    //LED AZUL 2. Le
resta al total de (255) el valor obtenido del potenciómetro azul.
}
```

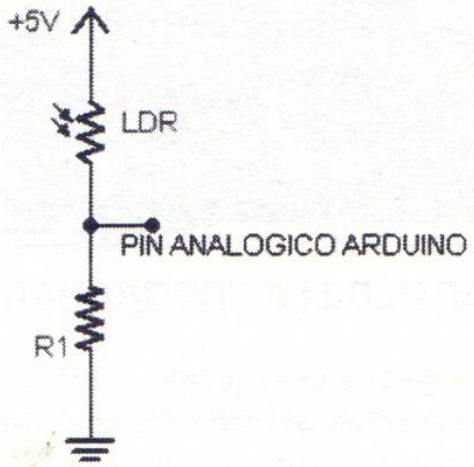
En esta ocasión cuando se **Verifique** y **Cargue** el código, podremos observar la iluminación de los LED RGB conforme varía la resistencia de los tres potenciómetros, donde cada potenciómetro controla la intensidad de cada color (rojo, verde y azul). Cuando variamos la resistencia de los potenciómetros, estos generan señales de 0 V a 5 V y Arduino lee estas señales como valores discretos de 0 a 1023, cada señal es multiplicada por una mapeada para convertir las señales de 0 a 255 y así poder obtener las distintas intensidades.

3.2 LED RGB interactivo con sensor de luz (LDR)

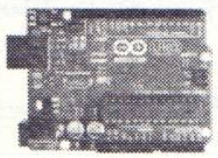
En este ejemplo usaremos una LDR para variar la intensidad de luz y colores de un LED RGB.

Una LDR es un dispositivo cuyo valor de resistencia varía a la intensidad de luz incidente en él.

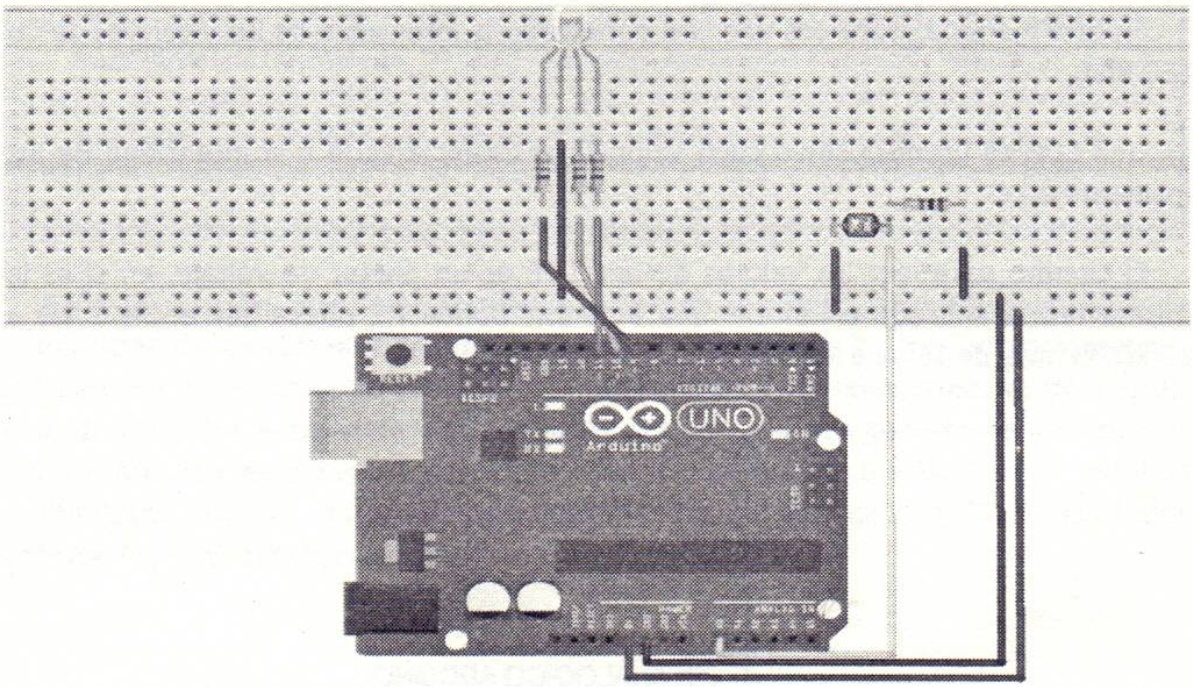
El circuito para leer los valores de una LDR es un divisor de voltaje, en sí es la misma teoría aplicada en el ejemplo anterior, donde R1 puede ser entre 1 kΩ a 10 kΩ. El nivel de señal a leer es tomado entre la LDR y R1.



MATERIALES

 Arduino UNO	 Protoboard	 1 LED RGB	 3 Resistencias de 330 Ω	 1 Resistencia de 1 kΩ
 1 Fotocelda	 Cables			

CIRCUITO



DESCRIPCIÓN GENERAL DE LA PROGRAMACIÓN

Para la programación se declara una variable entera para el puerto analógico del sensor LDR, y una variable para el almacenamiento de datos analógicos:

```
int PinLDR = A0;     // Puerto analógico A0 para la fotocelda.
int LDRValor = 0;    // Variable para la lectura de valores
                        analógicos.
```

Al igual que en ejemplos anteriores declaramos variables enteras para los puertos de salida correspondientes al LED RGB:

```
int LedRojo = 9;     // Puerto analógico 9 para el color rojo.
int LedVerde=10;     // Puerto  analógico  10 para el color Verde.
int LedAzul=11;      // Puerto  analógico  11 para el color Azul.
```

Dentro de la función principal `void setup()` configuramos los puertos de salida para el LED RGB:

```
// Configuramos los puertos analógicos 9, 10 y 11 como salida.
pinMode(LedRojo, OUTPUT);
pinMode(LedVerde, OUTPUT);
pinMode(LedAzul, OUTPUT);
```

Dentro de la función principal `void loop()` leemos los datos del sensor LDR, y les delimitamos sus valores analógicos para diferentes intensidades y efectos de iluminación en el LED RGB:

```
// Delimitamos los valores analógicos de la LDR.
if(LDRValor >= 1023){
  analogWrite(LedRojo, 128);
  analogWrite(LedVerde, 0);
  analogWrite(LedAzul, 0);
}
else if((LDRValor>= 959) && (LDRValor < 1023)){
  analogWrite(LedRojo, 255);
  analogWrite(LedVerde, 0);
  analogWrite(LedAzul, 0);
}

...etc
```

CÓDIGO DE PROGRAMACIÓN

```
/*****************************************************************
*       Práctica 3.2 - LED RGB interactivo con fotocelda.        *
*****************************************************************
Este programa permite realizar el encendido de luces multicolor
interactivamente con un sensor de iluminación.
*****************************************************************/

int PinLDR = A0;     // Asignamos el puerto analógico A0 para el
sensor de iluminación o fotocelda.
int LDRValor = 0;    // Asignamos el valor leído por el sensor de
iluminación.
int LedRojo = 9;     // Asignamos el puerto analógico 9 para el color
rojo.
int LedVerde=10;     // Asignamos el puerto  analógico  10 para el
color Verde.
int LedAzul=11;      // Asignamos el puerto  analógico  11 para el
color Azul.

void setup(){
  // Configuramos los puertos analógicos 9, 10 y 11 como salida.
  pinMode(LedRojo, OUTPUT);
  pinMode(LedVerde, OUTPUT);
  pinMode(LedAzul, OUTPUT);
}
void loop(){
  LDRValor = analogRead(PinLDR); // Leo el valor analógico del sensor
de iluminación.

  // Delimitamos los valores analógicos de la LDR.
  if(LDRValor >= 1023){
    analogWrite(LedRojo, 128);
```

```
    analogWrite(LedVerde, 0);
    analogWrite(LedAzul, 0);
  }
  else if((LDRValor>= 959) && (LDRValor < 1023)){
    analogWrite(LedRojo, 255);
    analogWrite(LedVerde, 0);
    analogWrite(LedAzul, 0);
  }
  else if((LDRValor >= 895) && (LDRValor < 959)){
    analogWrite(LedRojo, 255);
    analogWrite(LedVerde, 128);
    analogWrite(LedAzul, 0);
  }
  else if((LDRValor >= 831) && (LDRValor < 895)){
    analogWrite(LedRojo, 255);
    analogWrite(LedVerde, 255);
    analogWrite(LedAzul, 0);
  }
  else if((LDRValor >= 767) && (LDRValor< 831)){
    analogWrite(LedRojo, 255);
    analogWrite(LedVerde, 255);
    analogWrite(LedAzul, 128);
  }
  else if((LDRValor >= 703) && (LDRValor < 767)){
    analogWrite(LedRojo, 128);
    analogWrite(LedVerde, 255);
    analogWrite(LedAzul, 255);
  }
  else if((LDRValor >= 639) && (LDRValor < 703)){
    analogWrite(LedRojo, 128);
    analogWrite(LedVerde, 128);
    analogWrite(LedAzul, 255);
  }
  else if((LDRValor >= 575) && (LDRValor < 639)){
    analogWrite(LedRojo, 0);
    analogWrite(LedVerde, 128);
    analogWrite(LedAzul, 255);
  }else if((LDRValor >= 511) && (LDRValor < 575)){
    analogWrite(LedRojo, 0);
    analogWrite(LedVerde, 0);
    analogWrite(LedAzul, 255);
  }
  else if((LDRValor >= 447) && (LDRValor < 511)){
    analogWrite(LedRojo, 0);
    analogWrite(LedVerde, 0);
```

```
    analogWrite(LedAzul, 128);
  }
  else if((LDRValor >= 383) && (LDRValor < 447)){
    analogWrite(LedRojo, 0);
    analogWrite(LedVerde, 128);
    analogWrite(LedAzul, 0);
  }
  else if((LDRValor >= 319) && (LDRValor < 383)){
    analogWrite(LedRojo, 0);
    analogWrite(LedVerde, 255);
    analogWrite(LedAzul, 0);
  }
  else if((LDRValor >= 255) & (LDRValor < 319)){
    analogWrite(LedRojo, 128);
    analogWrite(LedVerde, 255);
    analogWrite(LedAzul, 0);
  }
  else if((LDRValor >= 191) & (LDRValor < 255)){
    analogWrite(LedRojo, 0);
    analogWrite(LedVerde, 255);
    analogWrite(LedAzul, 128);
  }
  else if((LDRValor >= 127) && (LDRValor < 191)){
    analogWrite(LedRojo, 128);
    analogWrite(LedVerde, 255);
    analogWrite(LedAzul, 128);
  }
  else if((LDRValor >= 63) && (LDRValor < 127)){
    analogWrite(LedRojo, 128);
    analogWrite(LedVerde, 128);
    analogWrite(LedAzul, 128);
  }
  else{
    analogWrite(LedRojo, 0);
    analogWrite(LedVerde, 0);
    analogWrite(LedAzul, 0);
  }
}
```

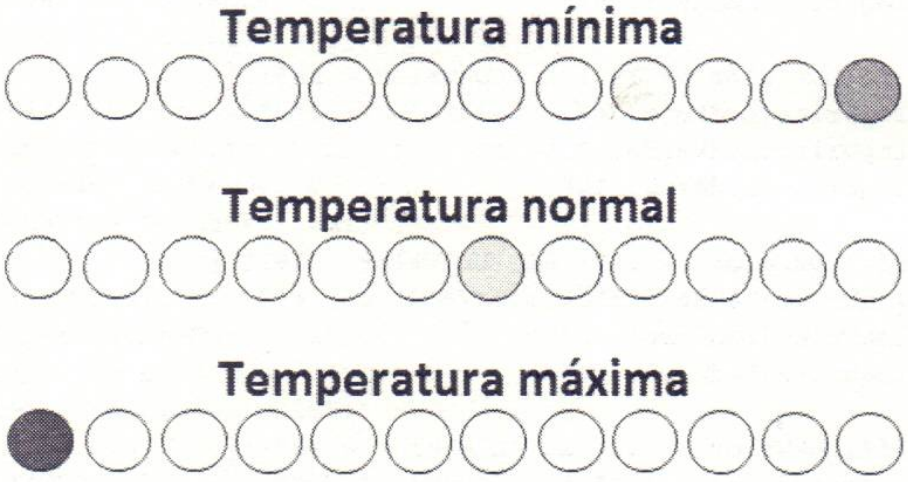
Una vez **Verificado** y **Cargado** el programa en la placa Arduino podremos observar el cambio de colores a medida que variamos la fotorresistencia aproximando a una fuente luminosa. La tonalidad de los colores del LED RGB varía dependiendo de la intensidad luminosa incidida al sensor.

3.3 Termómetro LED

Para culminar con esta práctica de señales analógicas, vamos a realizar un termómetro LED con un sensor de temperatura LM35.

En este realizaremos un vector de 12 LED como indicador, donde encenderemos cada uno dependiendo de la temperatura leída por el sensor, es decir, cada LED indicará el nivel de temperatura de un ambiente.

El rango a medir es de 10 a 40 grados centígrados (°C), el primer LED indica el nivel más bajo de temperatura (10 °C), el sexto LED indica el nivel medio de temperatura o temperatura ambiente (25 °C), y el último LED indica el nivel más alto de temperatura (40 °C).

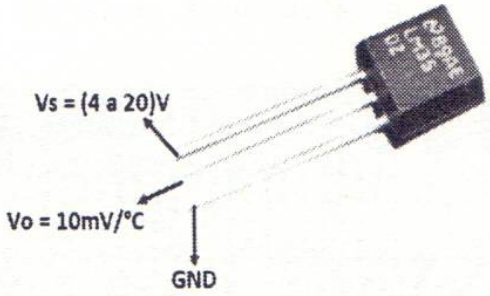


El sensor LM35 es un dispositivo que mide la temperatura de un ambiente cuya salida es una señal de voltaje proporcional a la escala Celsius. Tiene un alcance de -55 °C a +150 °C con una exactitud aproximada de 0,25 °C.

La temperatura de este dispositivo se calcula con la siguiente fórmula:

$$T = \frac{V}{10mV}°C$$

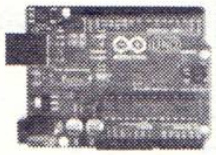
Para cada grado de temperatura hay una diferencia de 10 mV. Donde V es el voltaje de salida del sensor.



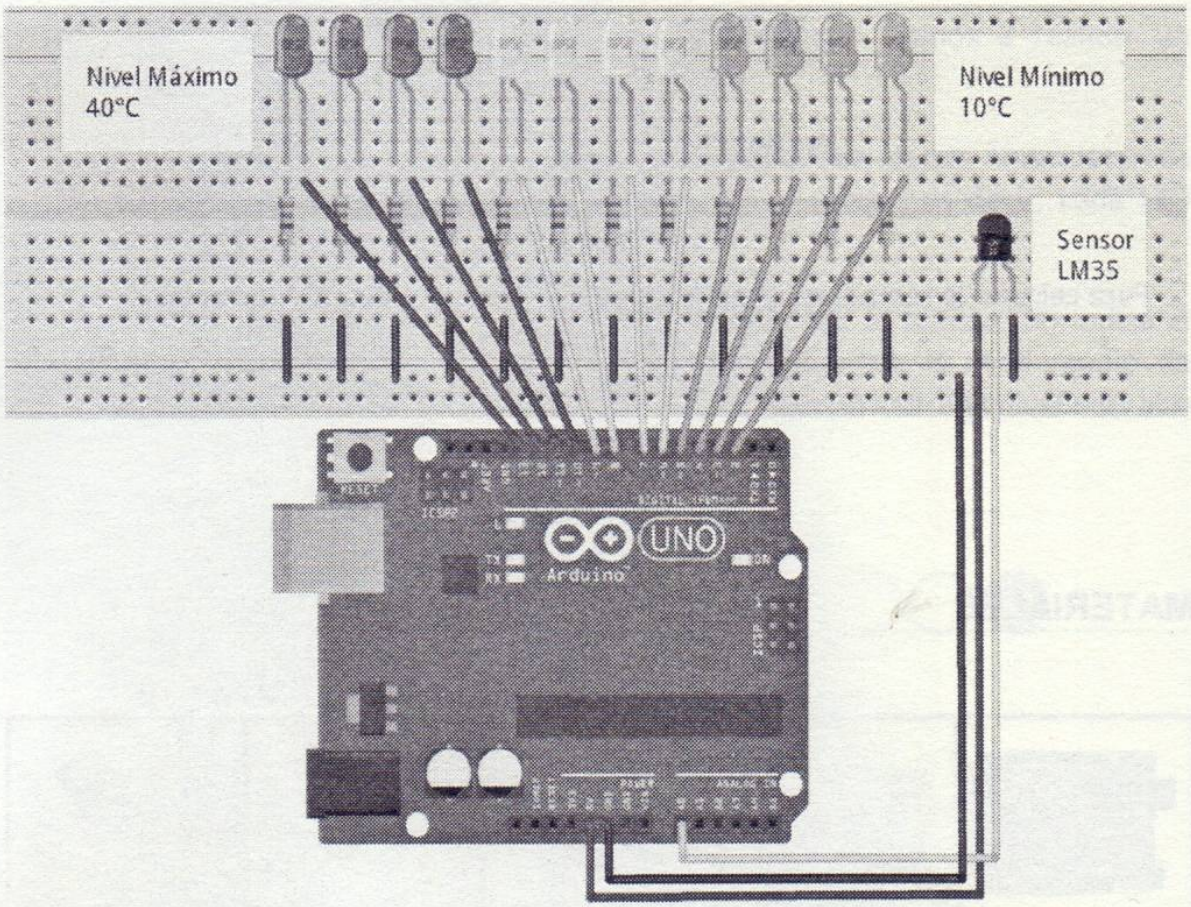
Para calcular la temperatura en Arduino usaremos la siguiente fórmula:

$$T = \frac{5 * ADC}{1023 * 0.01}$$

MATERIALES

 Arduino UNO	 Protoboard	 12 LED (rojo, amarillo y verde)	 12 Resistencias de 330 Ω	 1 Sensor LM35
 Cables				

CIRCUITO



DESCRIPCIÓN GENERAL DE LA PROGRAMACIÓN

Para la programación se declara una variable `int` para el puerto analógico del sensor LM35, y una variable para el almacenamiento de datos analógicos:

```
int Sensor_LM35 = A0;    // Puerto analógico A0 para el sensor LM35.
int Valor_Analogico = 0;// Variable para guardar los datos del sensor.
```

Declaramos una variable `float` para el cálculo de temperatura. Para el cálculo necesitamos aplicar la fórmula escrita anteriormente, por lo que manipularemos números decimales:

```
float Temperatura = 0.0;
```

Declaramos una variable array tipo `int` para 12 LED del termómetro, y una variable para recorrer sus niveles:

```
int Termometro[] = {2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13};
int niveles;
```

Dentro de la función principal `void setup()` configuramos los puertos de salida para el LED del termómetro a través del bucle `for`:

```
for (int x = 0; x < 12; x ++){
  pinMode(Termometro[x],OUTPUT);// Configuración de puertos de
salida.
}
delay(1000); // Retardo
```

Dentro de la función principal `void loop()` leemos los datos del sensor, y calculamos la temperatura como un número decimal según la fórmula para Arduino:

```
Valor_Analogico= analogRead(Sensor_LM35);  // Leer datos del sensor.
Temperatura = (Valor_Analogico * 5) / (1023 * 0.01); /* Calcular
                        temperatura según la fórmula establecida. */
```

Dentro de la estructura lógica `if((Temperatura>=10)&&(Temperatura<=40))` limitamos la temperatura del sensor a un rango permitido, es decir, de 10 a 40 °C, y mapeamos los niveles de temperatura para obtener un rango de 0 a 12, es decir, 0 equivale a 10 °C y 12 equivale a 40 °C. Estos niveles sirven para encender los LED para indicar en qué nivel de temperatura se encuentra el ambiente a medir.

```
if ((Temperatura >= 10)&&(Temperatura <= 40)){ /* Limitar los
                                    valores interesados a medir. */
  niveles = map(Temperatura, 10, 40, 0, 12);  /* Mapear temperatura
                    (10-40°C) a 12 niveles del termómetro (0-12). */
}
```

Encendemos los LED correspondientes al nivel de temperatura:

```
digitalWrite(Termometro[niveles], HIGH);
```

Agregamos un retardo de 200 milisegundos para mantener encendido el LED correspondiente y visible al ojo humano:

```
delay(200);
```

Finalmente apagamos todos los LED a través del bucle `for`, para evitar mantener encendido cualquier LED no deseado al nivel correspondiente.

```
for (int x = 0; x < 12; x ++){
  digitalWrite(Termometro[x],LOW);  // Apagar todos los LED.
}
```

Por primera vez somos capaces de usar variables flotantes (`float`) para la manipulación de datos con punto decimal. En nuestro ejemplo usamos datos del resultado de una fórmula matemática cuyos valores son números decimales.

CÓDIGO DE PROGRAMACIÓN

```
/*****************************************************************
*                Práctica 3.3 - Termómetro LED.                  *
*****************************************************************
Este programa permite leer la temperatura de un sensor de
temperatura LM35 y mostrar los niveles de temperatura en una barra
de 12 LED como termómetro indicador.
*****************************************************************/

int Sensor_LM35 = A0;      // Declaramos el puerto analógico A0 para
el sensor LM35.
int Valor_Analogico = 0;  // Declaramos una variable para guardar los
datos del sensor.
float Temperatura = 0.0;  // Declaramos una variable para el cálculo
de temperatura.

int Termometro[] = {2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13}; //
Variable para 12 puertos digitales para el termómetro.

int niveles;               // Declaramos una variable para los niveles
del termómetro.

void setup(){
  for (int x = 0; x < 12; x ++){
    pinMode(Termometro[x],OUTPUT);  // Configuración de puertos de
salida.
  }
  delay(1000); // Retardo
}

void loop(){
    Valor_Analogico= analogRead(Sensor_LM35);  // Leemos los datos
del sensor.
    Temperatura = (Valor_Analogico * 5) / (1023 * 0.01); //
Calculamos temperatura según la fórmula establecida.

    if ((Temperatura >= 10)&&(Temperatura <= 40)){ // Limitamos los
valores interesados a medir.
```

```
      niveles = map(Temperatura, 10, 40, 0, 12);    // Mapeamos la
temperatura (10-40°C) a 12 niveles del termómetro (0-12).
    }

    digitalWrite(Termometro[niveles], HIGH);  // Activamos 5 V a los
puertos de salida correspondiente a los niveles del termómetro.
    delay(200);  // Retardo de 200 mseg. para mantener encendido el
nivel (LED) correspondiente.

    for (int x = 0; x < 12; x ++){
      digitalWrite(Termometro[x],LOW);  // Apagamos todos los LED.
    }
}
```

Una vez **Verificado** y **Cargado** el programa en la placa Arduino podremos observar el cambio de encendido de los LED por posición a medida que variamos la temperatura del sensor.

Cada LED indica un nivel de temperatura, el LED 12 indica una temperatura de 40 °C y el LED 1 indica una temperatura de 10 °C.

Para variar la temperatura del sensor se aproxima el termómetro a una fuente de calor, como por ejemplo: un secador de cabello o un encendedor.

CUARTA PRÁCTICA (PANTALLAS LCD)

En esta práctica vamos a realizar la presentación de texto y símbolos en una pantalla LCD (Liquid Circuit Display). Por primera vez somos capaces de mostrar cualquier tipo texto o datos producto de la información de un dispositivo.

Al final de esta práctica se aprenderá a generar textos y símbolos, y desplazarlos como en las pantallas electrónicas LED para despliegue de textos. Además en la programación aprenderemos a cómo incorporar librerías auxiliares para el uso de estas herramientas.

4.1 Crear caracteres personalizados

Para este primer ejemplo con pantallas LCD crearemos una carita feliz y una batería descargada para presentar en una pantalla LCD de 16x2 (16 caracteres de 2 filas).

La pantalla LCD ofrece un gran conjunto de caracteres preconfigurados para mostrar texto, pero por lo general hace falta incorporar textos especiales como: ♪, Σ, √, etc., y afortunadamente se pueden crear estos caracteres en Arduino.

Tabla ASCII resumida en valores decimales:

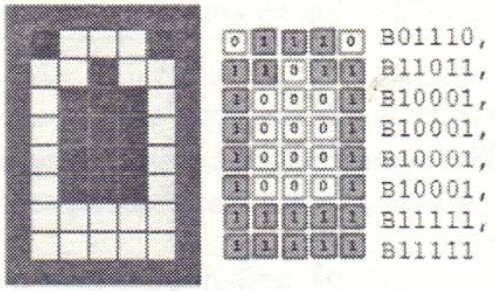
Símbolo	Nombre	Valor decimal	Símbolo	Nombre	Valor decimal
	Espacio	32	p		80
!	Cierre de exclamación	33	Q		81
"	Comilla doble	34	R		82
#	Numeral	35	S		83
$	Signo dólar	36	T		84
%	Porcentaje	37	U		85
&	Ampersand	38	V		86
'	Comilla	39	W		87
(	Paréntesis izquierdo	40	X		88
)	Paréntesis derecho	41	Y		89
*	Asterisco	42	Z		90
+	Signo más	43	[	Corchete izquierdo	91

Símbolo	Nombre	Valor decimal	Símbolo	Nombre	Valor decimal
,	Coma	44	\	Barra inversa	92
-	Signo menos	45	]	Corchete derecho	93
.	Punto	46	^	Intercalación	94
/	Barra oblicua	47	_	Guion bajo	95
0	Cero	48	´	Acento	96
1	Uno	49	a		97
2	Dos	50	b		98
3	Tres	51	c		99
4	Cuatro	52	d		100
5	Cinco	53	e		101
6	Seis	54	f		102
7	Siete	55	g		103
8	Ocho	56	h		104
9	Nueve	57	i		105
:	Dos puntos	58	j		106
;	Punto y coma	59	k		107
<	Menos que	60	l		108
=	Igual	61	m		109
>	Mayor que	62	n		110
?	Cierre de interrogación	63	o		111
@	Arroba	64	p		112
A		65	q		113
B		66	r		114
C		67	s		115
D		68	t		116
E		69	u		117
F		70	v		118
G		71	w		119
H		72	x		120
I		73	y		121
J		74	z		122
K		75	{	Llave apertura	123
L		76	|	Barra vertical	124
M		77	}	Llave de cierre	125
N		78	~	Equivalencia	126
O		79		No definido	127

Para consultar más caracteres, véase el Apéndice B. Tabla ASCII extendida.

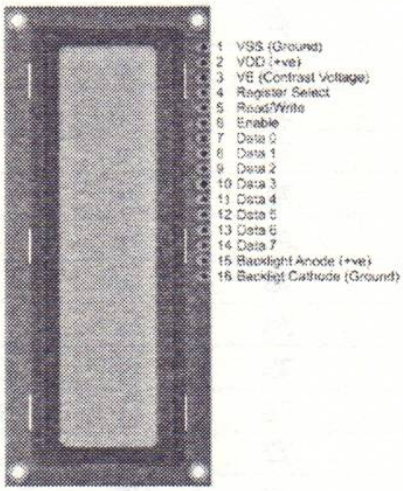
En una pantalla LCD cada carácter ASCII está definido en un tamaño de 8x5 píxeles.

La forma para crear caracteres personalizados es creando un mapa de bits. Estos bits definirán qué píxeles de la pantalla serán activos o desactivos.

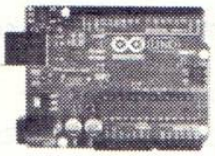


Como se puede observar en la línea B01110, que corresponde a la primera fila del carácter, donde los ceros son píxeles apagados y los unos son píxeles encendidos. Cada fila está precedida de la letra B.

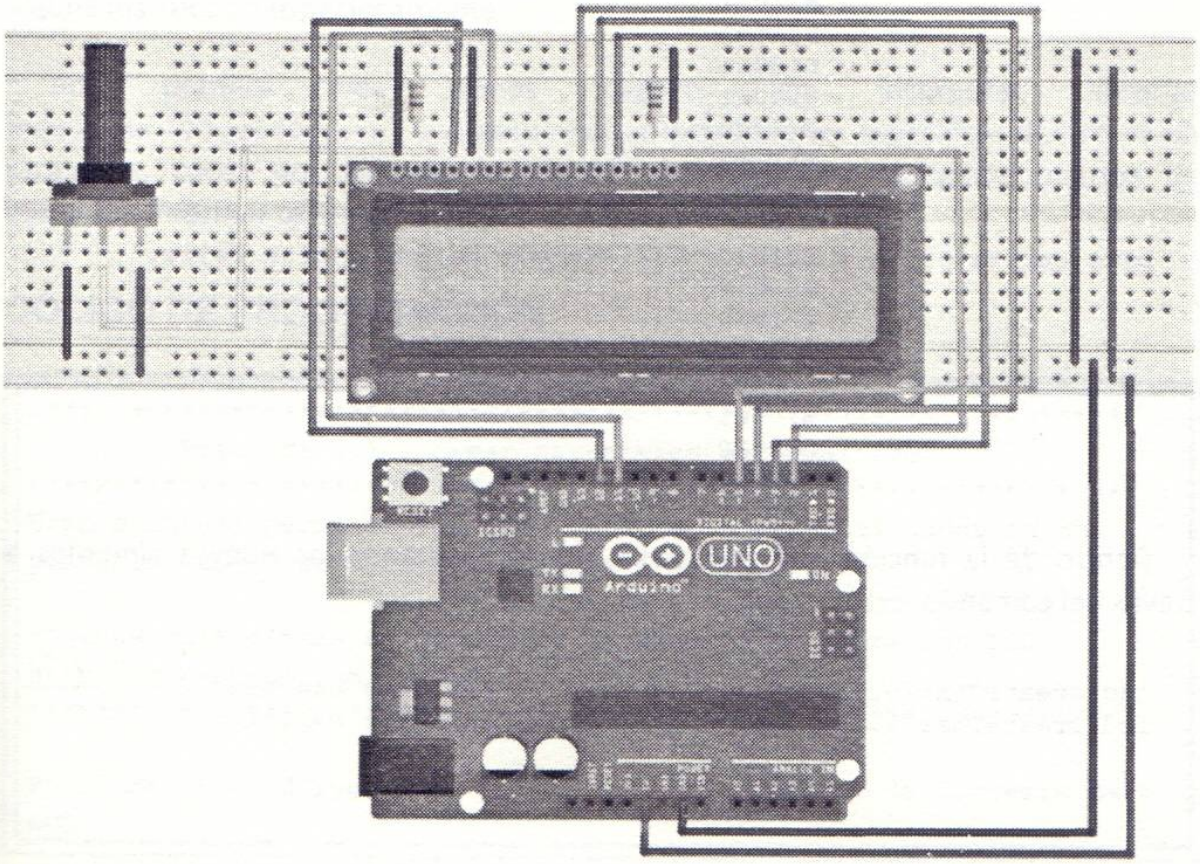
Los puertos de una pantalla LCD están descritos en la siguiente imagen:



MATERIALES

 Arduino UNO	 Protoboard	 2 Resistencias de 330 Ω	 1 Potenciómetro	 Pantalla LCD 16x2
 Cables				

CIRCUITO



DESCRIPCIÓN GENERAL DE LA PROGRAMACIÓN

Antes de empezar a trabajar en la programación, incluimos la librería `LiquidCrystal`. Esta librería permite controlar todo tipo de pantalla LCD, y se encuentra en la barra **Programa/Incluir** Librería.

```
#include <LiquidCrystal.h>
```

Es muy importante definir los puertos para conectar la pantalla a la tarjeta Arduino, nosotros usaremos los pines 2, 3, 4, 5, 11 y 12:

```
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
/*  PIN LCD Register Select  al Puerto digital  12 del Arduino
    PIN LCD Enable            al Puerto digital  11 del Arduino
    PIN LCD Data4            al Puerto digital  5  del Arduino
    PIN LCD Data5            al Puerto digital  4  del Arduino
    PIN LCD Data6            al Puerto digital  3  del Arduino
    PIN LCD Data7            al Puerto digital  2  del Arduino */
```

Para crear los símbolos definimos un mapa de bits dentro de una variable array tipo entero de 8 bits `byte`:

```
byte bateria[8] = { B01110,  // Símbolo de una batería descargada.
                    B11011,
                    B10001,
                    B10001,
                    B10001,
                    B10001,
                    B11111,
                    B11111 };

byte carita[8]  = { B01010,  // Símbolo de una carita feliz.
                    B01010,
                    B01010,
                    B00000,
                    B00100,
                    B10001,
                    B01110,
                    B00000 };
```

Dentro de la función principal `void setup()` creamos los nuevos símbolos a través del comando `creatChar`:

```
lcd.createChar(0, bateria); // creo el carácter batería.
lcd.createChar(1, carita);  // Creo el carácter carita.
```

Después inicializamos la pantalla para una LCD de 16x2, es decir, una pantalla de 16 caracteres y 2 filas:

```
lcd.begin(16, 2);
```

Configuramos el cursor de la LCD en la posición (columna = 0, fila = 0), es decir, establecemos la ubicación en la cual se mostrará el texto en la LCD. Después escribimos el nuevo carácter en la pantalla a partir de la posición definida:

```
lcd.setCursor(0, 0); // Posición para el nuevo carácter. (Col, Fil).
lcd.write(byte(0));  // Escribir en pantalla el carácter (batería)
```

Para imprimir un texto adicional, lo hacemos con la función `print`:

```
lcd.print(" Bateria Baja");
```

Para el carácter carita repetimos los mismos procedimientos anteriores:

```
lcd.setCursor(0, 1); // Posición para el nuevo carácter. (Col, Fil).
lcd.write(byte(1));  // Escribir en pantalla el carácter (carita)
lcd.print(" Carita Feliz");  // Presento un texto adicional.
```

Dentro de la función `void loop()` no se realiza ninguna acción o no se ejecuta alguna instrucción específicamente.

Por primera vez somos capaces de incorporar librerías (`#include <LiquidCrystal.h>`) para realizar instrucciones específicas dentro de un programa, como por ejemplo: configurar una pantalla, configurar puertos de conexión, imprimir texto, etc.

CÓDIGO DE PROGRAMACIÓN

```
/*****************************************************************
*        Práctica 4.1 - Crear caracteres personalizados.         *
*****************************************************************
Este programa permite presentar caracteres personalizados en una
pantalla LCD (Liquid Crystal Display) de 16x2.

Páginas online para el diseño de caracteres en pantallas LCD:
http://gotencool.com/lcdchar/
*****************************************************************/

#include <LiquidCrystal.h>              // Incluyo la librería para
LCD.
```

```
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);  // Definimos los puertos para
conectar la LCD.

// Conexión de los puertos de la pantalla LCD a la placa Arduino...
/*  PIN LCD RS      al PIN digital  D12 del Arduino
    PIN LCD Enable al PIN digital  D11 del Arduino
    PIN LCD D4      al PIN digital  D5  del Arduino
    PIN LCD D5      al PIN digital  D4  del Arduino
    PIN LCD D6      al PIN digital  D3  del Arduino
    PIN LCD D7      al PIN digital  D2  del Arduino       */

byte bateria[8] = { B01110, // Declaración de los bits del símbolo
batería
                    B11011,
                    B10001,
                    B10001,
                    B10001,
                    B10001,
                    B11111,
                    B11111 };

byte carita[8]  = { B01010, // Declaración de los bits del símbolo
carita
                    B01010,
                    B00000,
                    B00100,
                    B10001,
                    B01110,
                    B00000 };

void setup() {

    lcd.createChar(0, bateria); // Creo el carácter batería.
    lcd.createChar(1, carita);  // Creo el carácter carita.
    lcd.begin(16, 2);           // Configuramos para una pantalla LCD
de 16x2.
    lcd.setCursor(0, 0);        // Configuro la posición para
presentar el nuevo carácter. (Columna, Fila).
    lcd.write(byte(0));         // Escribo en pantalla el carácter
(batería)
    lcd.print(" Bateria Baja"); // Presento un comentario.
```

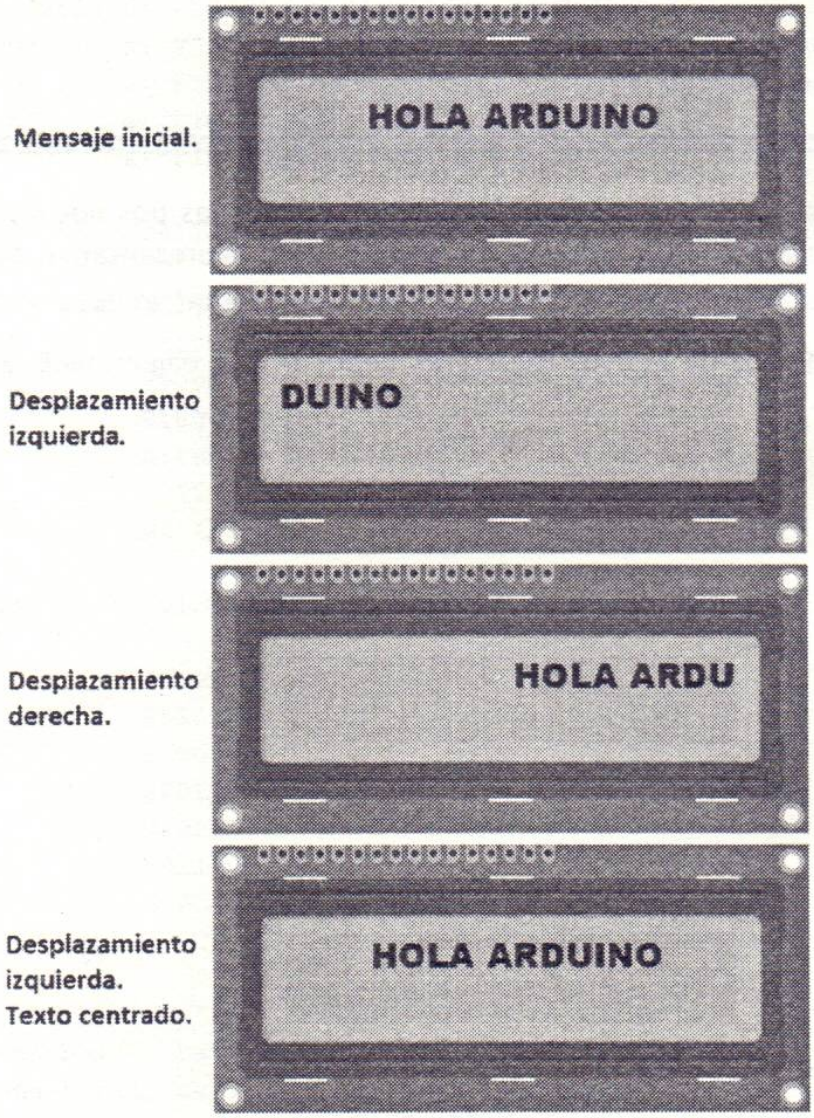
```
    lcd.setCursor(0, 1);         // Configuro la posición para
presentar el nuevo carácter. (Columna, Fila).
    lcd.write(byte(1));          // Escribo en pantalla el carácter
(carita)
    lcd.print(" Carita Feliz");     // Presento un comentario.
}

void loop() {
  // No hay código aquí.
}
```

Cuando se **Verifique** y **Cargue** el código veremos las dos nuevas imágenes, una batería descargada en la primera fila con un texto representativo `Batería Baja` y una carita feliz en la segunda fila con un texto representativo `Cara Feliz`.

Para el diseño de nuevos caracteres debes ir a la página web disponible en el código.

4.2 Letrero pasa-mensajes

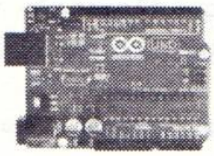
Para este primer ejemplo con pantallas LCD crearemos una carita feliz y una batería descargada para presentar en una pantalla LCD de 16x2 (16 caracteres de 2 filas).



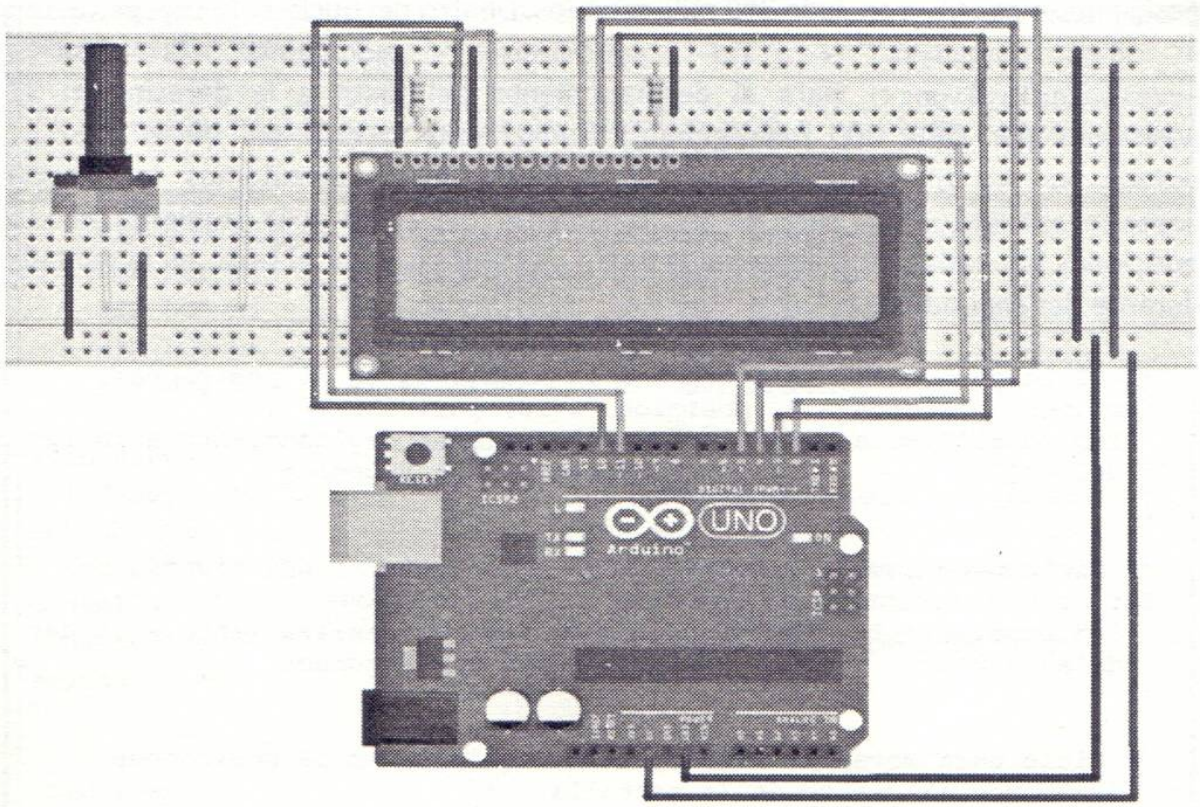
Para continuar con el uso de la pantalla LCD, en esta ocasión realizaremos el desplazamiento de texto de izquierda y derecha al igual que un letrero pasa-mensajes.

La librería `LiquidCrystal` permite controlar pantallas LCD para el desplazamiento de texto a través de las funciones `scrollDisplayLeft()` y `scrollDisplayRight()`. Estas funciones revierten la dirección del texto a presentar.

MATERIALES

 Arduino UNO	 Protoboard	 2 Resistencias de 330 Ω	 1 Potenciómetro	 Pantalla LCD 16x2
 Cables				

CIRCUITO



DESCRIPCIÓN GENERAL DE LA PROGRAMACIÓN

Al igual que el ejemplo anterior, incluimos la librería `LiquidCrystal`, y configuramos los puertos 2, 3, 4, 5, 11, y 12 de la tarjeta Arduino para conectar la pantalla LCD.

```
#include <LiquidCrystal.h>            // Librería de la pantalla LCD.
LiquidCrystal lcd(12, 11, 5, 4, 3, 2); /* Configuración de los
                                       puertos para conectar la LCD. */
```

Dentro de la función `void setup()` inicializamos la librería para una pantalla de 16x2, e imprimimos el texto a desplazar `Hola Arduino`.

```
lcd.begin(16, 2);           // Configuración de la pantalla LCD 16x2.
lcd.print("  Hola Arduino  ");// Imprimir el mensaje en la pantalla.
delay(1000);                  // Retardo de 1 segundo.
```

Dentro de la función principal `void loop()` ejecutamos las funciones de desplazamiento de texto a través de estructuras repetitivas `for`. Dentro del bucle `for(int posicion = 0; posicion < 16; posicion++)` ejecutamos la función `scrollDisplayLeft()` para el desplazamiento del texto a la izquierda en 16 posiciones o espacios en cada 300 milisegundos. Dentro del bucle `for(int posicion = 0; posicion < 32; posicion++)` ejecutamos la función `scrollDisplayRight()` para el desplazamiento del texto a la derecha en 32 posiciones por cada 300 milisegundos. Nuevamente dentro del bucle `for(int poisicion = 0; poisicion < 16; poisicion++)` ejecutamos la función `scrollDisplayRight()` para el desplazamiento del texto a la izquierda en 16 posiciones por cada 300 milisegundos para finalmente centrar el texto en la pantalla durante dos segundos.

```
// Ciclo para mover el mensaje a la izquierda en 16 posiciones.
for (int  posicion = 0; posicion  < 16; posicion++) {
  lcd.scrollDisplayLeft(); //Función para el deslizamiento a la izq.
  delay(300);              // Esperamos 0,3 segundos
}

// Ciclo para mover el mensaje a la derecha en 32 posiciones.
for (int posicion  = 0; posicion  < 32; posicion++) {
  lcd.scrollDisplayRight();//Función para el deslizamiento a la der.
  delay(300);              // Esperamos 0,3 segundos
}

/* Ciclo para mover el mensaje a la izquierda en 16 posiciones
   (volvemos al centro de la pantalla). */
for (int poisicion = 0; poisicion < 16; poisicion++) {
  lcd.scrollDisplayLeft();// Función para el deslizamiento a la izq.
```

```
    delay(300);                  // Esperamos 0,3 segundos
  }
  delay(2000);                   /* Espera 2 segundos para empezar a
                                 realizar todo el procedimiento anterior. */
```

Por primera vez somos capaces de utilizar las funciones de desplazamiento de texto en pantallas LCD para un estilo dinámico de presentación. Ahora serás capaz de desarrollar cualquier proyecto con este método.

CÓDIGO DE PROGRAMACIÓN

```
/*****************************************************************
*              Práctica 4.2 - Letrero pasa-mensajes.             *
*****************************************************************
Este programa permite realizar el deslizamiento de un mensaje de
texto en una pantalla LCD de 16x2.
*****************************************************************/

#include <LiquidCrystal.h>  // Incluimos la librería de la pantalla
LCD.

LiquidCrystal lcd(12, 11, 5, 4, 3, 2); //  Definimos los puertos para
conectar la LCD.

// Conexión de los puertos de la pantalla LCD a la placa Arduino...

/*  PIN LCD RS      al PIN digital  D12 del Arduino
    PIN LCD Enable  al PIN digital  D11 del Arduino
    PIN LCD D4      al PIN digital  D5  del Arduino
    PIN LCD D5      al PIN digital  D4  del Arduino
    PIN LCD D6      al PIN digital  D3  del Arduino
    PIN LCD D7      al PIN digital  D2  del Arduino       */

void setup() {
  lcd.begin(16, 2);              // Configuramos para una pantalla
LCD de 16x2.
  lcd.print("  Hola Arduino  "); // Imprimimos el mensaje en la
pantalla.
  delay(1000);                   // Insertamos un retardo de 1
segundo.
}

void loop() {
  for (int  posicion = 0; posicion  < 16; posicion++) { // Ciclo para
```

```
mover el mensaje a la izquierda en 16 posiciones.
    lcd.scrollDisplayLeft();  // Función para el deslizamiento del
mensaje por cada posición.
    delay(300);               // Esperamos 0,3 segundos
  }

  for (int posicion  = 0; posicion  < 32; posicion++) { // Ciclo
para mover el mensaje a la derecha en 32 posiciones.
    lcd.scrollDisplayRight(); // Función para el deslizamiento del
mensaje por cada posición.
    delay(300);               // Esperamos 0,3 segundos
  }

  for (int posicion = 0; posicion < 16; posicion++) { // Ciclo para
mover el mensaje a la izquierda en 16 posiciones (volvemos al centro
de la pantalla).
    lcd.scrollDisplayLeft();  // Función para el deslizamiento del
mensaje por cada posición.
    delay(300);               // Esperamos 0,3 segundos
  }

  delay(2000);                //Esperamos 2 segundos para empezar a
realizar todo el procedimiento anterior.
}
```

Cuando se **Verifique** y **Cargue** el código veremos el deslizamiento del texto `Hola Arduino` en la primera fila de la pantalla LCD.

El primer texto se desplaza hacia la izquierda hasta desaparecer por completo, después se desplaza hacia la derecha hasta desaparecer por completo, y por último vuelve a desplazarse hacia la izquierda hasta colocarse nuevamente en el centro de la pantalla durante dos segundos y después empieza a repetir todo el proceso de desplazamiento anterior.

QUINTA PRÁCTICA (TONOS Y MELODÍAS)

En esta práctica aprenderemos a generar melodías musicales, sonidos o alarmas a diferentes frecuencias y duración con un solo dispositivo a través de funciones propias.

Esta práctica es una apertura para el desarrollo de proyectos avanzados en Arduino, ya que recopila toda la información necesaria de las prácticas anteriores.

5.1 Tonos y melodías con Piezo Speaker o altavoz

En este ejemplo vamos a utilizar un circuito muy simple para producir sonidos con un Piezo Speaker. Para reproducir sonidos en Arduino, hablaremos de la capacidad de procesar señales PWM con el fin de reproducir música.

En este ejemplo, el código generará señales digitales y moduladas por ancho de pulso (PWM) para generar las diferentes notas musicales en los puertos digitales de salida PWM: 3, 5, 6, 9, 10 y 11.

En la siguiente tabla tenemos las notas musicales equivalentes:

Nota	Símbolo	Frecuencia	Periodo	Ancho de pulso (us)
Do	c	261	3830	1915
Re	d	294	3400	1700
Mi	e	329	3038	1519
Fa	f	349	2864	1432
Sol	g	392	2550	1275
La	a	440	2272	1136
Si	b	493	2028	1014
Do	C	523	1912	956

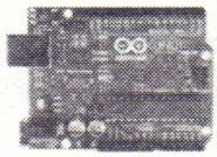
Para calcular los tonos se realiza mediante la siguiente fórmula matemática:

$$Ancho\ de\ pulso = \frac{Periodo}{2} = \frac{1}{2 * Frecuencia}$$

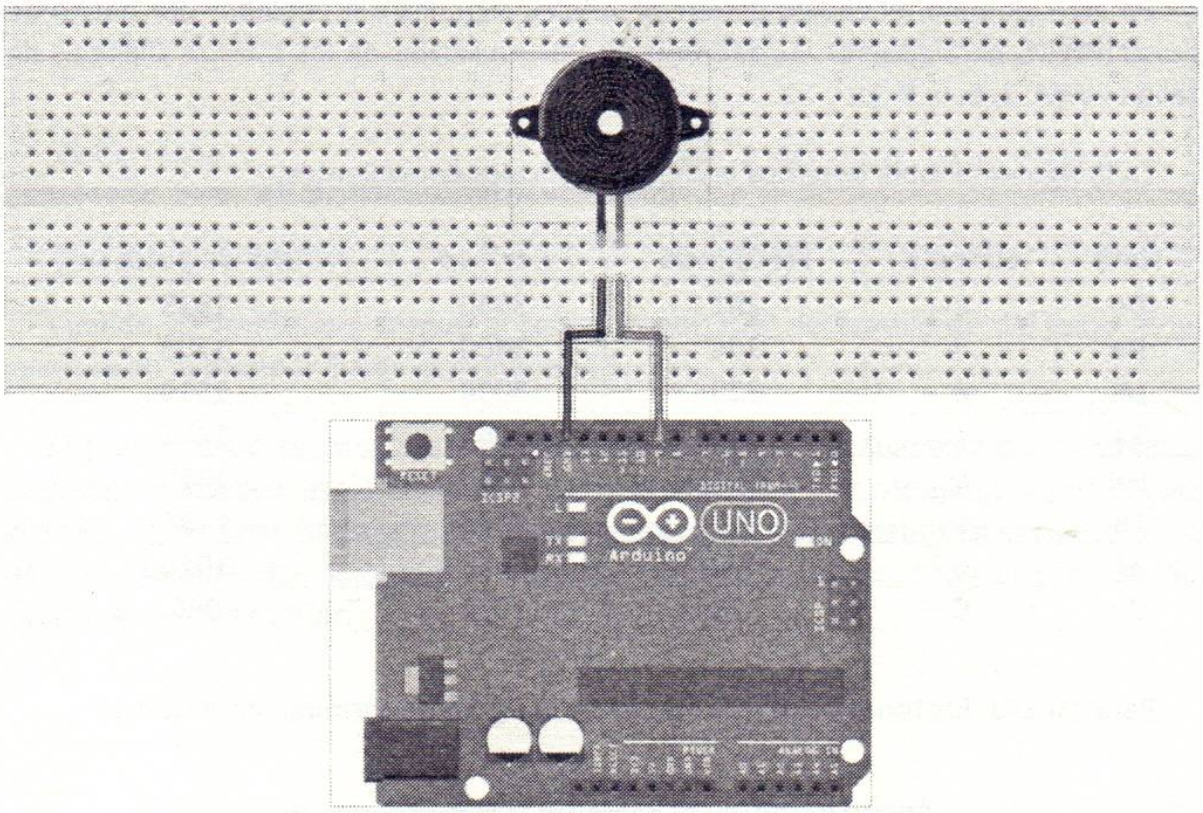
Un punto para recordar es que un Piezo Speaker tiene polaridad, los dispositivos comerciales tienen un cable rojo y un cable negro para conectarse correctamente a la tarjeta. Conectamos el cable negro a tierra o GND y el cable rojo a la salida del pin digital a utilizar de la tarjeta Arduino.

La melodía a reproducir se llama TAPION de la famosa serie japonesa DragonBall Z.

MATERIALES

			
Arduino UNO	Protoboard	Piezo Speaker	Cables

CIRCUITO



DESCRIPCIÓN GENERAL DE LA PROGRAMACIÓN

Para este ejemplo declaramos un puerto de salida para el speaker o altavoz:

```
int speakerPin = 9;
```

Para reproducir sonidos, es muy importante definir el tamaño de la melodía, es decir, el número de notas en total de la canción a reproducir. La canción a reproducir tiene 29 notas musicales y 4 espacios; en total, 33.

```
int longitud = 33;
```

En una variable array tipo `char` colocamos las notas musicales definidas en la tabla anterior más espacios para hacer pausas o sea 33 notas:

```
char notas[] = "ccfagfg deffgfe ccfagfg deffedcd ";
```

En otra variable array tipo `int` declaramos la duración de las notas musicales, esta variable es del mismo tamaño de la variable `notas[]`, es decir, de 33:

```
int golpes[] = { 1, 1, 4, 1, 1, 1, 4, 2, 1, 1, 4, 2, 1, 1, 4, 2, 1,
1, 4, 1, 1, 1, 4, 2, 1, 1, 4, 2, 1, 1, 1, 4, 2};
```

Declaramos una variable `int` para definir el tiempo de reproducción de las notas. Esta variable depende de si la canción ira lenta o rápida:

```
int tiempo = 250;
```

Dentro de la función principal `void` **`setup`**`()` configuramos el único puerto de salida para el speaker:

```
pinMode(speakerPin, OUTPUT);      /* Configuramos el PIN digital D9
                                     como salida. */
```

Dentro de la función principal `void` **`loop`**`()` ejecutamos las notas musicales dependiendo de los espacios y tiempos.

```
for (int i = 0; i < longitud; i++) {
  if (notas[i] == ' ') {
    delay(golpes[i] * tiempo);  // Retardo de acuerdo a los golpes.
  }
  else {
    Tocar_Nota(notas[i], golpes[i] * tiempo); /* Reproduce notas con
                                                 su duración. */
  }
  delay(tiempo / 2);       // Pausa entre notas.
}
```

Para reproducir la canción, utilizaremos nuestras propias funciones. La función `void Tocar_Tono(int tone, int duración)` reproduce las notas en el puerto de salida para cada nota dependiendo de los tiempos. La función `void Tocar_Nota(char nota, int duracion)` reproduce las notas dependiendo de su tono y duración definidos en la canción a reproducir:

```
// Una función para la reproducción de tonos.
void Tocar_Tono(int tone, int duracion) {
  for (long i = 0; i < duracion * 1000L; i += tone * 2) {
    delayMicroseconds(tone);  // Duración del tono en microsegundos.
    digitalWrite(speakerPin, LOW);  // Presenta el tono en bajo.
    delayMicroseconds(tone);  // Duración del tono en microsegundos.
  }
}
// Una función para la reproducción de notas.
void Tocar_Nota(char nota, int duracion) {
  // Variable local para el nombre de notas musicales existentes.
  char nombres[] = { 'c', 'd', 'e', 'f', 'g', 'a', 'b', 'C' };
   // Variable local para el ancho de pulso (us) de las notas.
  int tonos[] = { 1915, 1700, 1519, 1432, 1275, 1136, 1014, 956 };
  for (int i = 0; i < 8; i++) {     /* Reproducimos los tonos
                          correspondientes al nombre de las notas. */
    if (nombres[i] == nota) {
      Tocar_Tono(tonos[i], duracion); /* Llamamos a la función para
              la reproducción de tonos, según el tono y duración. */
    }
  }
}
```

Por primera vez somos capaces de generar melodías en Arduino con funciones propias y con señales a frecuencias correspondientes a cada nota musical. También usamos la función de tiempo `delayMicroseconds()` para generar señales digitales a altas frecuencias en los puertos digitales de salida PWM.

CÓDIGO DE PROGRAMACIÓN

```
/*****************************************************************
*       Práctica 5.1 - Tonos y melodías con Piezo Speaker        *
*****************************************************************
Este programa permite realizar la simulación de melodías con el
Piezo Speaker o altavoz.

La melodía reproducida se llama TAPION de la serie japonesa
DragonBall Z.
*****************************************************************/

int speakerPin = 9;                // Declaramos el PIN digital D9
para el Piezo Speaker.

int longitud = 33;                   // Número de notas que contiene
la melodía más uno.
char notas[] = "ccfagfg deffgfe ccfagfg deffedcd "; // Notas
musicales. Los espacios son notas vacías
```

```
int golpes[] = { 1, 1, 4, 1, 1, 1, 4, 2, 1, 1, 4, 2, 1, 1, 4, 2, 1,
1, 4, 1, 1, 1, 4, 2, 1, 1, 4, 2, 1, 1, 1, 4, 2}; // Duración de cada
una de las notas musicales.
int tiempo = 250;                    // Tiempo de duración para el toque
de notas.

void setup() {
  pinMode(speakerPin, OUTPUT);       // Configuramos el puerto digital
9 como salida.
}

void loop() {
  for (int i = 0; i < longitud; i++) {
    if (notas[i] == ' ') {
      delay(golpes[i] * tiempo);     //   Retardo de acuerdo a los
golpes.
    }
    else {
      Tocar_Nota(notas[i], golpes[i] * tiempo); // Reproduce notas
con su duración.
    }
    delay(tiempo / 2);       // Pausa entre notas.
  }
}

void Tocar_Tono(int tone, int duracion) {   // Una función para la
reproducción de tonos.
  for (long i = 0; i < duracion * 1000L; i += tone * 2) {
    digitalWrite(speakerPin, HIGH);        // Presenta el tono en alto.
    delayMicroseconds(tone);               // Duración del tono.
    digitalWrite(speakerPin, LOW);         // Presenta el tono en bajo.
    delayMicroseconds(tone);               // Duración del tono.
  }
}

void Tocar_Nota(char nota, int duracion) { // Una función para la
reproducción de notas.
  char nombres[] = { 'c', 'd', 'e', 'f', 'g', 'a', 'b', 'C' };
// Variable local para el nombre de notas musicales existentes.
  int tonos[] = { 1915, 1700, 1519, 1432, 1275, 1136, 1014, 956 };
// Variable local para el ancho de pulso (us) de las notas.

  for (int i = 0; i < 8; i++) {      // Reproducimos los tonos
correspondientes al nombre de las notas.
    if (nombres[i] == nota) {
```

```
      Tocar_Tono(tonos[i], duracion); // Llamamos a la función para
la reproducción de tonos, según el tono y duración.
    }
  }
}
```

Hacemos clic en **Verificar** para asegurarnos de no cometer ningún error, y por último en **Cargar** para subir el código a nuestra placa Arduino.

Cuando se ejecuta el código en la placa Arduino, podremos oír la melodía de la canción TAPION de la famosa serie animada japonesa DragonBall Z. Para obtener estas melodías es importante dejar bien definida la duración, frecuencia y tono para cada nota musical.

5.2 Monitor LCD de temperaturas y alarmas con luces indicadoras

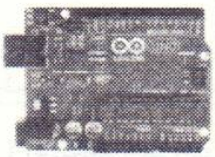
En este ejemplo realizaremos la presentación de temperaturas en una pantalla LCD con alarmas de alerta y luces indicadoras de temperaturas.

El funcionamiento del sistema consiste en leer la temperatura de un ambiente. La temperatura normal definida en este proyecto está comprendida entre 20 °C y 30 °C. Si la temperatura está dentro del rango permitido, imprimiremos en la pantalla la temperatura y un mensaje de texto "Temperatura Normal" y encenderemos un LED verde como indicador de normalidad.

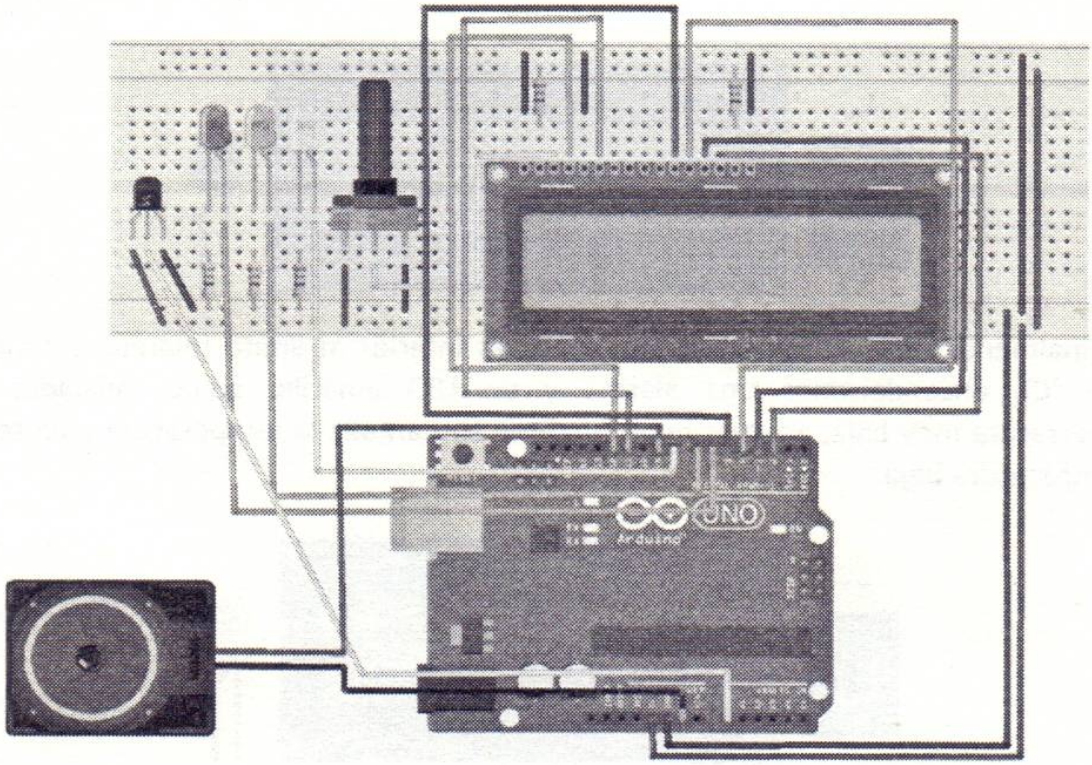
Si la temperatura ambiente es superior al límite máximo permitido (>30 °C), encenderemos una alarma y un LED rojo como indicador de temperatura muy elevada, además imprimiremos en pantalla la temperatura y un texto "Temperatura Alta".

Finalmente, si la temperatura ambiente es inferior al límite mínimo permitido (<20 °C), encenderemos una alarma y un LED amarillo como indicador de temperatura muy baja, además imprimiremos en pantalla la temperatura y un texto "Temperatura Baja".

MATERIALES

 Arduino UNO	 Protoboard	 3 LED (rojo, amarillo y verde)	 5 Resistencias de 330 Ω	 Piezo Speaker
 1 Potenciómetro	 1 Sensor LM35	 Pantalla LCD 16x2	 Cables	

CIRCUITO



DESCRIPCIÓN GENERAL DE LA PROGRAMACIÓN

Para la programación exportamos la librería para la pantalla LCD, y configuramos los puertos para conectar la pantalla a la tarjeta Arduino:

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
/*  PIN LCD Register Select  al Puerto digital  12 del Arduino
    PIN LCD Enable           al Puerto digital  11 del Arduino
    PIN LCD Data4            al Puerto digital  5  del Arduino
    PIN LCD Data5            al Puerto digital  4  del Arduino
    PIN LCD Data6            al Puerto digital  3  del Arduino
    PIN LCD Data7            al Puerto digital  2  del Arduino */
```

Declaramos variables para los puertos digitales de salida correspondientes a los LED y speaker, y una variable para el puerto analógico de entrada correspondiente al sensor de temperatura con sus variables para la lectura de datos y de cálculo de la temperatura:

```
int Sensor = A0;        // Puerto analógico para sensor LM35.
int ValorSensor = 0;    // Variable para lectura de datos analógicos.
int Temperatura = 0;    // Variable para el cálculo de temperatura.

int PinRojo = 6;        // Puerto digital 6 para el color rojo.
int PinVerde = 7;       // Puerto  digital 7 para el color verde.
int PinAmarillo = 8;    // Puerto  digital 8 para el color amarillo.
int PzSpeaker = 9;      // Puerto  digital 8 para el speaker.
```

Dentro de la función principal `void setup()` configuramos la pantalla LCD 16x2 y configuramos los puertos como salidas `OUTPUT` para los LED y speaker, y los inicializamos en `LOW`:

```
lcd.begin(16, 2);           // Configuración de la pantalla LCD 16x2.
delay(1000);                // Retardo de un segundo.
// Configuración de puertos digitales 6, 7, 8 y 10 como salidas.
pinMode(PinVerde,OUTPUT);
pinMode(PinRojo,OUTPUT);
pinMode(PinAmarillo,OUTPUT);
pinMode(PzSpeaker,OUTPUT);
// Inicializar puertos 6, 7 y 8 como nivel lógico 0.
digitalWrite(PinAmarillo, LOW);
digitalWrite(PinRojo, LOW);
digitalWrite(PinVerde, LOW);
```

Dentro de la función principal `void loop()` leemos los datos para el cálculo de la temperatura aplicando la fórmula del sensor LM35 para Arduino:

```
ValorSensor = analogRead(Sensor);
Temperatura = ValorSensor * 0.4899;  // Fórmula de temperatura.
```

Para limpiar la pantalla LCD de cualquier texto usamos la función:

```
lcd.clear();
```

Para imprimir la temperatura en la pantalla configuramos la columna y fila:

```
lcd.setCursor(0,0);// Primera col. y primera fil. a presentar texto.
lcd.print(Temperatura); //Presentamos la temperatura en la pantalla.
lcd.print("'C");  // Presentamos la unidad de medida de temperatura.
```

Dentro de la estructura lógica `if (Temperatura < 20)` ejecutamos la sentencia de temperatura baja donde imprimimos en la segunda fila de la pantalla un mensaje a través de la función `print`, activaremos una alarma a través de la función `playTone(x,y)` donde x e y son valores para definir la duración y frecuencia de la alarma, y encendemos el LED amarillo como indicador:

```
if (Temperatura < 20){ // Si la temperatura es menor a 20 grados.
  lcd.setCursor(0, 1);                  // Columna 0, fila 1.
  lcd.print("Temperatura Baja");        // Mensaje Temperatura baja.
  digitalWrite(PinAmarillo, HIGH);      // Encender led amarillo.
  digitalWrite(PinVerde, LOW);          // Apaga el resto de LED.
  digitalWrite(PinRojo, LOW);
  playTone(600, 500);                   // Función para la alarma.
  delay(100);                           // Esperamos 0,1 segundos.
  playTone(600, 900);                   // Función para la alarma.
  delay(100);                           // Esperamos 0,1 segundos.
}
```

Dentro de la estructura lógica `else if (Temperatura > 30)` ejecutamos la sentencia de temperatura alta, donde imprimimos en la segunda fila de la pantalla un mensaje a través de la función `print`, activaremos una alarma a través de la función `playTone(x,y)` donde `x` e `y` son valores para definir la duración y frecuencia de la alarma, y encendemos el LED rojo como indicador. La alarma es diferente para esta sentencia.

```
else if (Temperatura > 30){//Si la temperatura es mayor a 30 grados.
  lcd.setCursor(0, 1);                  // Columna 0, fila 1.
  lcd.print("Temperatura Alta");        // Mensaje Temperatura alta.
  digitalWrite(PinRojo, HIGH);          // Encender led rojo.
  digitalWrite(PinAmarillo, LOW);       // Apaga el resto de LED.
  digitalWrite(PinVerde, LOW);
  playTone(500, 600);                   // Función para la alarma.
  delay(100);                           // Esperamos 0,1 segundos.
  playTone(500, 800);                   // Función para la alarma.
  delay(100);                           // Esperamos 0,1 segundos.
}
```

Si la temperatura no cumple las condiciones anteriores, en la estructura lógica `else`, ejecutamos la sentencia de temperatura normal donde imprimimos en la segunda fila de la pantalla un mensaje a través de la función `print`, y encendemos el LED como indicador de normalidad. Ninguna alarma se ejecuta en esta estructura, mantenemos apagadas las alarmas a través de duración y frecuencia 0 `playTone(0, 0)`:

```
else{// Si no se cumple con las condiciones anteriores
  lcd.setCursor(0, 1);                  // Columna 0, fila 1.
  lcd.print("Temperatura Norm");        // Mensaje Temperatura normal.
  digitalWrite(PinVerde, HIGH);         // Encender LED verde
  digitalWrite(PinAmarillo, LOW);       // Apaga el resto de LED.
  digitalWrite(PinRojo, LOW);
  playTone(0, 0);                       // No toca ninguna alarma.
  delay(100);                           // Esperamos 0,1 segundos.
}
```

La función `void playTone(long duracion, int frecuencia)` define el ritmo de la alarma que va a ser reproducida según la duración y frecuencia. Esta función es similar al ejemplo anteriormente visto **"Tonos y melodías con Piezo Speaker o altavoz"**.

```
void playTone(long duracion, int frecuencia) {  /* Duración en
                              mili-segundos, Frecuencia en Hertz. */
  duracion *= 1000;                                 /* Por cada ciclo de
         la función multiplicamos la variable duración por 1000. */
  int periodo = (1.0 / frecuencia) * 1000000;    /* Periodo o ancho
                             de pulso de las señales de salida. */
  long lapzo_de_tiempo = 0;                         /* Variable para el
              lapso de tiempo para definir la duración del tono. */
  while (lapzo_de_tiempo < duracion) {
    digitalWrite(PzSpeaker,HIGH);              /* Envía el ritmo del
                                                  tono al puerto 10. */
    delayMicroseconds(periodo / 2);             /* La duración en
     alto del tono es la mitad del periodo de la señal de salida. */
    digitalWrite(PzSpeaker, LOW);
    delayMicroseconds(periodo / 2);
    lapzo_de_tiempo += (periodo);                /* El lapso de
                           tiempo se incrementa por cada periodo. */
  }
}
```

En este ejemplo por primera vez usamos la estructura repetitiva `while` para ejecutar un segmento de código hasta que se cumpla la condición (`lapzo_de_tiempo < duracion`). A diferencia del bucle `for`, el bucle `while` se ejecuta indefinidamente hasta su condición lógica.

CÓDIGO DE PROGRAMACIÓN

```
/*****************************************************************
*     Práctica 5.2 - Monitor LCD de temperaturas, alarmas y      *
*                    luces indicadoras.                          *
*****************************************************************
Este programa permite realizar el proceso de medida de
temperaturas con sensor LM35 con alarma de emergencia de
temperaturas altas o bajas, y luces indicadoras de temperaturas
para tres rangos (baja = amarillo, normal = verde y alta = roja).
*****************************************************************/

#include <LiquidCrystal.h>              // Incluimos librería LCD.

LiquidCrystal lcd(12, 11, 5, 4, 3, 2); // Inicializamos la librería
LCD con sus respectivos puertos de configuración.

int Sensor = A0;                        // Asignamos el PIN analógico
A0 para el sensor LM35.
int ValorSensor = 0;                    // Variable para el
almacenamiento del valor del sensor.
int Temperatura = 0;                    // Variable para almacenar el
valor de la temperatura.

int PinRojo = 6;                        // Asignación del puerto
digital 6 para el color rojo.
int PinVerde = 7;                       // Asignación del puerto
digital 7 para el color verde.
int PinAmarillo = 8;                    // Asignación del puerto
digital 8 para el color amarillo.
int PzSpeaker = 9;                      // Asignación del puerto
digital 10 para el Piezo Speaker.

void setup() {
  lcd.begin(16, 2);                     // Configuramos la pantalla
LCD para el número de columnas y filas (16x2).
  delay(1000);                          // Esperamos un segundo.
  // Configuramos los puertos digitales 6, 7, 8 y 10 como salidas.
  pinMode(PinVerde,OUTPUT);
  pinMode(PinRojo,OUTPUT);
  pinMode(PinAmarillo,OUTPUT);
  pinMode(PzSpeaker,OUTPUT);
  // Inicializamos los  puertos 6, 7 y 8 como nivel lógico 0.
  digitalWrite(PinAmarillo, LOW);
  digitalWrite(PinRojo, LOW);
```

```
  digitalWrite(PinVerde, LOW);
}

void loop() {
  ValorSensor = analogRead(Sensor);     // Leemos el valor del sensor.
  Temperatura = ValorSensor * 0.4899;  // (Multiplicamos por una
constante de 0.4899 = ((5/1023)/0.01) para obtener el valor del
sensor en voltios.
  lcd.clear();                          // Limpiamos la pantalla LCD
de cualquier carácter.
  lcd.setCursor(0, 0);                  // Configuramos en la columna
y fila a presentar el valor y el mensaje de temperatura.
  lcd.print(Temperatura);               // Presentamos el valor de la
temperatura en la pantalla.
  lcd.print("'C");                      // Presentamos un mensaje en
la pantalla.

  if (Temperatura < 20){ // Si la temperatura es menor a 20 grados.
    lcd.setCursor(0, 1);                // Configuramos en la primera
columna y segunda fila el mensaje de temperatura baja.
    lcd.print("Temperatura Baja");      // Presentamos el mensaje
Temperatura baja.
    digitalWrite(PinAmarillo, HIGH);    // Enciende el led amarillo.
    digitalWrite(PinVerde, LOW);        // Apaga el resto de LED.
    digitalWrite(PinRojo, LOW);
    playTone(600, 500);                 // Función para el toque de
una alarma con su duración y frecuencia.
    delay(100);                         // Esperamos 0,1 segundos.
    playTone(600, 900);                 // Función para el toque de
una alarma con su duración y frecuencia.
    delay(100);                         // Esperamos 0,1 segundos.
  }
  else if (Temperatura > 30){ // Si la temperatura es mayor a 30
grados.
    lcd.setCursor(0, 1);                // Configuramos en la primera
columna y segunda fila el mensaje de temperatura alta.
    lcd.print("Temperatura Alta");      // Presentamos el mensaje
Temperatura alta.
    digitalWrite(PinRojo, HIGH);        // Enciende el led rojo.
    digitalWrite(PinAmarillo, LOW);     // Apaga el resto de LED.
    digitalWrite(PinVerde, LOW);
    playTone(500, 600);                 // Función para el toque de
una alarma con su duración y frecuencia.
    delay(100);                         // Esperamos 0,1 segundos.
    playTone(500, 800);                 // Función para el toque de
```

```
una alarma con su duración y frecuencia.
    delay(100);                          // Esperamos 0,1 segundos.
  }
  else{// Si no se cumple con las condiciones anteriores
    lcd.setCursor(0, 1);                 // Configuramos en la primera
columna y segunda fila el mensaje de temperatura normal.
    lcd.print("Temperatura Norm");       // Presentamos el mensaje
Temperatura normal.
    digitalWrite(PinVerde, HIGH);        // Enciende el led verde
    digitalWrite(PinAmarillo, LOW);      // Apaga el resto de LED.
    digitalWrite(PinRojo, LOW);
    playTone(0, 0);                      // No tocamos ningún tono de
alarma.
    delay(100);                          // Esperamos 0,1 segundos.
  }
}

// Función para el toque de tonos de alarmas a diferente duración y
frecuencia.
void playTone(long duracion, int frecuencia) {  // Duración en mili-
segundos, Frecuencia en Hertz.
  duracion *= 1000;                              // Por cada ciclo de
la función multiplicamos la variable duración por 1000.
  int periodo = (1.0 / frecuencia) * 1000000;    // // Periodo o ancho
de pulso de las señales de salida.
  long lapzo_de_tiempo = 0;                      // Variable para el
lapso de tiempo para definir la duración del tono.
  while (lapzo_de_tiempo < duracion) {
    digitalWrite(PzSpeaker,HIGH);                // Envía el ritmo del
tono al puerto 10.
    delayMicroseconds(periodo / 2);              // La duración en
alto del tono es la mitad del periodo de la señal de salida.
    digitalWrite(PzSpeaker, LOW);
    delayMicroseconds(periodo / 2);
    lapzo_de_tiempo += (periodo);                // El lapso de tiempo
se incrementa por cada periodo.
  }
}
```

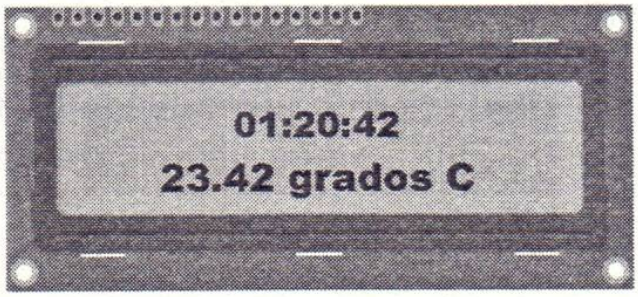
Cuando se **Verifique** y **Cargue** el código se verá en pantalla la temperatura leída por el sensor LM35.

Para el procesamiento de temperatura se ha definido un rango normal comprendido entre 20 y 30 grados Celsius. Si la temperatura es menor a 20 grados Celsius se activa una alarma y enciende el LED amarillo, es decir, la temperatura es baja. Si la temperatura es mayor a 30 grados Celsius se activa una alarma y enciende el LED rojo, es decir, la temperatura es alta. Pero si no se cumplen estas dos condiciones, no activa ninguna alarma y mantiene encendido el LED verde, o sea, la temperatura es normal.

Las alarmas para temperaturas altas y bajas son a diferentes frecuencias y duración de tonos.

5.3 Reloj termómetro en pantalla LCD

Para concluir con esta práctica vamos a realizar la lectura de temperatura con el sensor LM35 y un reloj (HH:MM:SS), expuestos en una pantalla LCD. Además, tiene la capacidad de igualar el tiempo con un solo pulsador.



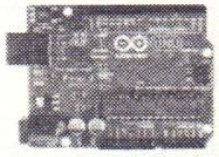
Cuando empieza a pasar el tiempo, el reloj inicia desde la hora 00:00:00, en la primera fila de la pantalla LCD.

Para igualar el tiempo a través del pulsador, este incrementará en 1 los minutos hasta 60 y después incrementará las horas en 1 hasta 24.

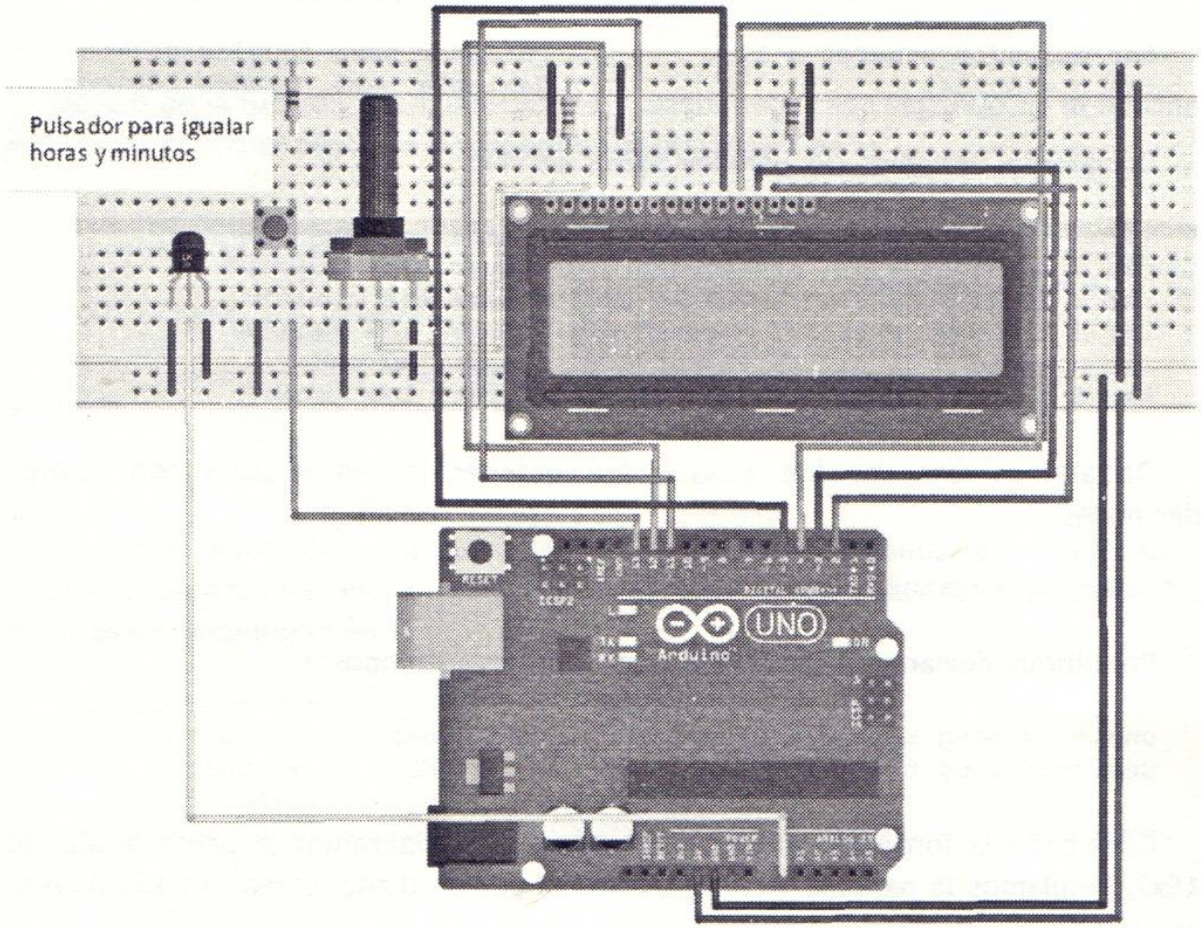
La temperatura se expondrá en un formato decimal con dos dígitos después del punto decimal en la segunda fila de la pantalla LCD.

Para darle más estilo a este ejemplo, los textos en la pantalla LCD se colocarán centrados.

MATERIALES

 Arduino UNO	 Protoboard	 2 Resistencias de 330 Ω	 1 Resistencia de 4,7 KΩ	 1 Potenciómetro
 1 pulsador	 Pantalla LCD 16x2	 Cables		

CIRCUITO



DESCRIPCIÓN GENERAL DE LA PROGRAMACIÓN

Para la programación exportamos la librería para la pantalla LCD, y configuramos los puertos para conectar la pantalla a la tarjeta Arduino:

```
#include <LiquidCrystal.h>        // Incluir la librería para LCD.
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
/*   PIN LCD Register Select  al Puerto digital  12 del Arduino
     PIN LCD Enable           al Puerto digital  11 del Arduino
     PIN LCD Data4            al Puerto digital  5  del Arduino
     PIN LCD Data5            al Puerto digital  4  del Arduino
     PIN LCD Data6            al Puerto digital  3  del Arduino
     PIN LCD Data7            al Puerto digital  2  del Arduino */
```

Declaramos el puerto analógico A0 para el sensor LM35:

```
int analogico_A0 = A0;
```

Declaramos el puerto digital 13 para el pulsador, y su variable para leer sus estados lógicos:

```
int pulsador_minutos = 13;        // Pulsador para igualar tiempo.
int p_minutos;                    // Variable para estados lógicos.
```

Declaramos variables `int` para los dígitos del reloj:

```
int u_segundos = 0;  // Primer dígito para los segundos.
int d_segundos = 0;  // Segundo dígito para los segundos.
int u_minutos  = 0;  // Primer dígito para los minutos.
int d_minutos  = 0;  // Segundo dígito para los minutos.
int u_horas    = 0;  // Primer dígito para las horas.
int d_horas    = 0;  // Segundo dígito para las horas.
```

Declaramos una variable `float` para obtener la temperatura con valores decimales:

```
float temperatura = 0;
```

Por último, declaramos dos variables para obtener tiempos:

```
unsigned long timer1 = 0;
unsigned long timer2 = 0;
```

Dentro de la función principal `void setup()` inicializamos la pantalla LCD de 16x2, limpiamos la pantalla y configuramos el puerto digital como entrada para el pulsador:

```
  lcd.begin(16,2);        // Pantalla LCD de 16x2.
  lcd.clear();            // Limpiar pantalla LCD de cualquier carácter.
  pinMode( pulsador_minutos, INPUT);//Puerto de entrada para puerto
13.
```

Dentro de la función principal `void loop ()` creamos nuestras propias funciones para ser llamadas y realizar los procesos de adquirir los segundos, minutos, horas, leer el pulsador para igualar el reloj en minutos y presentar el reloj en la pantalla LCD:

```
  unsegundo();              // Llamar la función unsegundo().
  puestaenhora();           // Llamar la función puestaenhora().
  contadorhoras();          // Llamar la función contadorhoras().
  lcdhoras();               // Llamar la función lcdhoras().
```

Dentro de la estructura lógica `if (u_segundos == 0)` adquirimos los datos del sensor de temperatura y presentar en la pantalla LCD cada 10 segundos:

```
if (u_segundos == 0) {
    sensorlm35();
    lcdtemperatura();
}
```

Dentro de la función `void unsegundo()` adquirimos cada segundo de la función `millis()`, el primer dígito de los segundos incrementará en 1:

```
void unsegundo() {
  timer2=(millis()/1000); /* Obtener los segundos a partir de la
                                                función millis(). */
  if (timer1 != timer2) {
    timer1=timer2;
    u_segundos++; // Unidades de segundo se incrementan cada
segundo.
  }
}
```

Dentro de la función `void puestaenhora()` leemos los estados del pulsador para igualar el tiempo del reloj en minutos. Cada vez que se presiona el pulsador los minutos incrementarán en 1:

```
void puestaenhora() {
  p_minutos=digitalRead(pulsador_minutos);
  if (p_minutos == HIGH){
    u_minutos++;
    delay(100);      // Retardo para eliminar rebotes en el pulsador.
  }
}
```

Dentro de la función `void contadorhoras()` realizamos el incremento de los segundos, minutos y horas por dígitos. Si el primer dígito de los segundos incrementa hasta 10, el segundo dígito incrementa en 1 hasta 6, esto es para obtener 60 segundos del tiempo real.

```
if (u_segundos == 10) {     /* Incrementamos el segundo dígito de los
                                          segundos cada 10seg. */
    u_segundos = 0;
    d_segundos++;
}
```

Por cada 60 segundos, el primer dígito de los minutos incrementa en 1 hasta 10.

```
if (( d_segundos == 6) && (u_segundos == 0)) {  /* Incrementamos las
                        unidades de los minutos por cada 60seg. */
    d_segundos = 0;
    u_minutos++;
}
```

Por cada 10 minutos, el segundo dígito de los minutos incrementa en 1 hasta 60, esto es para obtener 60 minutos del tiempo real.

```
if (u_minutos == 10) {      /* Incrementamos el segundo dígito de los
                                                    minutos cada 10min. */
    u_minutos =0;
    d_minutos++;
}
```

Cada 60 minutos, el primer dígito de las horas incrementa en 1 hasta 10.

```
if ((d_minutos == 6) && (u_minutos == 0)) {      /* Incrementamos las
                                unidades de las horas por cada 60min. */
    d_minutos =0;
    u_horas++;
}
```

Por cada 10 horas, el segundo dígito de las horas incrementa en 1 hasta 2, esto es para obtener las 24 horas del tiempo real.

```
if (u_horas == 10) {            /* Incrementamos el segundo dígito de
                                      las horas por cada 10 horas. */
    u_horas =0;
    d_horas++;
}
```

Finalmente por cada 24 horas, los dígitos de las horas se reinician a 0 para volver a las 0 horas.

```
if ((d_horas == 2) && (u_horas == 4)) {     /* Reiniciamos a cero las
                                    horas y minutos a las 24 horas. */
    u_horas =0;
    d_horas =0;
}
```

Dentro de la función `void lcdhoras()` imprimimos en pantalla los datos para las horas, minutos y segundos:

```
void lcdhoras() {
 lcd.setCursor(4,0);     // Coloca el cursor en la columna 4 y fila 0
 lcd.print(d_horas);     // Imprime horas.
 lcd.print(u_horas);
 lcd.print(":");
 lcd.print(d_minutos);   // Imprime minutos.
 lcd.print(u_minutos);
 lcd.print(":");
 lcd.print(d_segundos); // Imprime segundos.
 lcd.print(u_segundos);
}
```

Dentro de la función `void sensorlm35()` leemos los datos del sensor LM35 y el cálculo de temperatura para Arduino:

```
void sensorlm35() {
  temperatura = analogRead(analogico_A0);
  temperatura = temperatura * 0.488;
}
```

Finalmente dentro de la función `void lcdtemperatura()` imprimimos en la segunda fila de la pantalla la temperatura del sensor en un formato de dos dígitos decimales:

```
void lcdtemperatura() {
 lcd.setCursor(1,1);       // Colocar el cursor en la col. 1, fil. 1.
 lcd.print(temperatura,2);// Muestra la temperatura en pantalla LCD
 lcd.print(" grados C");
}
```

CÓDIGO DE PROGRAMACIÓN

```
/*****************************************************************
*        Práctica 5.3 - Reloj termómetro en pantalla LCD.        *
*****************************************************************
Este programa permite la vizualización de horas, minutos, segundos
y temperatura en una pantalla LCD.
*****************************************************************/

#include <LiquidCrystal.h>        // Incluyo la librería para LCD.

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);// Inicializo librería para
LCD.

int analogico_A0 = A0;            // Puerto analógico para sensor
LM35.
int pulsador_minutos = 13;        // Pulsador para igualar tiempo.
int p_minutos;

int u_segundos = 0;               // Unidad y decenas para los seg.,
min., y horas.
int d_segundos = 0;
int u_minutos  = 0;
int d_minutos  = 0;
int u_horas    = 0;
int d_horas    = 0;
```

```
float temperatura = 0;                  // Variable para el cálculo de
temperatura.

unsigned long timer1 = 0;
unsigned long timer2 = 0;

void setup() {
  lcd.begin(16,2);                      // Número de columnas y filas del
display LCD.
  lcd.clear();                          // Limpiamos la pantalla LCD de
cualquier carácter.
  pinMode( pulsador_minutos, INPUT);// Iniciamos con entrada el pin
13.
}

void loop() {
  unsegundo();                          // Llamamos a la función
unsegundo().
  puestaenhora();                       // Llamamos a la función
puestaenhora().
  contadorhoras();                      // Llamamos a la función
contadorhoras().
  lcdhoras();                           // Llamamos a la función lcdhoras().

  if (u_segundos == 0) {                // Para cada 10 segundos muestra la
temperatura en pantalla LCD.
    sensorlm35();
    lcdtemperatura();
  }
}

// Función temporizador para segundos.
void unsegundo() {
  timer2=(millis()/1000);               // Obtengo los segundos a partir de
la función millis().
  if (timer1 != timer2) {
    timer1=timer2;
    u_segundos++;                       // Unidades de segundo se incrementa
cada segundo.
  }
}

// Función puesta en hora.
void puestaenhora() {
  p_minutos=digitalRead(pulsador_minutos);
```

```
  if (p_minutos == HIGH){
    u_minutos++;
    delay(100);                    // Tiempo para eliminar rebotes en
el pulsador.
  }
}

// Función que determina los segundos, minutos y horas.
void contadorhoras() {
  if (u_segundos == 10) {          // Incrementamos el segundo dígito
de los segundos cada 10seg.
    u_segundos = 0;
    d_segundos++;
  }
  if (( d_segundos == 6) && (u_segundos == 0)) { // Incrementamos las
unidades de los minutos cada 60seg.
    d_segundos = 0;
    u_minutos++;
  }

  if (u_minutos == 10) {           // Incrementamos el segundo dígito
de los minutos cada 10min.
    u_minutos =0;
    d_minutos++;
  }
  if ((d_minutos == 6) && (u_minutos == 0)) { // Incrementamos las
unidades de las horas por cada 60min.
    d_minutos =0;
    u_horas++;
  }

  if (u_horas == 10) {             // Incrementamos el segundo dígito
de las horas por cada 10 horas.
    u_horas =0;
    d_horas++;
  }
  if ((d_horas == 2) && (u_horas == 4)) {  // Reseteamos a cero las
horas y minutos a las 24 horas.
    u_horas =0;
    d_horas =0;
  }
}

// Función que muestra las horas, minutos y segundos en display LCD
void lcdhoras() {
```

```
 lcd.setCursor(4,0);              // Coloca el cursor en la columna 4
y fila 0
 lcd.print(d_horas);              // Muestra las horas, minutos y
segundos en un display LCD
 lcd.print(u_horas);
 lcd.print(":");
 lcd.print(d_minutos);
 lcd.print(u_minutos);
 lcd.print(":");
 lcd.print(d_segundos);
 lcd.print(u_segundos);
}

// Función que obtiene el valor de la temperatura mediante el sensor
LM35 en °C.
void sensorlm35() {
  temperatura = analogRead(analogico_A0);
  temperatura = temperatura * 0.488;
}

// Función que muestra la temperatura en pantalla LCD
void lcdtemperatura() {
 lcd.setCursor(1,1);              // Coloca el cursor en la columna 1
y fila 1
 lcd.print(temperatura,2);        // Muestra la temperatura en
pantalla LCD
 lcd.print(" grados C");
}
```

Cuando se **Verifique** y **Cargue** el código se verán en pantalla la hora y la temperatura leída por el sensor LM35.

El pulsador permite colocar la hora correspondiente al tiempo real del día, la temperatura se presenta en la pantalla cada 10 segundos, es decir, se actualiza cada 10 segundos. El formato del tiempo es HH:MM:SS.

SEXTA PRÁCTICA (MOTORES)

En este capítulo aprenderemos a controlar motores de corriente continua (DC). Para el control de velocidad de un motor DC en Arduino vamos a utilizar los puertos de salida PWM. Para motores DC se usará el integrado L293D que es un circuito en puente H. Este dispositivo permite controlar un motor en ambos sentidos de giro, y sobre todo como protector y convertidor de potencia.

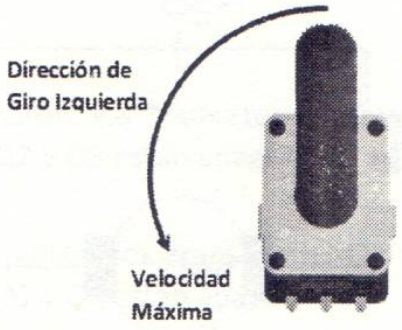
También aprenderemos a controlar la posición de giro de un servomotor. Al igual que cualquier motor DC, se usan los puertos digitales de salida PWM para el control de giro angular de estos dispositivos.

6.1 Control de velocidad de un motor DC con un potenciómetro

En este ejemplo vamos a realizar el control de un motor DC (dirección de giro y velocidad) por PWM dependiendo de la posición de un potenciómetro.

Es muy importante alimentar el motor DC con otra fuente de poder, para evitar ruido eléctrico generado por el motor y sobre todo evitar dañar la placa Arduino. El motor lo alimentaremos con una batería de 9 V y la tarjeta Arduino con el cable USB desde la PC.

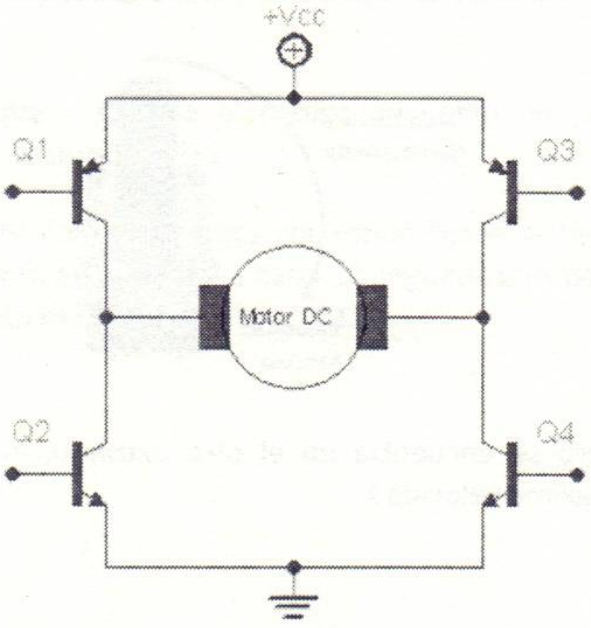
Si el potenciómetro se encuentra en un extremo, el motor girará en un sentido a máxima velocidad.



Si el potenciómetro se encuentra en el otro extremo, el motor girará en un sentido contrario a máxima velocidad.

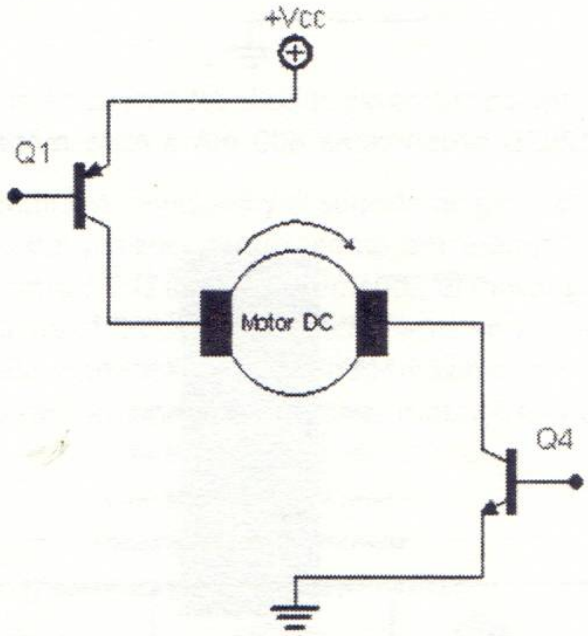
Pero si el potenciómetro está en una posición media, el motor no girará.

El circuito de control para un motor DC es el integrado L293D. Este dispositivo puede controlar dos motores DC independientemente en un sistema puente en H para el control del sentido de giro usando 4 transistores.



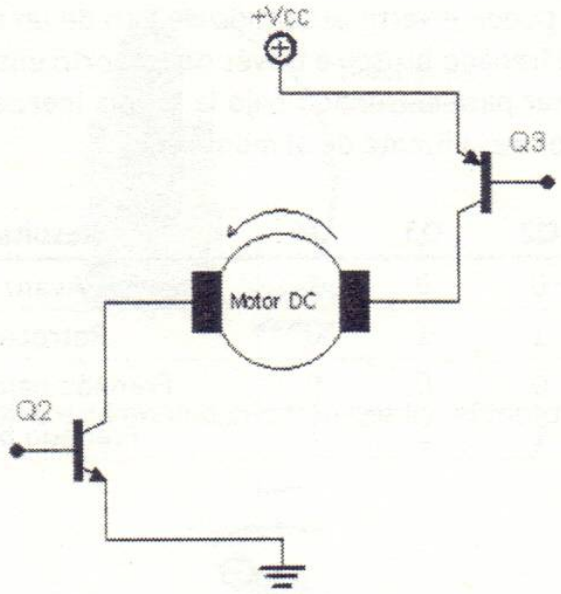
En el puente H se puede invertir el sentido de giro de un motor DC, pero también se puede usar para el frenado brusco a través de un corto entre los bornes del motor, o incluso se puede usar para el frenado bajo la propia inercia del motor mediante la desconexión del motor de la fuente de alimentación.

Q1	Q2	Q3	Q4	Resultado
1	0	0	1	Avanzar
0	1	1	0	Retroceder
0	0	0	0	Frenado bajo inercia
1	1	1	1	Frenado brusco



Durante el avance del motor, los transistores Q1 y Q4 están en estado de conducción y los transistores Q2 y Q3 están apagados o en estado de no conducción.

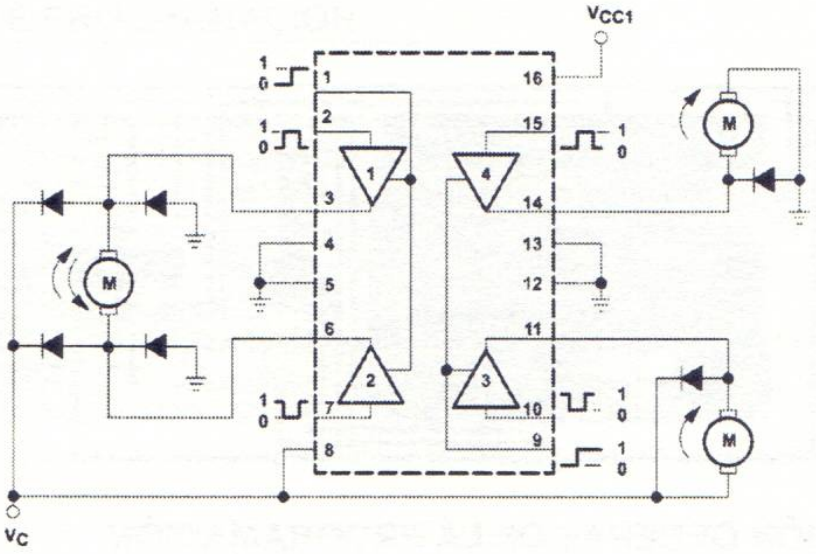
Durante el retroceso del motor, los transistores Q2 y Q3 están en estado de conducción y los transistores Q1 y Q4 están apagados o en estado de no conducción.



Este dispositivo L293D proporciona 600 mA a cada motor y soporta un voltaje de 4,5 a 36 V.

Existen algunos esquemas para usar este dispositivo. El esquema que se usará es conectar el motor a los puertos de salida 3 y 6 del dispositivo, y los puertos de entrada 2 y 7 son de control por PWM. Para controlar el giro y velocidad de un motor, las señales de PWM en los puertos son opuestos, es decir, si la señal en el puerto 2 va en aumento, la señal en el puerto 7 va en decremento, dentro de un rango de 0 a 255.

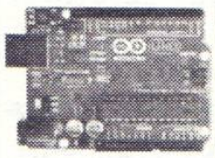
El motor frenará cuando las señales PWM en los dos puertos sean del mismo valor (128 aproximadamente). Las velocidades máximas del motor son alcanzadas en los valores altos de PWM en cada puerto, es decir, la velocidad máxima para un sentido de giro, la señal PWM en el puerto 2 será de 255 y en el puerto 7 será de 0, y la velocidad máxima para el sentido de giro opuesto, la señal PWM en el puerto 2 será de 0 y en el puerto 7 será de 255.



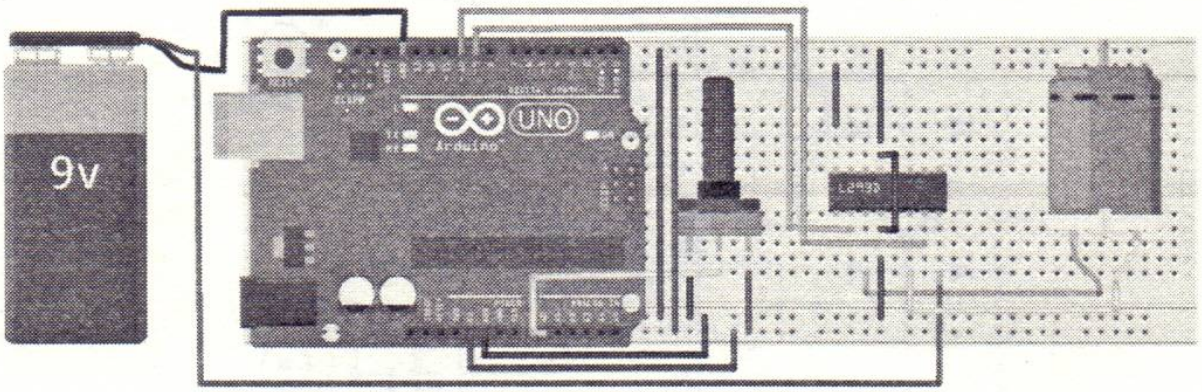
Para este esquema no se usarán los diodos externos, porque internamente ya se encuentran en el dispositivo.

El potenciómetro definirá la velocidad y el sentido de giro del motor dependiendo de los valores resistivos del potenciómetro, como por ejemplo, si la resistencia del potenciómetro se encuentra a 0 Ω aproximadamente, el motor girará en un sentido a velocidad máxima; si la resistencia del potenciómetro se encuentra al máximo, el motor girará en sentido contrario a velocidad máxima; y si la resistencia del potenciómetro se encuentra en un valor medio, el motor frenará y se mantendrá en nulo.

MATERIALES

 Arduino UNO	 Protoboard	 Motor DC	 Batería de 9V	 Porta-baterías de 9V
 Potenciómetro	 L293D	 Cables		

CIRCUITO



DESCRIPCIÓN GENERAL DE LA PROGRAMACIÓN

Para la programación declaramos dos puertos (9 y 10) del Arduino para las entradas PWM en el dispositivo, y un puerto analógico A0 para el potenciómetro:

```
int pin2 = 9;    // Puerto digital 9 para la entrada 2 del L293D.
int pin7 = 10;   // Puerto digital 10 para la entrada 7 del L293D.
int pot  = A0;   // Puerto analógico A0 para el potenciómetro.
```

Declaramos variables para la lectura del potenciómetro y para generar señales PWM para el dispositivo:

```
int valor_Potenciometro;// Variable para leer el potenciómetro.
int pwm1;               // Variable para generar PWM 1.
int pwm2;               // Variable para generar PWM 2.
```

Dentro de la función principal `void setup()` configuramos los puertos 9 y 10 como salidas:

```
pinMode(pin2, OUTPUT);      // Configuración de puertos como salidas.
pinMode(pin7, OUTPUT);
```

Dentro de la función principal `void loop()` leemos los datos del potenciómetro, y los mapeamos para obtener las señales de PWM:

```
valor_Potenciometro=analogRead(pot); // Leemos el potenciómetro.
// Mapeamos los valores del potenciómetro para generar señales PWM.
pwm1 = map(valor_Potenciometro, 0, 1023, 0, 255); // PWM 1.
pwm2 = map(valor_Potenciometro, 0, 1023, 255, 0); // PWM 2.
```

Enviamos los valores de PWM a los puertos de salida 9 y 10 para el control del motor a través del dispositivo:

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

CÓDIGO DE PROGRAMACIÓN

```
/*****************************************************************
*      Práctica  6.1 - Control de velocidad de un motor DC       *
*                      con un potenciómetro.                     *
*****************************************************************
Este programa permite realizar el control de un motor de
corriente continua desde la posición de un potenciómetro.
*****************************************************************/
int pin2 = 9;        // Puerto digital 9 para la entrada 2 del L293D.
int pin7 = 10;       // Puerto digital 10 para la entrada 7 del L293D.
int pot  = A0;       // Puerto analógico A0 para el potenciómetro.

int valor_Potenciometro;      // Variable para leer el potenciómetro.
int pwm1;                     // Variable para generar PWM 1.
int pwm2;                     // Variable para generar PWM 2.

void setup(){
  pinMode(pin2, OUTPUT);              // Configuramos los puertos
como salidas.
  pinMode(pin7, OUTPUT);
}

void loop(){
  valor_Potenciometro=analogRead(pot); // Almacenamos el valor del
potenciómetro dentro de esta variable.

  pwm1 = map(valor_Potenciometro, 0, 1023, 0, 255); // PWM 1.
  pwm2 = map(valor_Potenciometro, 0, 1023, 255, 0); //El PWM 2 está
invertido respecto al PWM 1.

  // Enviamos los valores de PWM a las dos puertos de salida usando
analogWrite(puerto, valor)
  analogWrite(pin2,pwm1);
  analogWrite(pin7,pwm2);
}
```

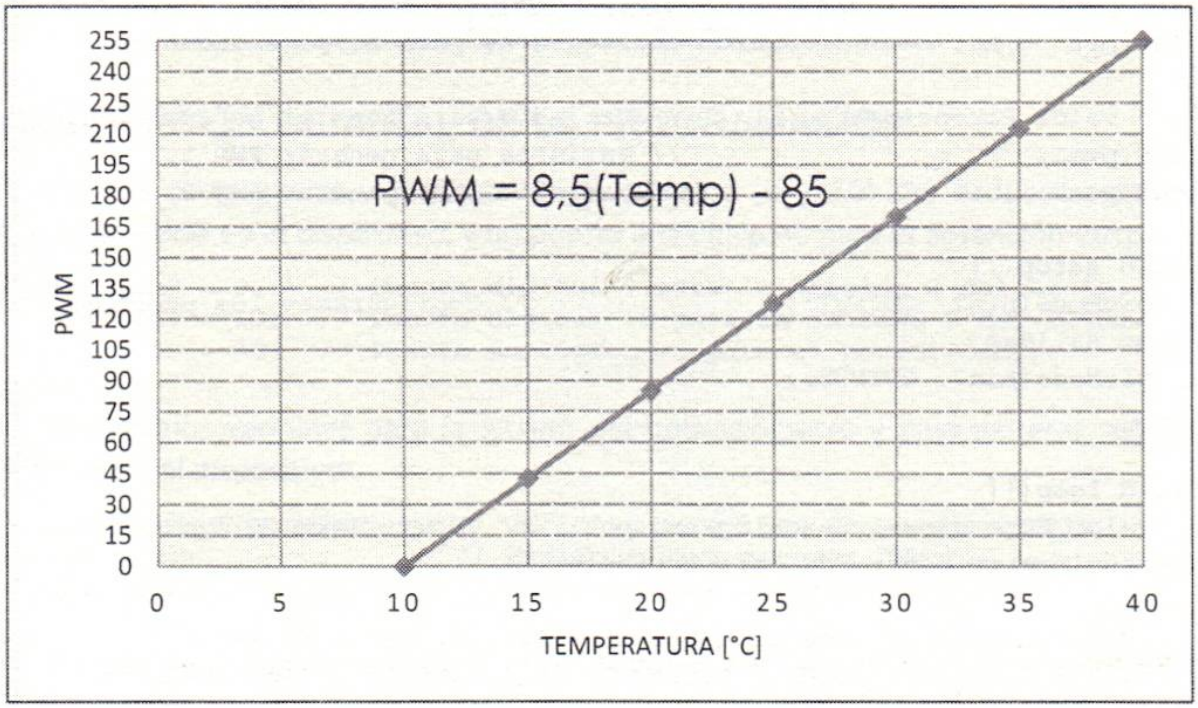
Cuando se **Verifique** y **Cargue** el código veremos el funcionamiento del motor DC en diferentes velocidades y ambos sentidos de giro.

Si giramos el potenciómetro a un punto medio de su valor resistivo aproximadamente, el motor no girará, pero a medida que variamos el valor resistivo del potenciómetro a uno de sus dos extremos, la velocidad del motor aumentará conforme el potenciómetro llegue al tope.

6.2 Control de velocidad de un motor DC por temperatura

En este ejemplo vamos a realizar el control de un motor DC por medio de la temperatura del sensor LM35. La particularidad de este ejemplo es que se puede aplicar para el control de la ventilación de una habitación a través de la temperatura ambiente.

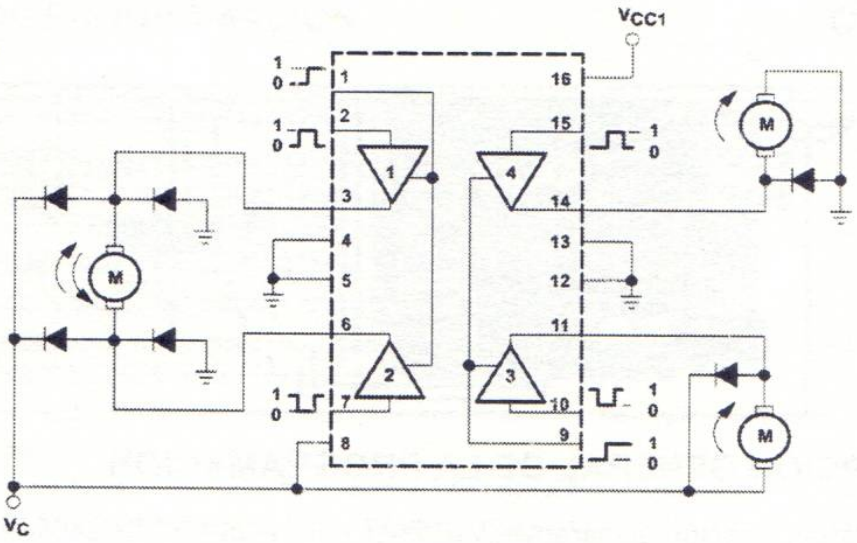
El motor gira en función de la temperatura medida, el rango permitido para el control del motor es de 10 a 40 °C. La velocidad del motor es lineal a la temperatura.



$$PWM = \begin{cases} 0 & Temp \leq 10°C \\ 8{,}5(Temp) - 85 & 10°C < Temp < 40°C \\ 255 & Temp \geq 40°C \end{cases}$$

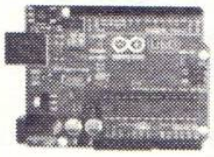
Es decir, para una temperatura de 10 °C la señal de PWM es 0, para bajas temperaturas el motor se apaga. Para una temperatura de 40 °C la señal de PWM es 255, para altas temperaturas el motor gira al máximo. Finalmente si las temperaturas son menores a 10 °C, el motor se mantiene apagado, y si las temperaturas son mayores a 40 °C, el motor se mantiene al máximo.

Dentro del rango de 10 y 40 °C el motor gira linealmente a la temperatura.

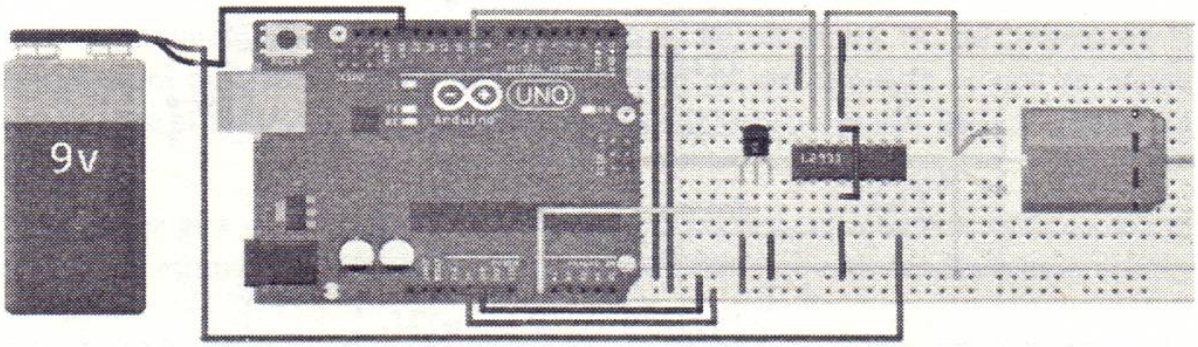


Para el control del motor realizaremos otra configuración en el dispositivo L293D, desde el puerto 14 del dispositivo conectaremos el motor, esto quiere decir que solo controlaremos el motor con una sola señal de PWM (puerto de entrada 15 en el dispositivo L293D).

MATERIALES

 Arduino UNO	 Protoboard	 Motor DC	 Batería de 9V	 Porta-baterías de 9V
 LM35	 L293D	 Cables		

CIRCUITO



DESCRIPCIÓN GENERAL DE LA PROGRAMACIÓN

Para la programación declaramos variables para el puerto de salida de la señal de PWM, puerto de entrada analógico para el sensor de temperatura:

```
int Pin2=9;    // Puerto digital 9 para la entrada 15 del L293D.
int Sensor=A0;// Puerto analógico para el sensor LM35.
```

Declaramos variables para el cálculo de temperatura, una variable para limitar el rango de temperatura medida, y para generar señales PWM:

```
int Temperatura;          // Variable para el cálculo de temperatura.
int Restricion_de_Temp;   // Variable para mapear la temperatura.
int pwm;                  // Variable para generar PWM.
```

Dentro de la función principal `void setup()` configuramos el puerto 9 como salida.

```
pinMode(Pin15,OUTPUT);  // Inicializar el puerto como salida.
```

Dentro de la función principal `void loop()` leemos la temperatura del sensor directamente aplicando la fórmula para Arduino:

```
Temperatura = (analogRead(Sensor))*0.4899;
```

Restringimos la temperatura para obtener un rango permitido de medición entre 10 y 40 °C a través de la función `constrain`:

```
Restricion_de_Temp = constrain(Temperatura, 10, 40);
```

Mapeamos el rango de temperatura (10 – 40 °C) a valores de (0 - 255) para generar señales de PWM para el control del motor:

```
pwm  = map(Restricion_de_Temp, 10, 40, 0, 255);
```

Finalmente escribimos en el puerto 9 del Arduino las señales de PWM para el control del motor en el puerto 15 del dispositivo:

```
analogWrite(Pin15,pwm);//Generar PWM en el puerto digital de salida.
```

CÓDIGO DE PROGRAMACIÓN

```
/*****************************************************************
*      Práctica  6.2 - Control de velocidad de un motor DC       *
*                      por temperatura                           *
*****************************************************************
Este programa permite realizar el control de un motor de
corriente continua en función de la temperatura.
- Si la temperatura es normal, es decir, entre 10°C y 40°C, el motor
     girará linealmente respecto a la temperatura.
- Si la temperatura es mayor a 40°C, el motor girará a velocidad
     máxima.
- Si la temperatura es menor a 10°C, el motor no girará.
*****************************************************************/
int Pin15=9;                // Puerto digital 9 para la entrada 15 del
L293D. Señal de PWM
int Sensor=A0;              // Puerto analógico para el sensor LM35.
int Temperatura;            // Variable para el cálculo de temperatura.
int Restricion_de_Temp;     // Variable para mapear la temperatura.
int pwm;                    // Variable para generar PWM.

void setup(){
  pinMode(Pin15,OUTPUT);  // Inicializamos  el puerto como salida.
}

void loop(){
  Temperatura = (analogRead(Sensor))*0.4899;     // Cálculo de
temperatura en °C T = x*5*100/1023.

  Restricion_de_Temp = constrain(Temperatura, 10, 40);  // Restricción
de la señal de temperatura para 10°C a 40°C
  pwm  = map(Restricion_de_Temp, 10, 40, 0, 255); // Mapeamos la
temperatura de 10°C - 40°C a 0 - 255

  analogWrite(Pin15,pwm);  // Generamos PWM en el puerto digital de
salida analogWrite(puerto, valor).
}
```

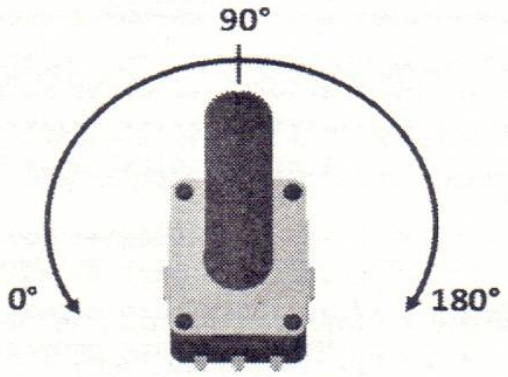
Una vez **Verificado** y **Cargado** el código a la tarjeta Arduino, el motor girará linealmente respecto a la temperatura dentro del rango permitido de 10 y 40 °C.

El motor no girará para temperaturas menores a 10 °C y girará al máximo para temperaturas mayores a 40 °C.

6.3 Control de un servomotor con un potenciómetro

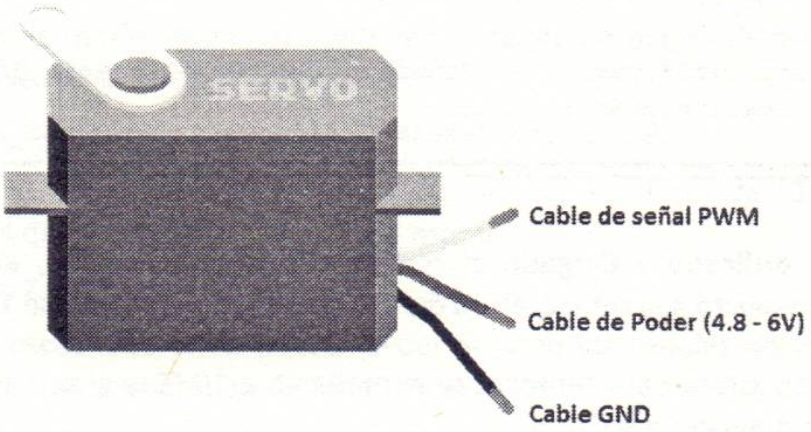
En este ejemplo vamos a realizar el control de un motor DC por medio de la temperatura del sensor LM35. La particularidad de este ejemplo es que se puede aplicar para el control de la ventilación de una habitación a través de la temperatura ambiente.

Este ejemplo permite controlar la posición angular de un servomotor a través de un potenciómetro.



Un servomotor es un dispositivo especial de motor DC que se caracteriza por su capacidad de posicionamiento angular de forma inmediata. El control de la posición angular del eje del dispositivo se hace a través de señales codificadas. Cuando la señal codificada cambia, la posición angular también.

Generalmente están formados por un amplificador, un motor, un sistema reductor de velocidad por ruedas dentadas y un circuito de realimentación. El control de un servomotor es a través de señales PWM para un margen de operación de 0° a 180°.

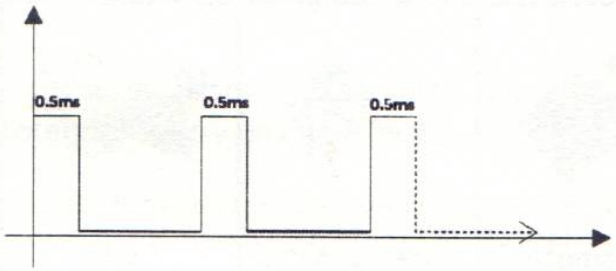


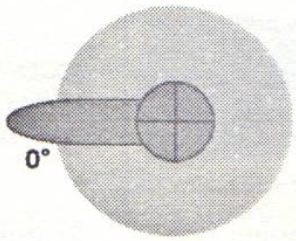
Los cables de un servomotor:

- Rojo - Alimentación (generalmente 4,8 a 6 voltios)
- Negro o Marrón - Masa o GND
- Amarillo, naranja o blanco - Señal de control (PWM)

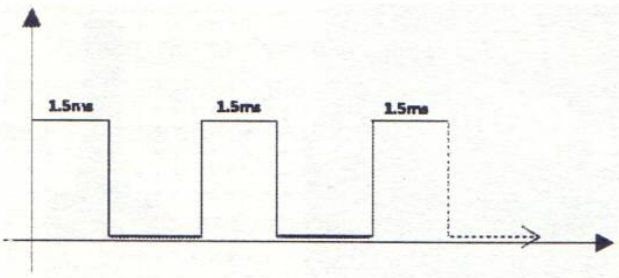
El funcionamiento de un servomotor se hace variando el ancho del pulso de la señal aplicada en milisegundos a una frecuencia constante de aproximadamente 50 Hz PWM (modulación por ancho de pulso). La duración del ancho del pulso determina el ángulo de giro del servomotor.

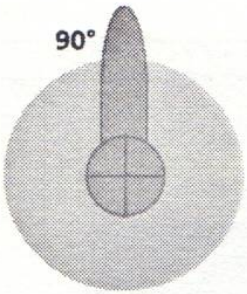
Para un giro de 0 grados, se modula la señal a un ancho de pulso con una duración de 0,5 milisegundos.



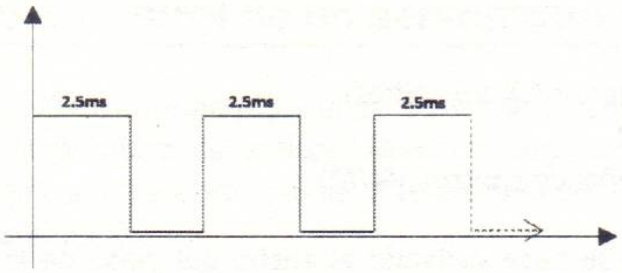


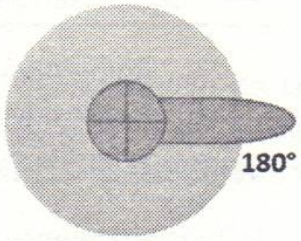
Para un giro de 90 grados, se modula una señal a un ancho de pulso con una duración de 1,5 milisegundos.





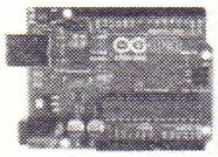
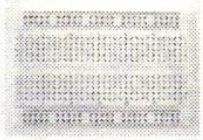
Para un giro de 180 grados, se modula una señal a un ancho de pulso con una duración de 2,5 milisegundos.



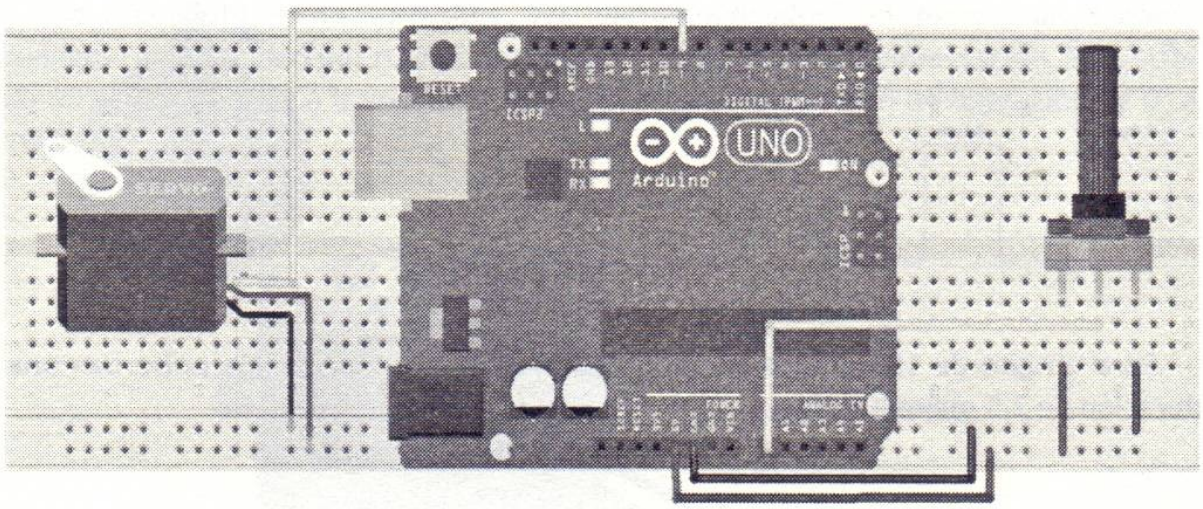


Para utilizar servomotores en Arduino se hace a través de una librería. Esta librería tiene la capacidad de soportar hasta 12 servomotores en la mayoría de las tarjetas Arduino y 48 servomotores en la tarjeta Arduino MEGA.

MATERIALES

				
Arduino UNO	Protoboard	Servomotor	Potenciómetro	Cables

CIRCUITO



DESCRIPCIÓN GENERAL DE LA PROGRAMACIÓN

Para la programación incluimos la librería `Servo.h` para cualquier tipo de servomotor, esta librería se encuentra en la barra Programa/incluir librería.

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

Para la funcionalidad del servomotor debemos crear un objeto de cualquier nombre:

```
Servo Mi_Servo;
```

Para el servomotor declaramos una variable entera para un puerto de salida de las señales PWM:

```
int Pin_Servo = 9;
```

Para el control declaramos una variable para el puerto analógico A0 del potenciómetro y una variable `long` para el almacenamiento de datos analógicos:

```
int Potenciometro = A0;// Puerto analógico A0 para el potenciómetro.
long Valor_analog = 0; /* Variable de almacenamiento de valores
                          analógicos. */
```

Declaramos una variable para definir el ángulo de giro del servomotor:

```
int Angulo = 0;
```

Dentro de la función principal `void setup()` posicionamos el servomotor al puerto digital de salida declarada anteriormente. En este puerto tendremos la señal de PWM para el servomotor:

```
Mi_Servo.attach(Pin_Servo);
```

Dentro de la función principal `void loop()` leemos los datos analógicos del potenciómetro. El bucle `for` permite realizar una suma de 100 valores analógicos con el propósito de eliminar un poco el ruido del potenciómetro:

```
for (int x = 0;x < 100; x++){
    Valor_analog += analogRead(Potenciometro);    /* Leemos los datos
                                                     analógicos. */
  };
```

Después realizamos la media aritmética para obtener un promedio del valor analógico del potenciómetro. Esto permite estabilizar un poco el servomotor.

```
Valor_analog = Valor_analog/100;
```

Mapeamos la señal analógica para convertirla en ángulo de giro para el servomotor, es decir, transformamos el rango analógico (0 - 1023) a (0° – 180°):

```
Angulo = map(Valor_analog, 0, 1023, 0, 180);
```

Finalmente escribimos en el puerto de salida el ángulo para el servomotor con un retardo para su correcto funcionamiento:

```
Mi_Servo.write(Angulo);  /* Escribir la posición del ángulo al
                            puerto del servomotor. */
```

La librería para servos permite controlar todo tipo de servomotores en ángulos de 0° a 180°.

CÓDIGO DE PROGRAMACIÓN

```
/*****************************************************************
*        Práctica  6.3 - Control de un servomotor con un         *
*                        potenciómetro.                          *
*****************************************************************
Este programa permite controlar la posición angular de un
servomotor a través de un potenciómetro.
*****************************************************************/

#include <Servo.h>      // Librería para servomotores.

Servo Mi_Servo;         // Creamos un objeto para el servomotor

int Pin_Servo = 9;      // Declaramos el puerto 9 para el servomotor.
int Potenciometro = A0;// Puerto analógico A0 para el potenciómetro
long Valor_analog = 0; // Variable de almacenamiento de valores
analógicos.
int Angulo = 0;         // Variable para guardar el ángulo de giro del
servo.

void setup(){
  Mi_Servo.attach(Pin_Servo); // Posicionar el servo al puerto 9.
}
```

```
void loop(){

  /* Realizamos una suma de 100 valores analógicos para eliminar
     un poco el ruido del potenciómetro. */
  for (int x = 0;x < 100; x++){
    Valor_analog += analogRead(Potenciometro); // Leemos los datos
analógicos.
  };

  Valor_analog = Valor_analog/100;     // Calculamos la  media
aritmética de la suma.
  Angulo = map(Valor_analog, 0, 1023, 0, 180); // Mapeamos los
valores analógicos a ángulos de giro.
  Mi_Servo.write(Angulo);  // Escribimos la posición del ángulo al
puerto del servo.
}
```

Después de **Verificar** y **Cargar** el código a la tarjeta Arduino, el servomotor girará a medida que se gira el potenciómetro a diferentes posiciones.

El servomotor gira a 180° cuando el valor de la resistencia es el máximo, 0° cuando el potenciómetro tiene una resistencia de 0 Ω, y 90° al valor medio resistivo del potenciómetro. El ángulo de giro del servomotor es lineal al voltaje de entrada del puerto analógico.

SÉPTIMA PRÁCTICA (COMUNICACIÓN SERIAL)

Para finalizar con esta última práctica vamos aprender a transmitir y recibir datos a través del puerto serial de la tarjeta Arduino y del monitor serial del entorno Arduino para comunicarnos y realizar múltiples tareas.

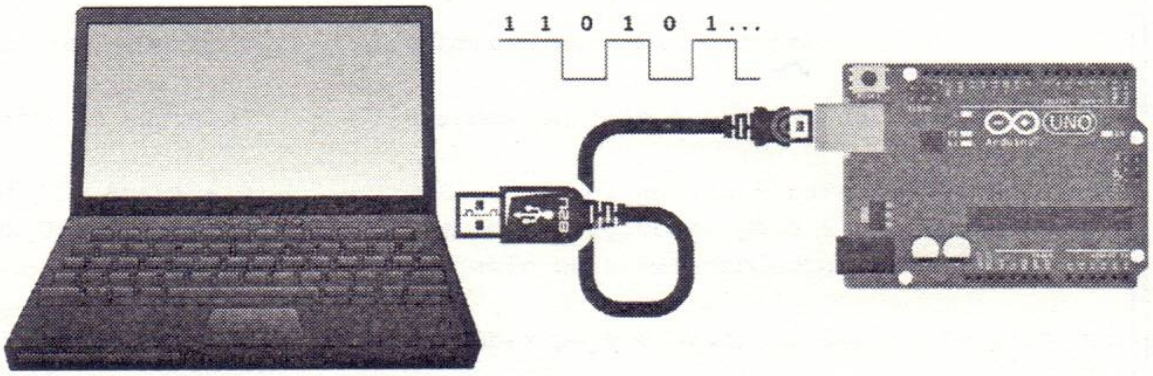
Este capítulo trata de cómo usar, procesar y mostrar cualquier tipo de información contenida en la tarjeta Arduino a través de las funciones de comunicación serial: `Serial.begin()`, `Serial.print()`, `Serial.read()`, `Serial.write()`, etc.

Para consultar todos los caracteres existentes, véase el Apéndice B. Tabla ASCII extendida.

7.1 Transmisión de datos desde el monitor serial de Arduino

En este primer ejemplo aprenderemos a usar el monitor serial de Arduino IDE para enviar y recibir información desde la tarjeta Arduino. Para la transmisión de datos usamos la tabla ASCII.

Para la transmisión de datos a través de un puerto serial, la información se envía bit a bit a una cierta velocidad, es decir, la información viaja en señales digitales de altos y bajos (HIGH y LOW), una técnica similar al encendido y apagado de un LED desde un punto de vista muy simple.

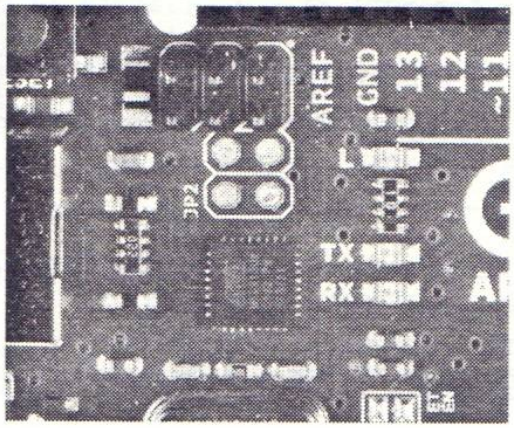


Arduino IDE permite enviar y recibir datos a través de su monitor serial por un puerto serial de comunicación. Para abrir un puerto serial, se debe fijar la velocidad en baudios (baudios/seg). El valor por defecto para comunicarse es de 9600 bps, aunque se pueden soportar otras velocidades: 4800, 14400, 19200, 28800, 38400, 57600 y 115200.

Cuando se usan los puertos de comunicación serial 0 (RX) y 1 (TX) no pueden utilizarse para otros propósitos.

Arduino MEGA tiene tres puertos seriales adicionales: Serial1 en los puertos 19 (RX) y 18 (TX), Serial2 en los puertos 17 (RX) y 16 (TX), y Serial3 en los puertos 15 (RX) y 14 (TX). El puerto de comunicación para el PC que se usa es el puerto Serial0.

Durante la comunicación serial, toda la capacidad de Arduino está a disposición del desarrollador. Cuando cargamos un sketch o cuando transmitimos datos, los LED TX y RX parpadean como señal de intercambio de información entre la tarjeta Arduino y un dispositivo. Cuando el LED RX parpadea, significa recepción de información y cuando el LED TX parpadea significa transmisión de información por el puerto serial de comunicación.



Para establecer comunicación con un dispositivo, Arduino no necesita incorporar una librería en especial, simplemente se usan funciones definidas en el entorno Arduino para la transmisión y recepción de datos, por ejemplo:

- **Serial.begin(9600)**. Abre el puerto serial y fija la velocidad en baudios para la transmisión de datos en serie. Por defecto es 9600.

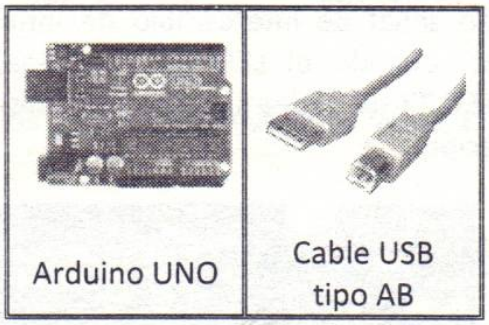
- **Serial.println(dato)**. Imprime los datos en el puerto serial, seguido por un retorno de carro y salto de línea.

- **Serial.println(dato, tipo de dato)**. Vuelca o envía un número o una cadena de caracteres al puerto serial, seguido por un retorno de carro y salto de línea.

- **Serial.print(dato, tipo de dato)**. Vuelca o envía un número o una cadena de caracteres al puerto serial.

- **Serial.available()**. Devuelve un entero con el número de bytes (caracteres) disponibles para leer desde el buffer serial.
- **Serial.read()**. Lee o captura un byte (carácter) desde el puerto serial.
- **Serial.write(dato)**. Escribe en el puerto serial byte o una cadena de bytes.

MATERIALES



Arduino UNO | Cable USB tipo AB

CIRCUITO

Para este ejemplo no es necesario armar ningún circuito en especial, solo se necesita la tarjeta Arduino y un cable UBS tipo AB.

DESCRIPCIÓN GENERAL DE LA PROGRAMACIÓN

Para la programación declaramos una variable tipo entero para almacenar los datos de llegada:

```
int Entrada_de_datos = 0;
```

Dentro de la función principal `void setup()` iniciamos el puerto serial para la comunicación a una velocidad de 9600 bps, e imprimimos mensajes de texto en el puerto serial, es decir, transmitimos cualquier información:

```
Serial.begin(9600);              /* Iniciamos el puerto serial con una
                                 velocidad de 9600 baudios/segundo. */
  delay(1000);
// Transmitimos texto con saltos de línea "println".
Serial.println("                        HOLA MUNDO!                          ");
Serial.println("|_____________________________________________________________|");
Serial.println("Este ejemplo envia y recibe datos del puerto serial");
Serial.println("|_____________________'-'_____________________|");
```

Dentro de la función principal `void loop()` leemos si en el buffer existen datos de llegada, si existe algún dato en el buffer, este es mayor a 0 bytes `Serial.available()>0`. Después leemos los datos, o capturamos esa información del puerto serial `Entrada_de_datos = Serial.read()`. Imprimimos textos y reenviamos los datos de llegada para nuevamente mostrarlos en el monitor serial de Arduino IDE en formatos diferentes (decimal, hexadecimal y binario):

```
if (Serial.available()>0){
  Entrada_de_datos = Serial.read();     // Leer o capturar los datos
(bytes) desde el puerto serial.
  Serial.print("El caracter enviado es: ");  // Imprimir un mensaje
sin salto de línea.
  Serial.write(Entrada_de_datos);       // Escribir en el puerto serial
el carácter de llegada.
  Serial.println();                     // Imprimir un salto de línea.
  Serial.print("Valor decimal:");       // Mensaje.
  Serial.print(Entrada_de_datos);       // Imprimir el valor decimal.
  Serial.print(", Valor Hexadacimal:");// Mensaje.
  Serial.print(Entrada_de_datos,HEX);   // Imprimir el valor
hexadecimal.
  Serial.print(", Valor Binario:");     // Mensaje.
  Serial.println(Entrada_de_datos,BIN);// Imprimir valor binario.
}
```

CÓDIGO DE PROGRAMACIÓN

```
/*****************************************************************
*         Práctica 7.1 - Transmisión de datos desde el           *
*                    monitor serial de Arduino                   *
 *****************************************************************
 Este programa permite enviar y recibir datos a través de un
 puerto serial.
 *****************************************************************/

int Entrada_de_datos = 0;  // Variable para el almacenamiento de
datos de llegada.

void setup(){
  Serial.begin(9600);      // Iniciamos el puerto serial con una
velocidad de 9600 baudios/segundo.
  delay(1000);
  // Imprimimos en el puerto serial caracteres, con un salto de línea
"println".
Serial.println("                    HOLA MUNDO!
");
Serial.println("|_____________________________________________|")
;
```

```
Serial.println("Este ejemplo envia y recibe datos del puerto
serial");
Serial.println("|_____________________'-
'_____________________|");
}

void loop(){
  /* Devuelve los datos (número de bytes) disponibles para leer desde
el buffer serie, o si hay alguno.
     Si hay algún dato disponible, Serial.available() será mayor que
0.
     El buffer serie puede almacenar como máximo 128 bytes. */
  if (Serial.available()>0){
    Entrada_de_datos = Serial.read();     // Leemos o capturamos los
datos (bytes) desde el puerto serial.
    Serial.print("El caracter enviado es: ");  // Imprimo un mensaje
sin salto de línea.
    Serial.write(Entrada_de_datos);       // Escribimos en el puerto
serial el carácter de llegada.
    Serial.println();                     // Imprimimos un salto de
línea.
    Serial.print("Valor decimal:");       // Mensaje.
    Serial.print(Entrada_de_datos);       // Imprimimos el valor
decimal.
    Serial.print(", Valor Hexadacimal:");// Mensaje.
    Serial.print(Entrada_de_datos,HEX);  // Imprimimos el valor
hexadecimal.
    Serial.print(", Valor Binario:");     // Mensaje.
    Serial.println(Entrada_de_datos,BIN);// Imprimimos valor binario.
  }
}
```

Después de **Verificar** y **Cargar** el código a la tarjeta Arduino, primero debemos abrir el monitor serial de Arduino IDE y asegurarnos de seleccionar la velocidad a 9600 bps. Si es así, recibiremos el siguiente mensaje:

```
                    HOLA MUNDO!
|_____________________________________________________|
Este ejemplo envía y recibe datos del puerto serial
|_____________________'-'_____________________|
```

Después enviaremos cualquier carácter para recibir información según la tabla ASCII extendida del apéndice B, por ejemplo:

```
El carácter enviado es: ñ
Valor decimal:241, Valor Hexadecimal:F1, Valor Binario: 11110001
```

7.2 Transmisión de temperaturas

Este programa permite leer y transmitir temperaturas del sensor LM35 por un puerto serial en grados Celsius, grados Fahrenheit y grados Kelvin.

Temperatura es el nivel de calor de un gas, líquido o sólido. Tres escalas sirven comúnmente para medir la temperatura: las escalas de Celsius, Fahrenheit y Kelvin. La escala Kelvin se usa pricipalmente en experimentos científicos.

La escala Celsius es conocida como escala centígrados (°C), esta escala divide el rango entre las temperaturas de congelación y ebullición del agua en 100 partes iguales.

La escala Fahrenheit (°F) divide la diferencia entre los puntos de fusión y de ebullición del agua en 180 intervalos iguales. Su fórmula es la siguiente:

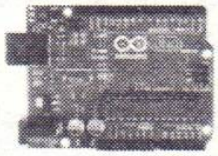
$$°F = (°C * 1{,}8) + 32$$

La escala Kelvin (°K) prolonga la escala Celsius hasta el cero absoluto, una temperatura hipotética caracterizada por una ausencia completa de energía calórica. Su fórmula es:

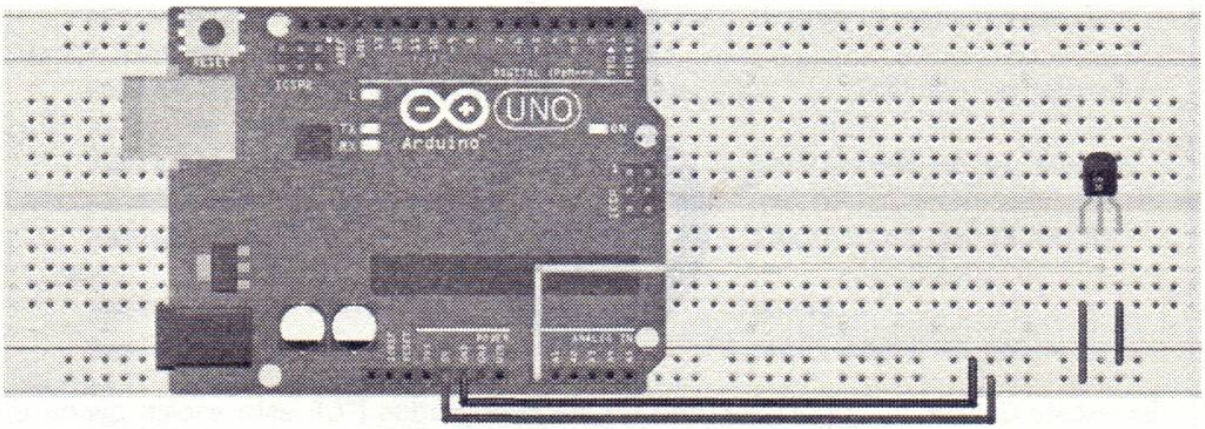
$$°K = °C + 273{,}15$$

Para la transmisión de datos se realizará a una velocidad distinta a la anterior 19200 bps.

MATERIALES

			
Arduino UNO	Protoboard	LM35	Cables

CIRCUITO



DESCRIPCIÓN GENERAL DE LA PROGRAMACIÓN

Para la programación declaramos una variable tipo `int` para el puerto analógico de entrada A0, una variable `int` para guardar sus datos analógicos, y una variable `float` para el cálculo de temperatura:

```
int Sensor_LM35 = A0;      /* Declarar el puerto analógico A0 para el
                                                    sensor LM35. */
int Valor_Analogico = 0;   /* Declarar una variable para guardar los
                                                    datos del sensor. */
float Temperatura = 0.0;   /* Declarar una variable para el cálculo
                                                    de temperatura. */
```

Declaramos variables tipo flotante para el cálculo de las diferentes escalas:

```
float Grados_Cent = 0.0;   // Variables para escala Celsius
float Grados_Fahr = 0.0;   // Variables para escala Fahrenheit
float Grados_Kelv = 0.0;   // Variables para escala Kelvin.
```

Dentro de la función principal `void setup()` iniciamos el puerto serial para la comunicación a una velocidad de 19200 bps, e imprimimos mensajes de texto en el puerto serial:

```
Serial.begin(19200);    /* Iniciar la comunicación a una velocidad de
                                                        19200 bps. */
delay(1000);            // Retardo
  // Imprimimos texto:
Serial.println("Transmision de Temperaturas en tres escalas:");
Serial.println("     Escala Celsius, Fahrenheit y Kelvin    ");
```

Dentro de la función principal `void loop()` leemos los datos del puerto analógico y calculamos la temperatura según la fórmula del sensor LM35 para Arduino:

```
Valor_Analogico= analogRead(Sensor_LM35);  // Leer datos del sensor.
Temperatura = (Valor_Analogico * 5) / (1023 * 0.01); /* Calcular
                        temperatura según la fórmula establecida. */
```

Realizamos el cálculo para las diferentes escalas según cada fórmula:

```
Grados_Cent = Temperatura;              // Grados centígrados.
Grados_Fahr = (Grados_Cent * 1.8) + 32;// Grados Fahrenheit.
Grados_Kelv = Grados_Cent + 273.15;     // Grados Kelvin.
```

Finalmente imprimimos las temperaturas de las tres escalas con tres valores decimales cada uno en el puerto serial:

```
// Imprimimos las temperaturas en las tres escalas.
Serial.print("Temperaturas: ");
Serial.print(Grados_Cent,3);
Serial.print("'C, ");
Serial.print(Grados_Fahr,3);
Serial.print("'F, ");
Serial.print(Grados_Kelv,3);
Serial.println("'K");
delay(1000); // Retardo para transmitir cada segundo.
```

CÓDIGO DE PROGRAMACIÓN

```
/*****************************************************************
*          Práctica 7.2 - Transmisión de temperaturas            *
*****************************************************************
Este programa permite leer la temperatura de un sensor de
temperatura LM35 y enviar por puerto serial temperaturas en grados
Celsius, grados Fahrenheit y grados Kelvin.
*****************************************************************/
```

```
int Sensor_LM35 = A0;      // Declaramos el puerto analógico A0 para
el sensor LM35.
int Valor_Analogico = 0;  // Declaramos una variable para guardar los
datos del sensor.
float Temperatura = 0.0;  // Declaramos una variable para el cálculo
de temperatura.

// Variables para la conversión de temperaturas en las tres escalas.
float Grados_Cent = 0.0;
float Grados_Fahr = 0.0;
float Grados_Kelv = 0.0;

void setup(){
  Serial.begin(19200);  // Iniciamos el puerto de comunicación a una
velocidad de 19200 bps.
  delay(1000); // Retardo
  // Imprimimos texto:
  Serial.println("Transmision de Temperaturas en tres escalas:");
  Serial.println("     Escala Celsius, Fahrenheit y Kelvin    ");
}

void loop(){
    Valor_Analogico= analogRead(Sensor_LM35);    // Leemos los datos
del sensor.
    Temperatura = (Valor_Analogico * 5) / (1023 * 0.01); //
Calculamos temperatura según la fórmula establecida.

    Grados_Cent = Temperatura;              // Grados centígrados.
    Grados_Fahr = (Grados_Cent * 1.8) + 32;// Grados Fahrenheit.
    Grados_Kelv = Grados_Cent + 273.15;     // Grados Kelvin.

    // Imprimimos las temperaturas en las tres escalas.
    Serial.print("Temperaturas: ");
    Serial.print(Grados_Cent,3);
    Serial.print("'C, ");
    Serial.print(Grados_Fahr,3);
    Serial.print("'F, ");
    Serial.print(Grados_Kelv,3);
    Serial.println("'K");
    delay(1000); // Retardo para transmitir cada segundo.
}
```

Después de **Verificar** y **Cargar** el código a la tarjeta Arduino, primero debemos abrir el monitor serial de Arduino IDE y asegurarnos de seleccionar la velocidad a 19200 bps. Si es así, recibiremos el siguiente mensaje:

```
Transmisión de Temperaturas en tres escalas:
     Escala Celsius, Fahrenheit y Kelvin
```

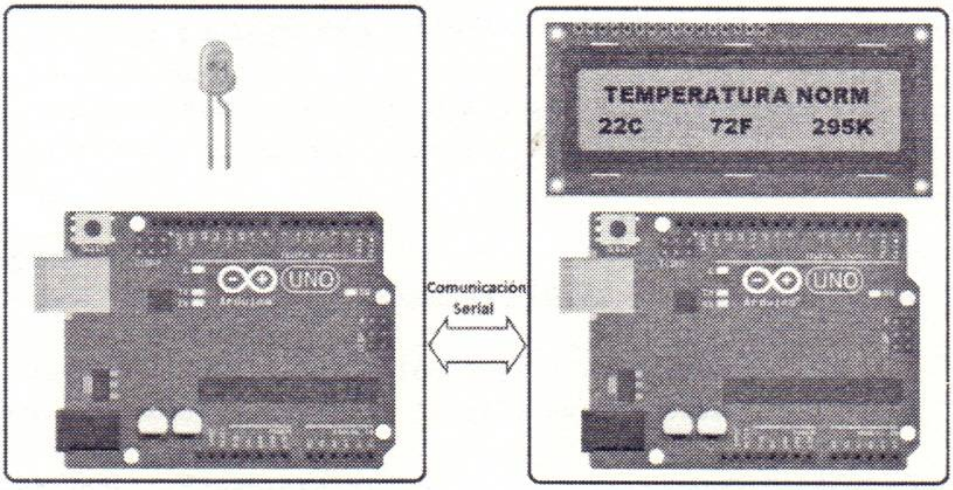
Después recibiremos las temperaturas en las tres escalas, Celsius, Fahrenheit y Kelvin cada segundo:

```
Temperaturas: 21.017'C, 69.830'F, 294.167'K
Temperaturas: 20.528'C, 68.950'F, 293.678'K
Temperaturas: 21.017'C, 69.830'F, 294.167'K
Temperaturas: 21.017'C, 69.830'F, 294.167'K
Temperaturas: 20.528'C, 68.950'F, 293.678'K
Temperaturas: 21.017'C, 69.830'F, 294.167'K

...
```

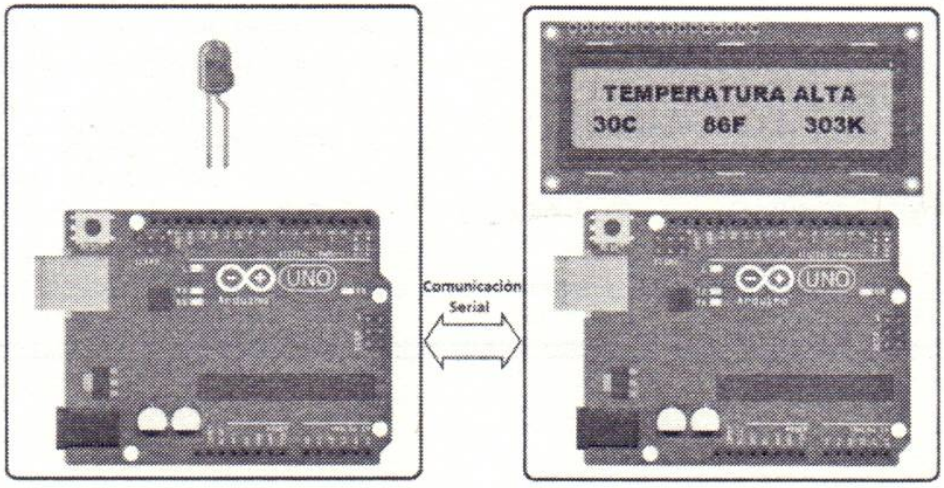
7.3 Transmisión serial de datos entre dos tarjetas Arduino

Para culminar esta práctica, realizaremos una transmisión serial de datos entre dos tarjetas Arduino. La tarjeta 1 realizará el cálculo de temperatura en las tres escalas (Celsius, Fahrenheit y Kelvin) y la transmitirá por el puerto serial, y el encendido de LED para temperaturas normales, bajas y altas. En la tarjeta 2 se presentarán las temperaturas en las tres escalas en una pantalla LCD con un comentario correspondiente a las temperaturas, y transmitir alertas para el encendido de LED. El rango de temperatura normal es de 20 a 25 °C.

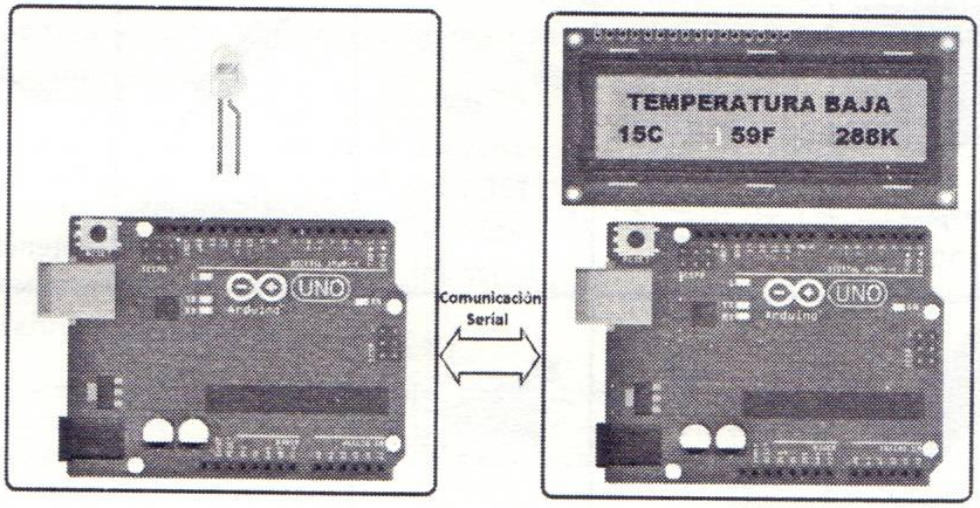
Si la temperatura es normal, en el Arduino 1 se encenderá un LED verde, y en el Arduino 2 se imprimirá en la pantalla LCD un mensaje "Temperatura Normal".



Si la temperatura es alta, en el Arduino 1 se encenderá un LED rojo, y en el Arduino 2 se imprimirá en la pantalla LCD un mensaje "Temperatura Alta".

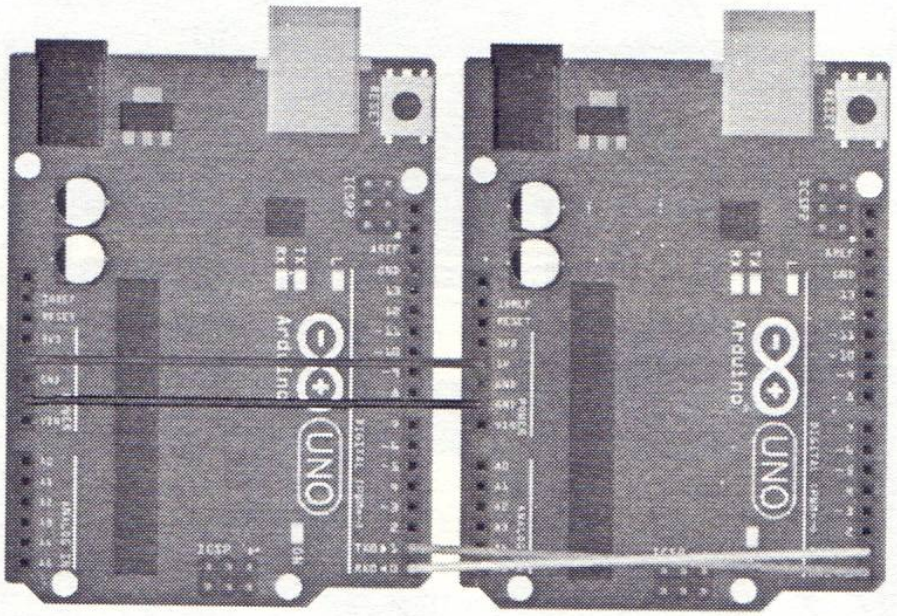


Si la temperatura es baja, en el Arduino 2 se encenderá un LED amarillo, y en el Arduino 2 se imprimirá en la pantalla LCD un mensaje "Temperatura Baja".

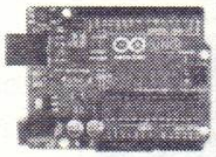


La transmisión de datos será cada medio segundo (500 ms) a la misma velocidad de trasmisión (115200 bps) para ambas tarjetas.

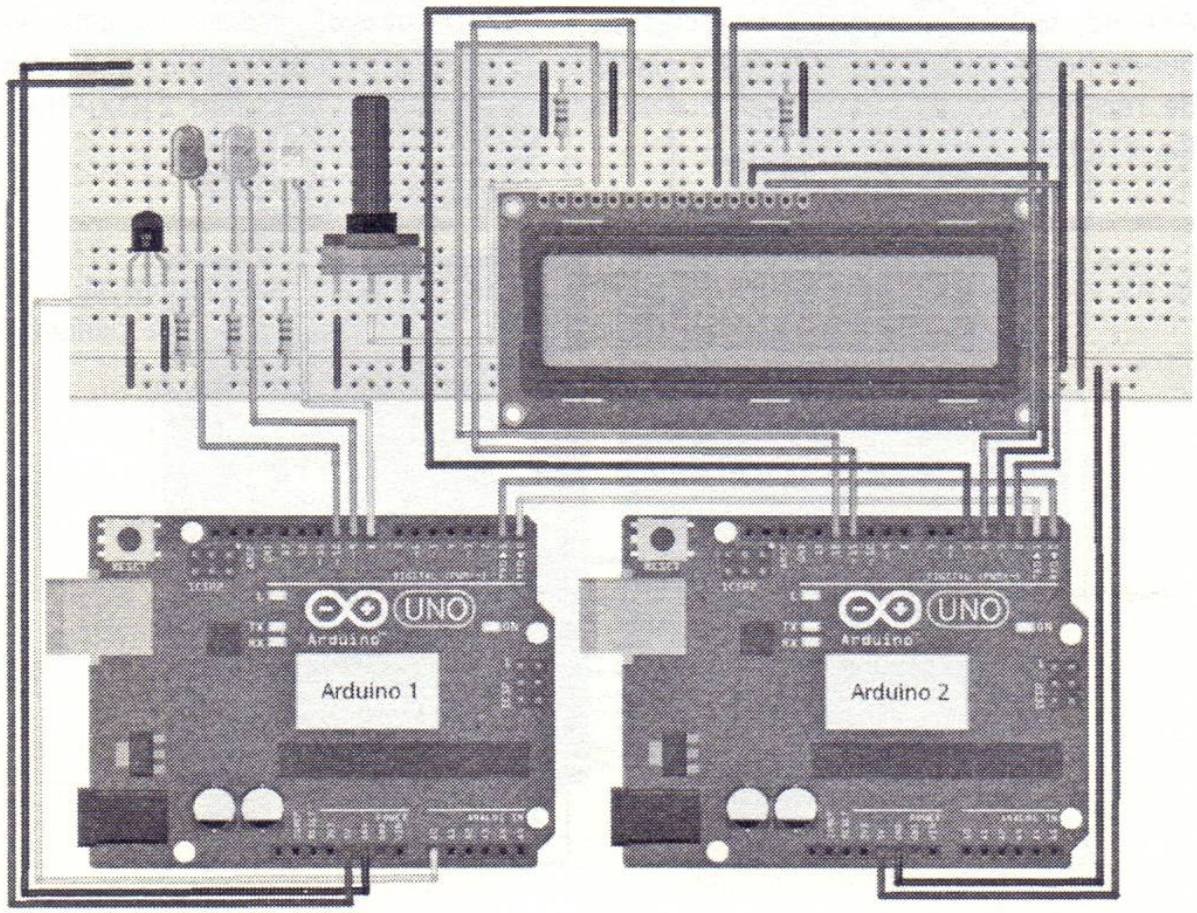
Para la transmisión de datos a través de un puerto serial entre dos tarjetas Arduino, es muy importante conectar los puertos cruzados, es decir, conectar el puerto de transmisión de la tarjeta 1 al puerto de recepción de la tarjeta 2 y el puerto de recepción de la tarjeta 1 al puerto de transmisión de la tarjeta 2 ($Tx_1 \rightarrow Rx_2$, $Rx_1 \rightarrow Tx_2$).



MATERIALES

 2 tarjetas Arduino UNO	 Protoboard	 1 LED (rojo, verde y amarillo)	 5 Resistencias de 330 Ω	 1 Potenciómetro
 Pantalla LCD 16x2	 LM35	 Cables		

CIRCUITO



DESCRIPCIÓN GENERAL DE LA PROGRAMACIÓN (TARJETA 1)

Para la programación en la tarjeta Arduino 1 declaramos una variable tipo entero para el puerto analógico A0, almacenamiento de datos y una variable flotante para el cálculo de temperatura:

```
int Sensor = A0;     //Puerto analógico A0 para el sensor LM35.
int ValorSensor = 0;//Variable de almacenamiento de datos del sensor
float Temperatura = 0;//Variable para el cálculo de temperatura.
```

Declaramos variables enteras para el cálculo de temperatura en las tres escalas (Celsius, Fahrenheit y Kelvin):

```
int Grados_Cent;     // Variable para grados centígrados.
int Grados_Fahr;     // Variable para grados Fahrenheit.
int Grados_Kelv;     // Variable para grados Kelvin.
```

Declaramos variables para los puertos digitales de salida correspondientes a los LED (rojo, verde, amarillo):

```
int Rojo = 10;          // Puerto digital 6 para LED rojo.
int Verde = 9;          // Puerto  digital 7 para LED verde.
int Amarillo = 8;       // Puerto  digital 8 para LED amarillo.
```

Declaramos una variable para los tiempos de transmisión de datos `unsigned long Tiempo_Tx`.

Dentro de la función principal `void setup()` iniciamos el puerto serial para la comunicación a un velocidad de 115200 bps. Configuramos los puertos de salida para los LED e inicializados en estado lógico 0 (`LOW`) para mantenerlos apagados en el momento de encender la tarjeta.

```
Serial.begin(115200);    // Velocidad de transmisión 115200 bps.
delay(1000);             // Retardo
// Configuración de puertos digitales 8, 9 y 10 como salidas.
pinMode(Rojo,OUTPUT);
pinMode(Verde,OUTPUT);
pinMode(Amarillo,OUTPUT);
// Inicialización de los puertos 8, 9 y 10 como nivel lógico 0.
digitalWrite(Rojo, LOW);
digitalWrite(Verde, LOW);
digitalWrite(Amarillo, LOW);
Tiempo_Tx = millis();    // Leer tiempo de inicio de programa.
```

Dentro de la función principal `void loop()` leemos los datos del sensor LM35 para el cálculo de temperatura según la fórmula para Arduino y el cálculo para las tres escalas (Celsius, Fahrenheit y Kelvin):

```
ValorSensor = analogRead(Sensor);    // Lectura del sensor LM35.
Temperatura = ValorSensor * 0.4899; // Cálculo de temperatura.
Grados_Cent = (int)Temperatura;      // Grados Celsius o centígrados.
Grados_Fahr = (int)(Grados_Cent * 1.8) + 32;// Grados Fahrenheit.
Grados_Kelv = (int)(Grados_Cent + 273.15);  // Grados Kelvin.
```

Después transmitimos los datos calculados en las tres escalas de temperatura con sus unidades correspondientes (C, F y K) por cada medio segundo (500 ms) a través de la función de tiempo `millis()`. La transmisión de datos se envía a través de la función **`Serial`**`.print()`.

```
if ((millis()-Tiempo_Tx) > 500){ // Transmisión en medio segundo.
  Serial.print(Grados_Cent);    // Transmisión de grados centígrados.
  Serial.print("C");              // Enviamos la unidad.
  Serial.print(Grados_Fahr);     // Transmisión en Fahrenheit.
  Serial.print("F");              // Enviamos la unidad.
  Serial.print(Grados_Kelv);     // Transmisión en Kelvin.
  Serial.print("K");              // Enviamos la unidad.
  Tiempo_Tx = millis();          // Tiempo de actualización.
}
```

Finalmente para el encendido de los LED para las temperaturas bajas, normales y altas, iniciamos el proceso de recepción de datos del puerto serial **`Serial`**`.available()>0`. Este proceso es controlado por la tarjeta 2, ya que ordena a la tarjeta 1 cuándo las temperaturas se encuentran fuera del rango permitido (20 – 25 °C) para encender los LED correspondientes. Recibe el carácter `'a'` para temperaturas altas, el carácter `'n'` para temperaturas normales y el carácter `'b'` para temperaturas bajas. Los datos son leídos a través de la función **`Serial`**`.read()`.

```
if (Serial.available()>0){
  int datos = Serial.read();       // Lectura de datos.
  if(datos == 'a'){                // Temperatura alta.
    digitalWrite(Rojo, HIGH);      // Encendido del LED rojo.
    digitalWrite(Verde, LOW);
    digitalWrite(Amarillo, LOW);
  }
  if(datos == 'b'){                // Temperatura baja.
    digitalWrite(Rojo, LOW);
    digitalWrite(Verde, LOW);
    digitalWrite(Amarillo, HIGH); // Encendido del LED amarillo.
  }
  if(datos == 'n'){                // Temperatura normal.
    digitalWrite(Rojo, LOW);
    digitalWrite(Verde, HIGH);     // Encendido del LED verde.
    digitalWrite(Amarillo, LOW);
  }
}
```

CÓDIGO DE PROGRAMACIÓN (TARJETA 1)

```
/*****************************************************************
*     Práctica 7.3 - Transmisión serial de datos entre dos      *
*                        tarjetas Arduino                       *
*****************************************************************
Este ejemplo permite realizar una transmisión serial de datos entre
dos tarjetas Arduino.
 - La tarjeta 1 realizará el cálculo de temperatura en las tres
   escalas (Celsius, Fahrenheit, Kelvin) y el encendido de LED
   para temperaturas normales, bajas y altas.
 - En la tarjeta 2 se presentarán las temperaturas en las tres
   escalas en una pantalla LCD. EL rango de temperatura normal es
   de 20 a 25 °C.
*****************************************************************/

int Sensor = A0;                        // Asignamos el puerto
analógico A0 para el sensor LM35.
int ValorSensor = 0;                    // Variable para el
almacenamiento del valor del sensor.
float Temperatura = 0;                  // Variable para almacenar el
valor de la temperatura.

int Grados_Cent;                        // Variable para grados
centígrados.
int Grados_Fahr;                        // Variable para grados
Fahrenheit.
int Grados_Kelv;                        // Variable para grados
Kelvin.

int Rojo = 10;                          // Asignación del puerto
digital 6 para el color rojo.
int Verde = 9;                          // Asignación del puerto
digital 7 para el color verde.
int Amarillo = 8;                       // Asignación del puerto
digital 8 para el color amarillo.

unsigned long Tiempo_Tx;

void setup() {
  Serial.begin(115200);                 // Iniciamos el puerto de
comunicación a una velocidad de 115200 bps.
  delay(1000);                          // Retardo
  // Configuramos los puertos digitales 8, 9 y 10 como salidas.
  pinMode(Rojo,OUTPUT);
```

```
  pinMode(Verde,OUTPUT);
  pinMode(Amarillo,OUTPUT);
  // Inicializamos los pines 6, 7 y 8 como nivel lógico 0.
  digitalWrite(Rojo, LOW);
  digitalWrite(Verde, LOW);
  digitalWrite(Amarillo, LOW);

  Tiempo_Tx = millis();                  // Leemos el tiempo de inicio
de programa.
}

void loop() {
  ValorSensor = analogRead(Sensor);     // Leemos el valor del sensor.
  Temperatura = ValorSensor * 0.4899;  // (Multiplicamos por una
constante de 0.4899 = ((5/1023)/0.01) para obtener el valor del
sensor en voltios.

  Grados_Cent = (int)Temperatura;              // Grados Celsius o
centígrados.
  Grados_Fahr = (int)(Grados_Cent * 1.8) + 32;// Grados Fahrenheit.
  Grados_Kelv = (int)(Grados_Cent + 273.15);  // Grados Kelvin.

  // Transmisión de datos
  if ((millis()-Tiempo_Tx) > 500){     // Transmisión en cada segundo.
1seg = 1000 miliseg.
    Serial.print(Grados_Cent);          // Transmisión en grados
centígrados
    Serial.print("C");                  // Enviamos la unidad.
    Serial.print(Grados_Fahr);          // Transmisión en Fahrenheit.
    Serial.print("F");                  // Enviamos la unidad.
    Serial.print(Grados_Kelv);          // Transmisión en Kelvin.
    Serial.print("K");                  // Enviamos la unidad.
    Tiempo_Tx = millis();               // Tiempo de actualización.
  }

  // Recepción de datos.
  if (Serial.available()>0){
    int datos = Serial.read();          // Lectura de datos.
    if(datos == 'a'){                   // Temperatura alta.
      digitalWrite(Rojo, HIGH);         // Encendemos LED para
temperaturas altas.
      digitalWrite(Verde, LOW);
      digitalWrite(Amarillo, LOW);
    }
    if(datos == 'b'){                   // Temperatura baja.
```

```
      digitalWrite(Rojo, LOW);
      digitalWrite(Verde, LOW);
      digitalWrite(Amarillo, HIGH);     // Encendemos LED para
temperaturas bajas.
    }
    if(datos == 'n'){                   // Temperatura normal.
      digitalWrite(Rojo, LOW);
      digitalWrite(Verde, HIGH);        // Encendemos LED para
temperaturas normales.
      digitalWrite(Amarillo, LOW);
    }
  }
}
```

DESCRIPCIÓN GENERAL DE LA PROGRAMACIÓN (TARJETA 2)

Para la programación en la tarjeta Arduino 2 incluiremos la librería `LiquidCrystal` para la pantalla LCD e inicializamos la librería con los puertos 2, 3, 4, 5, 11, y 12 para la comunicación entre la pantalla y la tarjeta Arduino.

```
#include <LiquidCrystal.h>                  // Librería LCD.
LiquidCrystal lcd(12, 11, 5, 4, 3, 2); // Configuración de puertos.
```

Para la recepción de datos declaramos una variable array tipo char de tamaño 13. Esta variable permite guardar los datos de llegada del puerto serial. Cada dato en el puerto serial es leído byte a byte, por lo que es necesario guardarlo en cada posición del array, y para eso es necesario declarar otra variable entera para el almacenamiento por posición. Los datos de llegada entran en una trama de la siguiente forma: **21C69F294K**, donde 21C es la temperatura en Celsius, 69F es la temperatura en Fahrenheit, y 294K es la temperatura en Kelvin.

```
char datos[13];     // Variable array para capturar datos de llegada.
int posicion = 0;   // Variable de posición para datos de llegada.
```

Declaramos una variable tipo char para la segmentación de datos de llegada. Declaramos una variable array tipo char para separar estos datos de llegada, esta variable separadora identifica cada variable antes de los caracteres (C, F y K). Declaramos una variable array tipo entera para el almacenamiento de cada uno de los datos separados, es decir, cada escala de temperatura por separado.

```
char *resultado = NULL;      // Variable de segmentación de datos.
char separador[] = "CFK";    // Variable separadora de datos.
int valores[] = {0,0,0};     /* Variable para almacenar los datos
                        segmentados {Celsius, Fahrenheit, Kelvin}. */
```

Declaramos variables enteras para asignar las escalas correspondientes (Celsius, Fahrenheit y Kelvin) después de la segmentación de datos:

```
int Grados_Cent;     // Variable para grados centígrados.
int Grados_Fahr;     // Variable para grados Fahrenheit.
int Grados_Kelv;     // Variable para grados Kelvin.
```

Declaramos variables para los tiempos de transmisión de datos e impresión de datos en la pantalla LCD en cada medio segundo (500ms):

```
unsigned long Tiempo_LCD;                    /* Variable de tiempo para
                                                impresión de datos. */
unsigned long Tiempo_Tx;                     /* Variable de tiempo para
                                              transmisión de datos. */
```

Dentro de la función principal `void setup()` iniciamos la pantalla LCD 16x2, iniciamos el puerto serial para la comunicación a una velocidad de 115200 bps, y finalmente leemos tiempos después del encendido de la tarjeta Arduino:

```
lcd.begin(16, 2);       // Inicio de pantalla LCD 16x2.
Serial.begin(115200);   // Velocidad de transmisión 115200 bps.
delay(1000);            // Retardo.
Tiempo_LCD = millis(); // Leer tiempo de inicio para pantalla LCD.
Tiempo_Tx = millis();  // Leer tiempo de inicio para Tx.
```

Dentro de la función principal `void loop()` declaramos una variable local `boolean segmentación` para preparar los datos a segmentar, pero antes de eso leemos los datos de llegada en el puerto serial y los almacenamos por posición `datos[posicion]`. Durante este proceso aprobamos la segmentación de datos después de terminar de leer todos los datos de llegada:

```
while(Serial.available()>0){      /* Mientras haya datos en el buffer
                                     ejecuta la función. */
  delay(5);                       // Retardo para recepción de datos.
  datos[posicion] = Serial.read();    /* Leer un carácter del string
     "datos" de la "posición", luego leer el siguiente carácter con
                                                  "posición++". */
  posicion++;                     /* Aumentamos la variable en 1 para
                        ir leyendo los datos de llegada uno a uno. */
  segmentacion = true;            /* Después de la recepción de
     datos, aprobamos (true) para entrar a segmentación de datos. */
}
```

Es importante encerrar la variable posicion después de terminar de leer todos los datos del puerto serial para preparar la variable `datos[]` durante las próximas recepciones de datos:

```
posicion = 0; /* Encerramos el contador de posición para
        empezar de nuevo en la próxima recepción de datos. */
```

Ahora estamos listos para segmentar los datos. Primero separamos los datos correspondientes de las tres escalas de temperatura a través de un identificador mediante la función `strtok`, después colocamos cada resultado segmentado dentro del vector `valores[index++]` por posición convertidos directamente a datos enteros a través de la función `atoi`. Finalmente anulamos todos los resultados hasta las próximas segmentaciones de nuevos datos. Es importante desactivar la segmentación hasta nuevas recepciones de datos, esto es, para evitar segmentar los mismos resultados varias veces antes de cada recepción y desperdiciar procesos y procesos:

```
if (segmentacion == true){
  int index = 0;
  resultado = strtok( datos, separador );           /* Separar datos
                                        correspondientes a C, F y K. */
  while((resultado != NULL) && (index < 3)) {
    valores[index++] = atoi(resultado);     /* Almacenamos los datos
  segmentados en el array y los transformamos de string a entero. */
    resultado = strtok( NULL, separador );          /* Anulamos los
                        resultados para una próxima segmentación. */
  }
  segmentacion = false;                /* Después de la segmentación,
        lo desaprobamos (false) hasta nuevos datos de recepción. */
}
```

Después asignamos cada escala correspondiente a Celsius, Fahrenheit y Kelvin del vector valores:

```
Grados_Cent = valores[0];       /* Leer los datos correspondientes a
                      grados Celsius en la posición 0 del array. */
Grados_Fahr = valores[1];       /* Leer los datos correspondientes a
                    grados Fahrenheit en la posición 1 del array. */
Grados_Kelv = valores[2];       /* Leer los datos correspondientes a
                      grados Kelvin en la posición 2 del array. */
```

Seguidamente imprimimos los resultados en la pantalla LCD durante cada medio segundo a través de la función `millis()`. Los colocamos espaciadamente en la segunda fila de la pantalla:

```
if((millis() - Tiempo_LCD) > 500){
  lcd.clear();  // Limpiar pantalla antes de presentar nuevos datos.
  lcd.setCursor(0,1); // Fijar el cursor de la pantalla en 0,1.
  lcd.print(Grados_Cent);//Imprimimos temperatura en grados Celsius.
  lcd.print("C");          // Imprimir unidad.
  lcd.setCursor(6,1); // Fijar el cursor de la pantalla en 6,1.
  lcd.print(Grados_Fahr);//Imprimir temperatura en grados Fahrenheit
  lcd.print("F");          // Imprimir unidad.
  lcd.setCursor(12,1); // Fijar el cursor de la pantalla en 12,1.
  lcd.print(Grados_Kelv);//Imprimimos temperatura en grados Kelvin.
```

```
    lcd.print("K");          // Imprimir unidad.
    Tiempo_LCD = millis(); // Tiempo de actualización.
  }
```

Finalmente transmitimos los resultados para el encendido de LED indicadores de temperatura en la tarjeta 1. La transmisión de datos es en cada medio segundo (500 ms) a través de la función `millis()`. Si la temperatura es baja envía el carácter `"b"`, si es normal envía el carácter `"n"` y si es alta envía el carácter `"a"`, y además imprime en la pantalla LCD un comentario para cada temperatura (baja, normal y alta).

```
if ((millis() - Tiempo_Tx) > 500){
  if ((Grados_Cent >= 20)&&(Grados_Cent <= 25)){
    lcd.setCursor(0,0);     // Fijar el cursor de la pantalla en 0,0.
    lcd.print("TEMPERATURA NORM"); // Imprimimos comentario.
    Serial.print("n");             // Enviar el carácter n.
  }
  else if (Grados_Cent < 20){
    lcd.setCursor(0,0       // Fijar el cursor de la pantalla en 0,0.
    lcd.print("TEMPERATURA BAJA"); // Imprimir comentario.
    Serial.print("b");             // Enviar el carácter b.
  }
  else{
    lcd.setCursor(0,0);     // Fijar el cursor de la pantalla en 0,0.
    lcd.print("TEMPERATURA ALTA"); // Imprimir comentario.
    Serial.print("a");             // Enviar el carácter a.
  }
  Tiempo_Tx = millis();
}
```

CÓDIGO DE PROGRAMACIÓN (TARJETA 2)

```
/*****************************************************************
 *     Práctica 7.3 - Transmisión serial de datos entre dos      *
 *                        tarjetas Arduino                       *
 *****************************************************************
 * Este ejemplo permite realizar transmisión serial de datos entre
 * dos tarjetas Arduino.
 * - La tarjeta 1 realizará el cálculo de temperatura en las tres
 * escalas (Celsius, Fahrenheit, Kelvin) y el encendido de LED
 * para temperaturas normales, bajas y altas.
 * - En la tarjeta 2 se presentarán las temperaturas en las tres
 * escalas en una pantalla LCD. El rango de temperatura normal es
 * de 20 a 25 °C.
 *****************************************************************/
```

```
#include <LiquidCrystal.h>                    // Incluimos librería LCD.

LiquidCrystal lcd(12, 11, 5, 4, 3, 2); // Inicializamos la librería
LCD con sus respectivos puertos de configuración.

char datos[13];                               // Variable array para
capturar datos de llegada.
int posicion = 0;                             // Variable de posición para
datos de llegada.

char *resultado = NULL;                       // Variable de segmentación de
datos string de llegada.
char separador[] = "CFK";                     // Variable separadora de
datos (Celsius, Fahrenheit, Kelvin).
int valores[] = {0,0,0};                      // Variable tipo array para
almacenar datos segmentados {Celsius, Fahrenheit, Kelvin}.
int Grados_Cent;                              // Variable para grados
centígrados.
int Grados_Fahr;                              // Variable para grados
Fahrenheit.
int Grados_Kelv;                              // Variable para grados
Kelvin.

unsigned long Tiempo_LCD;                     // Variable de tiempo para
impresión de texto en pantalla LCD.
unsigned long Tiempo_Tx;                      // Variable de tiempo para
transmisión de datos.

void setup() {
  lcd.begin(16, 2);                           // Inicializamos pantalla LCD
de 16x2.
  Serial.begin(115200);                       // Iniciamos el puerto de
comunicación a una velocidad de 115200bps.
  delay(1000);                                // Retardo.
  Tiempo_LCD = millis();                      // Leemos tiempo de inicio de
programa para pantalla LCD.
  Tiempo_Tx = millis();                       // Leemos tiempo de inicio de
programa para Tx.
}

void loop() {
  boolean segmentacion;                       // Variable para segmentación
de datos después de la recepción de datos.

  // Recepción de datos.
```

```
  while(Serial.available()>0){          // Mientras haya datos en el
buffer ejecuta la función
    delay(5);                           // Ponemos un pequeño retardo
para mejorar la recepción de datos
    datos[posicion] = Serial.read();    // Leemos un carácter del
string "cadena" de la "posición", luego lee el siguiente carácter con
"posición++"
    posicion++;                         // Aumentamos la variable en 1
para ir leyendo los datos de llegada uno a uno.
    segmentacion = true;                // Después de la recepción de
datos, aprobamos (true) para introducir la segmentación de datos.
  }
  posicion = 0;                         // Encerramos el contador de
posición para empezar de nuevo en la próxima recepción de datos.

  // Segmentación de datos.
  if (segmentacion == true){
    int index = 0;
    resultado = strtok( datos, separador );  // Separamos los datos
correspondientes a C, F y K.
    while((resultado != NULL) && (index < 3)) {
      valores[index++] = atoi(resultado);    // Almacenamos los datos
segmentados en el array y los transformamos de string a entero.
      resultado = strtok( NULL, separador ); // Anulamos los
resultados para una próxima segmentación.
    }
    segmentacion = false;               // Después de la segmentación,
lo desaprobamos (false) hasta nuevos datos de recepción.
  }

  Grados_Cent = valores[0];             // Leemos los datos
correspondientes a grados Celsius en la posición 0 del array.
  Grados_Fahr = valores[1];             // Leemos los datos
correspondientes a grados Fahrenheit en la posición 1 del array.
  Grados_Kelv = valores[2];             // Leemos los datos
correspondientes a grados Kelvin en la posición 2 del array.

  // Impresión en la pantalla LCD.
  if((millis() - Tiempo_LCD) > 500){
    lcd.clear();                        // Limpiamos la pantalla antes
de presentar los nuevos datos.
    lcd.setCursor(0,1);                 // Fijamos el cursor de la
pantalla en columna 0, fila 1.
    lcd.print(Grados_Cent);             // Imprimimos temperatura en
grados Celsius.
```

```
    lcd.print("C");                          // Imprimimos la unidad.
    lcd.setCursor(6,1);                      // Fijamos el cursor de la
pantalla en columna 6, fila 1.
    lcd.print(Grados_Fahr);                  // Imprimimos temperatura en
grados Fahrenheit.
    lcd.print("F");                          // Imprimimos la unidad.
    lcd.setCursor(12,1);                     // Fijamos el cursor de la
pantalla en columna 12, fila 1.
    lcd.print(Grados_Kelv);                  // Imprimimos temperatura en
grados Kelvin.
    lcd.print("K");                          // Imprimimos la unidad.
    Tiempo_LCD = millis();                   // Tiempo de actualización.
  }

  // Transmisión de datos
  if ((millis() - Tiempo_Tx) > 500){

    if ((Grados_Cent >= 20)&&(Grados_Cent <= 25)){
      lcd.setCursor(0,0);                    // Fijamos el cursor de la
pantalla en columna 0, fila 0.
      lcd.print("TEMPERATURA NORM");         // Imprimimos comentario.
      Serial.print("n");                     // Enviamos el carácter n
para temperaturas normales.
    }
    else if (Grados_Cent < 20){
      lcd.setCursor(0,0);                    // Fijamos el cursor de la
pantalla en columna 0, fila 0.
      lcd.print("TEMPERATURA BAJA");         // Imprimimos comentario.
      Serial.print("b");                     // Enviamos el carácter b
para temperaturas bajas.
    }
    else{
      lcd.setCursor(0,0);                    // Fijamos el cursor de la
pantalla en columna 0, fila 0.
      lcd.print("TEMPERATURA ALTA");         // Imprimimos comentario.
      Serial.print("a");                     // Enviamos el carácter a
para temperaturas altas.
    }
    Tiempo_Tx = millis();
  }
}
```

Después de **Verificar** y **Cargar** los códigos a las tarjetas Arduino, observaremos en la primera tarjeta el encendido de los LED de acuerdo a las temperaturas leídas por el sensor LM35. En la segunda tarjeta Arduino observaremos en la pantalla las temperaturas del sensor en las tres escalas (Celsius, Fahrenheit y Kelvin) y un comentario de acuerdo al rango de medición, entre 20 y 25 grados temperatura normal, menor a 20 grados temperatura baja, y mayor a 25 temperatura alta.

Para el encendido de los LED en la primera tarjeta se puede hacer directamente el cálculo para el control de ellas en la misma, pero este capítulo trata de transmisión serial, por lo que los LED son controlados por la segunda tarjeta enviando instrucciones cuando estas deben encender y apagar de acuerdo a los rangos de temperaturas.

APÉNDICES

A. CÓDIGO ESTÁNDAR DE COLORES EN RESISTENCIAS

Existe una serie estándar de valores para las resistencias de baja potencia. Los códigos de bandas de colores indican el valor de la resistencia, así como una tolerancia. Los tipos más comunes de resistencias son los de composición de carbón y los de película de carbón.

El código de colores para el valor del resistor utiliza dos dígitos y un dígito multiplicador, en este orden, como se muestra en la figura A.1. Una cuarta banda designa la tolerancia. En la siguiente tabla A.1 se ilustran los valores estándar para los dos primeros dígitos.

La resistencia de un resistor con las cuatro bandas de colores puede escribirse como:

$$R = (a \times 10 + b)m \pm tolerancia$$

Donde a y b son los valores de la primera y la segunda banda, respectivamente, y m es un multiplicador. Estos valores de resistencia corresponden a resistores con tolerancias de 2 % y 5 % como se indica en la tabla A.1. En la tabla A.2 se ilustra el código de color, en las tablas A.3 y A.4 se muestran los códigos de colores para el multiplicador y la tolerancia, respectivamente. Considérese un resistor con las cuatro bandas amarilla, violeta, naranja y oro. La resistencia se escribe como:

$$R = \big((4 \times 10) + 7\big)1k\Omega \pm 5\%$$

$$R = 47k\Omega \pm 5\%$$

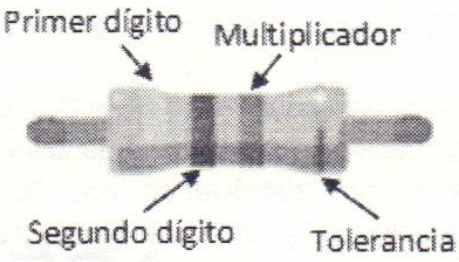


Figura A.1 Resistor con cuatro bandas de colores

Tabla A.1 Valores estándar de los dos primeros dígitos para resistores con tolerancias de 2 % y 5 %				
10	16	27	43	68
11	18	30	47	75
12	20	33	51	82
13	22	36	56	91
15	24	39	62	100

Tabla A.2 Código de colores	
0	Negro
1	Marrón
2	Rojo
3	Naranja
4	Amarillo
5	Verde
6	Azul
7	Violeta
8	Gris
9	Blanco

Tabla A.3 Código de colores del multiplicador	
Plata	0,01 Ω
Oro	0,1 Ω
Negro	1 Ω
Marrón	10 Ω
Rojo	100 Ω
Naranja	1 kΩ
Amarillo	10 kΩ
Verde	100 kΩ
Azul	1 MΩ
Violeta	10 MΩ
Gris	100 MΩ

Tabla A.4 Código de la banda de tolerancia	
Rojo	2%
Oro	5%
Plata	10%
Ninguno	20%

B. TABLA ASCII EXTENDIDA

Código ASCII (American Standard Code for Information Interchange).

Símbolo	Nombre	Decimal	Hexadecimal	Octal	Binario
NULL	Carácter nulo	00	00	000	00000000
SOH	Inicio de encabezado	01	01	001	00000001
STX	Inicio texto	02	02	002	00000010
ETX	Fin texto	03	03	003	00000011
EOT	Fin transmisión	04	04	004	00000100
ENQ	Consulta	05	05	005	00000101
ACK	Confirmación	06	06	006	00000110
BEL	Timbre	07	07	007	00000111
BS	Retroceso	08	08	010	00001000
HT	Tabulador horizontal	09	09	011	00001001
LF	Nueva línea	10	A	012	00001010
VT	Tabulador vertical	11	B	013	00001011
FF	Nueva página	12	C	014	00001100
CR	Retorno de carro	13	D	015	00001101
SO	Mayúsculas fuera	14	E	016	00001110
SI	En mayúsculas	15	F	017	00001111
DLE	Enlace de datos/Escape	16	10	020	00010000
DC1	Control de dispositivo 1	17	11	021	00010001
DC2	Control de dispositivo 2	18	12	022	00010010
DC3	Control de dispositivo 3	19	13	023	00010011
DC4	Control de dispositivo 4	20	14	024	00010100
NAK	Confirmación negativa	21	15	025	00010101
SYN	Sincronizado en espera	22	16	026	00010110
ETB	Fin de transmisión del bloque	23	17	027	00010111
CAN	Cancelar	24	18	030	00011000
EM	Fin del medio	25	19	031	00011001
SUB	Sustitución	26	1A	032	00011010
ESC	Escape	27	1B	033	00011011
FS	Separador de archivo	28	1C	034	00011100
GS	Separador de grupo	29	1D	035	00011101
RS	Separador de registro	30	1E	036	00011110
US	Separador de unidad	31	1F	037	00011111
	Espacio	32	20	040	00100000
!	Cierre de exclamación	33	21	041	00100001
"	Comilla doble	34	22	042	00100010
#	Numeral	35	23	043	00100011
$	Signo dólar	36	24	044	00100100
%	Signo de porcentaje	37	25	045	00100101

Símbolo	Nombre	Decimal	Hexadecimal	Octal	Binario
&	Signo et	38	26	046	00100110
'	Comilla simple	39	27	047	00100111
(	Paréntesis de apertura	40	28	050	00101000
)	Paréntesis de cierre	41	29	051	00101001
*	Asterisco	42	2A	052	00101010
+	Signo más	43	2B	053	00101011
,	Coma	44	2C	054	00101100
-	Guion o signo menos	45	2D	055	00101101
.	Punto, punto seguido o punto final	46	2E	056	00101110
/	Barra diagonal o signo de división	47	2F	057	00101111
0	Cero	48	30	060	00110000
1	Uno	49	31	061	00110001
2	Dos	50	32	062	00110010
3	Tres	51	33	063	00110011
4	Cuatro	52	34	064	00110100
5	Cinco	53	35	065	00110101
6	Seis	54	36	066	00110110
7	Siete	55	37	067	00110111
8	Ocho	56	38	070	00111000
9	Nueve	57	39	071	00111001
:	Dos puntos	58	3A	072	00111010
;	Punto y coma	59	3B	073	00111011
<	Menos que	60	3C	074	00111100
=	Igual	61	3D	075	00111101
>	Mayor que	62	3E	076	00111110
?	Cierre de interrogación	63	3F	077	00111111
@	Arroba	64	40	100	01000000
A	A mayúscula	65	41	101	01000001
B	B mayúscula	66	42	102	01000010
C	C mayúscula	67	43	103	01000011
D	D mayúscula	68	44	104	01000100
E	E mayúscula	69	45	105	01000101
F	F mayúscula	70	46	106	01000110
G	G mayúscula	71	47	107	01000111
H	H mayúscula	72	48	110	01001000
I	I mayúscula	73	49	111	01001001
J	J mayúscula	74	4A	112	01001010
K	K mayúscula	75	4B	113	01001011
L	L mayúscula	76	4C	114	01001100
M	M mayúscula	77	4D	115	01001101

Símbolo	Nombre	Decimal	Hexadecimal	Octal	Binario
N	N mayúscula	78	4E	116	01001110
O	O mayúscula	79	4F	117	01001111
p	P mayúscula	80	50	120	01010000
Q	Q mayúscula	81	51	121	01010001
R	R mayúscula	82	52	122	01010010
S	S mayúscula	83	53	123	01010011
T	T mayúscula	84	54	124	01010100
U	U mayúscula	85	55	125	01010101
V	V mayúscula	86	56	126	01010110
W	W mayúscula	87	57	127	01010111
X	X mayúscula	88	58	130	01011000
Y	Y mayúscula	89	59	131	01011001
Z	Z mayúscula	90	5A	132	01011010
[	Corchete de apertura	91	5B	133	01011011
\	Barra diagonal inversa	92	5C	134	01011100
]	Corchete de cierre	93	5D	135	01011101
^	Signo de intercalación	94	5E	136	01011110
_	Guion bajo	95	5F	137	01011111
`	Acento grave	96	60	140	01100000
a	a minúscula	97	61	141	01100001
b	b minúscula	98	62	142	01100010
c	c minúscula	99	63	143	01100011
d	d minúscula	100	64	144	01100100
e	e minúscula	101	65	145	01100101
f	f minúscula	102	66	146	01100110
g	g minúscula	103	67	147	01100111
h	h minúscula	104	68	150	01101000
i	i minúscula	105	69	151	01101001
j	j minúscula	106	6A	152	01101010
k	k minúscula	107	6B	153	01101011
l	l minúscula	108	6C	154	01101100
m	m minúscula	109	6D	155	01101101
n	n minúscula	110	6E	156	01101110
o	o minúscula	111	6F	157	01101111
p	p minúscula	112	70	160	01110000
q	q minúscula	113	71	161	01110001
r	r minúscula	114	72	162	01110010
s	s minúscula	115	73	163	01110011
t	t minúscula	116	74	164	01110100
u	u minúscula	117	75	165	01110101
v	v minúscula	118	76	166	01110110
w	w minúscula	119	77	167	01110111

Símbolo	Nombre	Decimal	Hexadecimal	Octal	Binario
x	x minúscula	120	78	170	01111000
y	y minúscula	121	79	171	01111001
z	z minúscula	122	7A	172	01111010
{	Llave de apertura	123	7B	173	01111011
|	Barra vertical	124	7C	174	01111100
}	Llave de cierre	125	7D	175	01111101
~	Signo de equivalencia	126	7E	176	01111110
DEL	Suprimir	127	7F	177	01111111
€	Símbolo de euro	128	80	200	10000000
		129	81	201	10000001
‚	Comilla tipográfica en forma de 9 simple y baja	130	82	202	10000010
ƒ	Letra latina minúscula q con gancho	131	83	203	10000011
„	Comilla tipográfica en forma de 9 doble y baja	132	84	204	10000100
…	Puntos suspensivos horizontales	133	85	205	10000101
†	Cruz	134	86	206	10000110
‡	Cruz doble	135	87	207	10000111
ˆ	Acento circunflejo modificador de letra	136	88	210	10001000
‰	Signo de por mil	137	89	211	10001001
Š	Letra latina mayúscula S con acento anticircunflejo	138	8ª	212	10001010
‹	Comilla tipográfica simple angular de apertura	139	8B	213	10001011
Œ	Ligadura latina mayúscula OE	140	8C	214	10001100
		141	8D	215	10001101
Ž	Letra latina mayúscula Z con acento circunflejo	142	8E	216	10001110
		143	8F	217	10001111
		144	90	220	10010000
‘	Comilla simple de apertura	145	91	221	10010001
’	Comilla simple de cierre	146	92	222	10010010
“	Comillas dobles de apertura	147	93	223	10010011
”	Comillas dobles de cierre	148	94	224	10010100
•	Viñeta	149	95	225	10010101
–	Guion corto	150	96	226	10010110
—	Guion largo	151	97	227	10010111
˜	Tilde pequeña	152	98	230	10011000
™	Signo de marca comercial	153	99	231	10011001
š	Letra latina minúscula s con acento anticircunflejo	154	9A	232	10011010

Símbolo	Nombre	Decimal	Hexadecimal	Octal	Binario
›	Comilla tipográfica simple angular de cierre	155	9B	233	10011011
œ	Ligadura latina minúscula OE	156	9C	234	10011100
		157	9D	235	10011101
ž	Letra latina minúscula z con acento anticircunflejo	158	9E	236	10011110
Ÿ	Letra latina mayúscula Y con diéresis	159	9F	237	10011111
	Espacio de no separación	160	A0	240	10100000
¡	Signo de exclamación de apertura	161	A1	241	10100001
¢	Signo de céntimo	162	A2	242	10100010
£	Signo de libra esterlina	163	A3	243	10100011
¤	Signo de moneda	164	A4	244	10100100
¥	Signo del yen	165	A5	245	10100101
¦	Barra vertical rota	166	A6	246	10100110
§	Signo de sección	167	A7	247	10100111
¨	Diéresis de espaciado: diéresis	168	A8	250	10101000
©	Signo de derechos de autor	169	A9	251	10101001
ª	Indicador de número ordinal femenino	170	AA	252	10101010
«	Comillas angulares dobles de apertura	171	AB	253	10101011
¬	Sin signo	172	AC	254	10101100
	Guion virtual	173	AD	255	10101101
®	Signo de marca comercial registrada	174	AE	256	10101110
¯	Acento largo de espaciado: sobrelínea	175	AF	257	10101111
°	Signo de grado	176	B0	260	10110000
±	Signo de más o menos	177	B1	261	10110001
²	Dos en superíndice: al cuadrado	178	B2	261	10110010
³	Tres en superíndice: al cubo	179	B3	263	10110011
´	Acento agudo: agudo de espaciado	180	B4	264	10110100
µ	Signo de micro	181	B5	265	10110101
¶	Signo de antígrafo: párrafo	182	B6	266	10110110
·	Coma georgiana	183	B7	267	10110111
¸	Cedilla de espaciado	184	B8	270	10111000
¹	Uno en superíndice	185	B9	271	10111001
º	Indicador de número ordinal masculino	186	BA	272	10111010

Símbolo	Nombre	Decimal	Hexadecimal	Octal	Binario
»	Comillas angulares dobles de cierre	187	BB	273	10111011
¼	Un cuarto	188	BC	274	10111100
½	Un medio	189	BD	275	10111101
¾	Tres cuartos	190	BE	276	10111110
¿	Signo de interrogación de apertura	191	BF	277	10111111
À	Letra latina mayúscula A con acento grave	192	C0	300	11000000
Á	Letra latina mayúscula A con acento agudo	193	C1	301	11000001
Â	Letra latina mayúscula A con acento circunflejo	194	C2	302	11000010
Ã	Letra latina mayúscula A con tilde	195	C3	303	11000011
Ä	Letra latina mayúscula A con diéresis	196	C4	304	11000100
Å	Letra latina mayúscula A con anillo encima	197	C5	305	11000101
Æ	Letra latina mayúscula AE	198	C6	306	11000110
Ç	Letra latina mayúscula C con cedilla	199	C7	307	11000111
È	Letra latina mayúscula E con acento grave	200	C8	310	11001000
É	Letra latina mayúscula E con acento agudo	201	C9	311	11001001
Ê	Letra latina mayúscula E con acento circunflejo	202	CA	312	11001010
Ë	Letra latina mayúscula E con diéresis	203	CB	313	11001011
Ì	Letra latina mayúscula I con acento grave	204	CC	314	11001100
Í	Letra latina mayúscula I con acento agudo	205	CD	315	11001101
Î	Letra latina mayúscula I con acento circunflejo	206	CE	316	11001110
Ï	Letra latina mayúscula I con diéresis	207	CF	317	11001111
Ð	Letra latina mayúscula ETH	208	D0	320	11010000
Ñ	Letra latin mayúscula N con tilde	209	D1	321	11010001
Ò	Letra latina mayúscula O con acento grave	210	D2	322	11010010

Símbolo	Nombre	Decimal	Hexadecimal	Octal	Binario
Ó	Letra latina mayúscula O con acento agudo	211	D3	323	11010011
Ô	Letra latina mayúscula O con acento circunflejo	212	D4	324	11010100
Õ	Letra latina mayúscula O con tilde	213	D5	325	11010101
Ö	Letra latina mayúscula O con diéresis	214	D6	326	11010110
×	Signo de multiplicación	215	D7	327	11010111
Ø	Letra latina mayúscula O con barra diagonal	216	D8	330	11011000
Ù	Letra latina mayúscula U con acento grave	217	D9	331	11011001
Ú	Letra latina mayúscula U con acento agudo	218	DA	332	11011010
Û	Letra latina mayúscula U con acento circunflejo	219	DB	333	11011011
Ü	Letra latina mayúscula U con diéresis	220	DC	334	11011100
Ý	Letra latina mayúscula Y con acento agudo	221	DD	335	11011101
Þ	Letra latina mayúscula THORN	222	DE	336	11011110
ß	Letra latina minúscula s afilada: ess-zed	223	DF	337	11011111
à	Letra latina minúscula a con acento grave	224	E0	340	11100000
á	Letra latina minúscula a con acento agudo	225	E1	341	11100001
â	Letra latina minúscula a con acento circunflejo	226	E2	342	11100010
ã	Letra latina minúscula a con tilde	227	E3	343	11100011
ä	Letra latina minúscula a con diéresis	228	E4	344	11100100
å	Letra latina minúscula a con anillo encima	229	E5	345	11100101
æ	Letra latina minúscula ae	230	E6	346	11100110
ç	Letra latina minúscula c con cedilla	231	E7	347	11100111
è	Letra latina minúscula e con acento grave	232	E8	350	11101000

Símbolo	Nombre	Decimal	Hexadecimal	Octal	Binario
é	Letra latina minúscula e con acento agudo	233	E9	351	11101001
ê	Letra latina minúscula e con acento circunflejo	234	EA	352	11101010
ë	Letra latina minúscula e con diéresis	235	EB	353	11101011
ì	Letra latina minúscula i con acento grave	236	EC	354	11101100
í	Letra latina minúscula i con acento agudo	237	ED	355	11101101
î	Letra latina minúscula i con acento circunflejo	238	EE	356	11101110
ï	Letra latina minúscula i con diéresis	239	EF	357	11101111
ð	Letra latina minúscula eth	240	F0	360	11110000
ñ	Letra latina minúscula n con tilde	241	F1	361	11110001
ò	Letra latina minúscula o con acento grave	242	F2	362	11110010
ó	Letra latina minúscula o con acento agudo	243	F3	363	11110011
ô	Letra latina minúscula o con acento circunflejo	244	F4	364	11110100
õ	Letra latina minúscula o con tilde	245	F5	365	11110101
ö	Letra latina minúscula o con diéresis	246	F6	366	11110110
÷	Signo de división	247	F7	367	11110111
ø	Letra latina minúscula o con barra diagonal	248	F8	370	11111000
ù	Letra latina minúscula u con acento grave	249	F9	371	11111001
ú	Letra latina minúscula u con acento agudo	250	FA	372	11111010
û	Letra latina minúscula u con acento circunflejo	251	FB	373	11111011
ü	Letra latina minúscula u con diéresis	252	FC	374	11111100
ý	Letra latina minúscula y con acento agudo	253	FD	375	11111101
þ	Letra latina minúscula thorn	254	FE	376	11111110
ÿ	Letra latina y minúscula con diéresis	255	FF	377	11111111

C. REFERENCIAS

[1]. "Open-source electronic prototyping platform allowing to create interactive electronic objects". [Online]. Disponible en: https://www.arduino.cc

[2]. J. F. Kelly, H. Timmis, "Challenge 1: Fun Stuff to Know", Arduino Adventures Escape form Gemini Station, Apress. 2013, pp. 1 – 18.

[3]. M. Schmidt, "Welcome to tha Arduino", Arduino A Quick-Start Guide, The Pragmatic Bookshelf. 2011, pp. 23 – 45.

[4]. D. Wilcher, Learn Electronics with Arduino, Apress. 2012.

[5]. M. McRoberts, Arduino Starters Kit Manual, Earthshine Design. 2009, pp. 9 – 18.

[6]. R. C. Dorf, J. A. Svoboda, "Código estándar de colores en resistencias", Circuitos Eléctricos, 6ª Ed. Grupo Alfaomega, John Wiley & Scns 2006, pp. 801-802.

ÍNDICE ANALÍTICO

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

Esta edición se terminó de imprimir en **junio** *de* **2016.** *Publicada por*
ALFAOMEGA GRUPO EDITOR, S.A. de C.V.
Dr. Isidoro Olvera (Eje 2 sur) No. 74, Col. Doctores, C.P. 06720,
Del. Cuauhtémoc, Ciudad México
La impresión y encuadernación se realizó en
CARGRAPHICS, S.A. de C.V. *Calle Aztecas No.27*
Col. Santa Cruz Acatlán, Naucalpan, Estado de México, C.P. 53150. México.

ARDUINO

Guía práctica

Arduino es una plataforma de hardware y software libre que permite a cualquier principiante en electrónica desarrollar proyectos reales. Gracias a su sencillez, bajo coste y facilidad de programación está especialmente indicada para desarrollar proyectos aplicados al mundo de la electrónica y la robótica.

Además, Arduino permite acceder a su entorno sin necesidad de adquirir licencias o derechos de autor, es decir, los desarrolladores pueden modificar, copiar y publicar cualquier diseño tanto de hardware como de software (desarrollar placas personalizadas, crear o modificar librerías, y copiar o compartir códigos con el resto de desarrolladores).

Escrito con un lenguaje sencillo y descriptivo, este libro le permitirá desarrollar sus capacidades de programador y le ayudará a conocer las numerosas posibilidades y funcionalidades que ofrece esta polivalente placa. A lo largo de sus páginas encontrará multitud de ejemplos prácticos que le permitirán trabajar en sus propios proyectos de forma imaginativa y muy creativa, solo con la ayuda de una tarjeta Arduino UNO y un conjunto de componentes electrónicos (LEDs, resistencias, potenciómetros, sensores, motores, etc.).

Esta guía práctica está especialmente recomendada para estudiantes de todos los niveles, desde secundaria hasta universidad, y para aquellos usuarios interesados en electrónica y programación que deseen aprender de forma rápida y autodidacta.

Byron O. Ganazhapa, es ingeniero en electrónica y telecomunicaciones, y programador desde 2008. Especialista en programación industrial, C/C++, y desarrollo de proyectos electrónicos. Ha sido un desarrollador activo de proyectos en Arduino desde 2011 y en la actualidad desempeña el cargo de desarrollador de aplicaciones automatizadas y de control.

www.alfaomega.com.mx

ÁREA	SUBÁREA
Eléctrica-Electrónica	Mecatrónica y Robótica

ISBN 978-607-622-698-8

9 786076 226988

"Te acerca al conocimiento"

Alfaomega Grupo Editor