USERS

ARDUINO
DE CERO A EXPERTO

PROYECTOS PRÁCTICOS

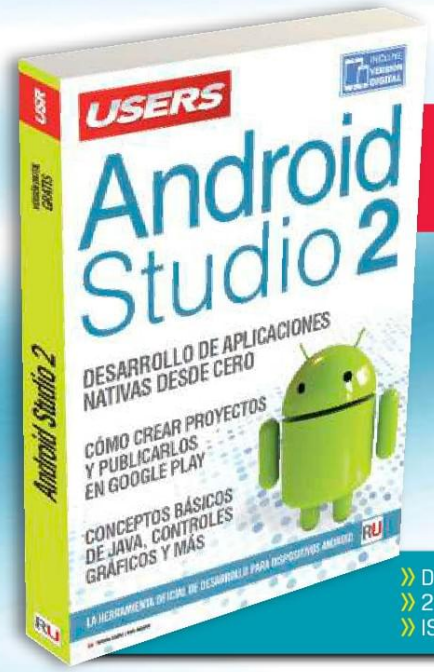
ELECTRÓNICA,
HARDWARE
Y PROGRAMACIÓN

LA GUÍA PARA REALIZAR TUS PROTOTIPOS ELECTRÓNICOS

RU

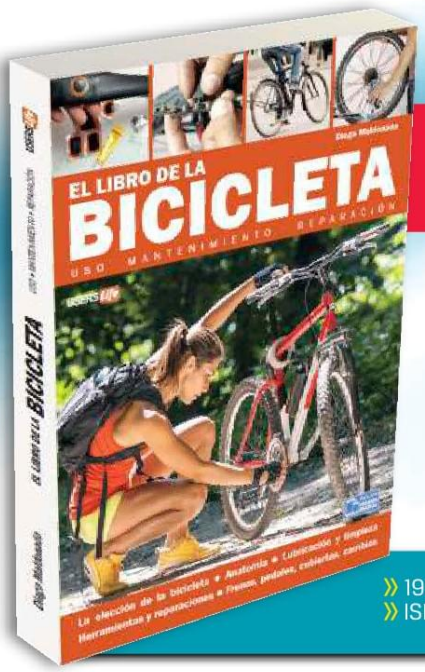
CLAUDIO PEÑA MILLAHUAL

ADQUIERA NUESTROS LIBROS EN USERSHOP
Y OBTENGA LA VERSIÓN DIGITAL GRATUITA



LA HERRAMIENTA OFICIAL DE DESARROLLO PARA DISPOSITIVOS ANDROID

» DESARROLLO
» 220 PÁGINAS
» ISBN 9789877340631

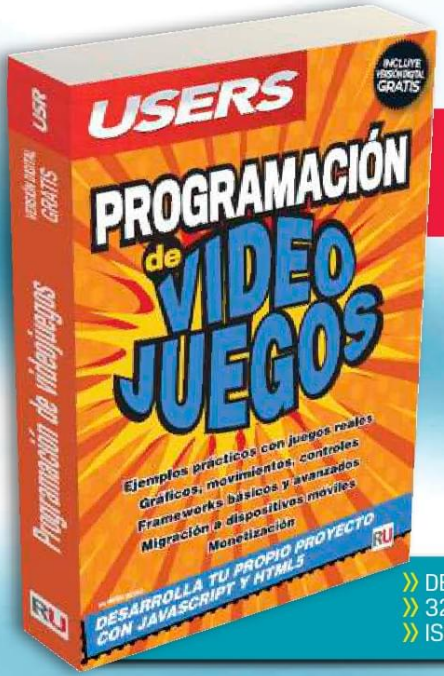


USO
MANTENIMIENTO
REPARACIÓN

» 192 PÁGINAS
» ISBN 9789877340617

TECNOLOGÍA DIGITAL AL ALCANCE DE TODOS

» EDUCACIÓN
» 320 PÁGINAS
» ISBN 9789877340648



DESARROLLA TU PROPIO PROYECTO CON JAVASCRIPT Y HTML5

» DESARROLLO
» 320 PÁGINAS
» ISBN 9789877340570

ARDUINO

LA GUÍA PARA REALIZAR TUS PROTOTIPOS ELECTRÓNICOS

De cero a experto - Proyectos prácticos paso a paso

por Claudio Peña Millahual

RedUSERS

Título: Arduino

Autor: Claudio Peña Millahual

Coordinador editorial: Miguel Lederkremer

Producción gráfica: Gustavo De Matteo

Edición: Lorena Blanco

Maquetado: Marina Mozzetti

Colección: Manuales Users

Formato: 24 x 17 cm

Páginas: 320

ISBN 978-987- 46518-7-7

Peña Millahual, Claudio Alejandro
Arduino / Claudio Alejandro Peña Millahual. - 1a ed. - Ciudad Autónoma de Buenos Aires : Six Ediciones, 2017.
320 p. ; 24 x 17 cm. - (Manuales Users ; 283)

ISBN 978-987-46518-7-7

1. Hardware. 2. Título.
CDD 004.64

CLAUDIO PEÑA

Nació en 1982 en Licán Ray, un pequeño pueblo del sur de Chile. Durante su vida, se ha dedicado a adquirir y profundizar los conocimientos en diversas áreas de la informática, tanto en forma académica como autodidacta. Posee estudios de Psicología e Informática, en la Universidad de la Frontera y Universidad de Los Lagos así como también un Bachillerato en Comunicaciones en UNIACC.

Desde el primer contacto con una computadora, a los 8 años, hasta el día de hoy, la necesidad de aprender y descubrir todo lo que ofrece la Informática no ha cambiado. Ha escrito diversos libros y variados artículos especializados, y además colabora como editor en diversas publicaciones de la editorial que lo vio nacer como autor.

A los 26 años, escribió su primer libro: 101 Secretos de Windows Vista. Luego vinieron títulos como PC Soluciones, Windows 7 Avanzado, Creación de distribuciones Linux, Proyectos con Windows, Redes Home, Windows Técnico, Office 2013, Windows 8, Windows 10 y Windows 10 Avanzado, entre muchos otros.

Actualmente Claudio dirige talleres de Arduino donde enseña conceptos básicos de programación y electrónica para niños y jóvenes.

PRÓLOGO

A pequeña o a gran escala, todo nuevo avance tecnológico siempre ha causado una revolución. Esto se puede observar en la introducción de las nuevas tecnologías en la vida diaria o en la modificación de los procesos productivos y de enseñanza, en todo ámbito, de tal forma que la introducción de tecnología resulta evidente en pequeños o grandes cambios.

En este sentido, Arduino no es la excepción. La importancia de esta plataforma ha trascendido todos los límites pues, en pocos años, ha dejado de ser una sencilla herramienta que solo perseguía fines educativos para transformarse en una plataforma que está presente en la primera línea de la industria tecnológica, en la creación de nuevos prototipos y en proyectos de diversa índole.

La clave que se encuentra tras el éxito de Arduino se relaciona con su carácter de proyecto abierto, no solo el software asociado se distribuye en forma libre, sino también las especificaciones técnicas de la placa Arduino están a disposición de todos. Gracias a esto, las empresas y los usuarios entusiastas están a pocos pasos de descargar, estudiar y producir nuevos dispositivos basados en Arduino.

Arduino se encuentra en el corazón de muchas de las propuestas tecnológicas actuales y, gracias a este libro, podemos dar los primeros pasos para entender su funcionamiento y así crear nuestros primeros proyectos.

EL LIBRO DE UN VISTAZO

El objetivo de este libro es brindarle al lector los conocimientos y las herramientas necesarias para desarrollar proyectos en Arduino, como la electrónica básica, el hardware y el entorno de programación. Por eso, esta obra contiene un paso a paso de distintos proyectos prácticos para que el lector obtenga sus primeros prototipos.

01

CONCEPTOS INICIALES

Antes de ingresar al mundo de Arduino, vamos a conocer las bases teóricas necesarias para acercarnos a la electrónica. A través de la información que brinda este libro, aprenderemos todo lo que se necesita para poner manos a la obra y lograr espectaculares creaciones.

02

¿QUÉ ES ARDUINO?

Dar la definición de Arduino no es una tarea sencilla, ya que se trata de una plataforma que incorpora hardware y software en apoyo de múltiples proyectos de electrónica, pero, además, esta se ha convertido en toda una filosofía en la que la premisa del hardware libre es un punto esencial.

03

¿QUÉ SE NECESITA?

Hasta aquí ya conocimos algunas de las placas de Arduino oficiales y, también, las no oficiales. Ahora, es el momento de profundizar en el conocimiento de Arduino UNO, la placa que utilizaremos en nuestros primeros proyectos, así como también aquellos componentes básicos que usaremos para iniciarnos en el mundo de Arduino.

04

ARDUINO IDE

Aquí conoceremos la forma en que nos comunicamos con nuestra placa de desarrollo: el Arduino IDE, donde a través de esta, vamos a contar con todo lo necesario para escribir los códigos que darán vida a nuestros proyectos.

05
PROGRAMAR ARDUINO
Luego de haber conocido las características principales, cómo instalarlo y cómo configurarlo, en este capítulo vamos a analizar la sintaxis adecuada para crear los sketchs para nuestros proyectos.

06
TRABAJAR CON LEDS
¡Manos a la obra! Aquí realizaremos nuestros primeros proyectos sencillos, para ello, utilizamos nuestra placa Arduino junto a un conjunto de LEDs y algunos componentes adicionales.

07
SENSORES
Una vez que pudimos trabajar en nuestros primeros proyectos utilizando una tarjeta Arduino junto con sus componentes básicos, vamos a conocer qué son los sensores y para qué sirven, además, veremos cómo pueden ayudarnos a completar nuestros proyectos.

08
DETECCIÓN DE LUZ
En esta ocasión, trabajaremos en detalle con un sensor LDR, para lograr proyectos que sean capaces de efectuar la detección del nivel o la intensidad de la luz.

09
EMISIÓN DE SONIDOS
Aquí vamos a agregar un componente más a nuestros proyectos: la emisión de sonidos. Sin duda esto abre nuevas oportunidades y nos permite generar novedosas propuestas electrónicas.

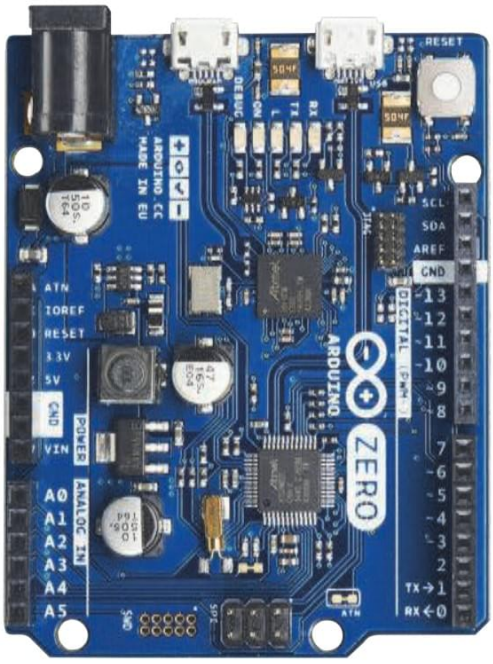
10
DISPLAY LCD Y RELOJ DIGITAL
En el desarrollo de este capítulo, realizaremos un pequeño pero interesante proyecto: un reloj digital. Para lograrlo, utilizaremos algunos componentes que ya conocemos, pero también agregaremos una pantalla LCD.

11
POTENCIAL DE ARDUINO
A lo largo de este libro, hemos dado los primeros pasos en el mundo de Arduino. Pero, aunque desarrollamos unos proyectos interesantes, debemos incrementar el enorme potencial que tiene Arduino, demostrado por la gran gama de posibilidades que conoceremos en este capitulo.

Ap
SHIELDS
En este apartado, conoceremos la placa Shield, un elemento apropiado para dotar de mayores capacidades a nuestro proyecto Arduino. Tambien veremos otras opciones disponibles.

Contenido

01
CONCEPTOS INICIALES



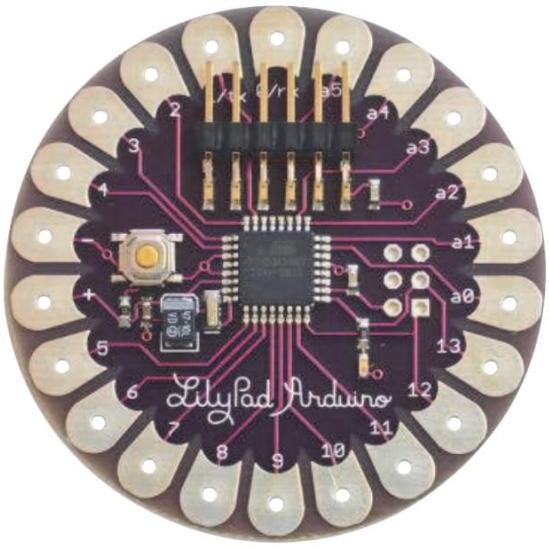
02
¿QUÉ ES ARDUINO?

03
¿QUÉ SE NECESITA?

04
ARDUINO IDE

05
PROGRAMAR ARDUINO

06
TRABAJAR CON LEDS



07
SENSORES

08
DETECCIÓN DE LUZ

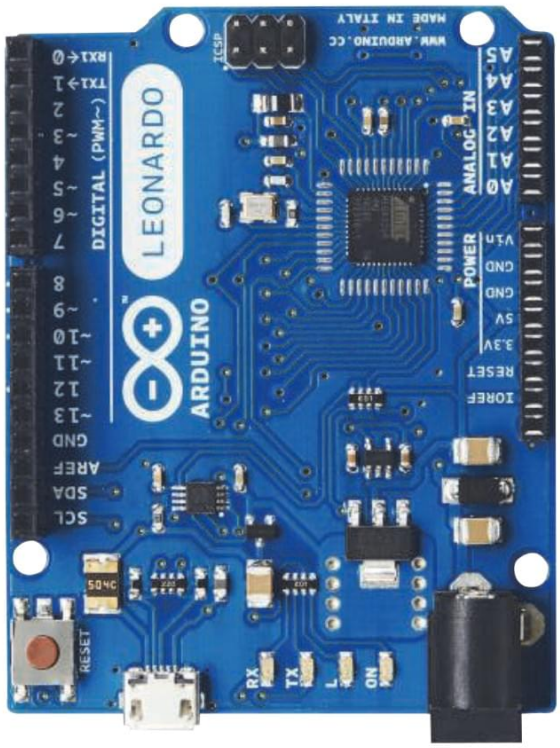
09
EMISIÓN DE SONIDOS

10
DISPLAY LCD Y RELOJ DIGITAL

11
POTENCIAL DE ARDUINO

Ap
SHIELDS

*
SERVICIOS AL LECTOR



Introducción

En la actualidad, Arduino está presente en todo tipo de dispositivos tecnológicos, en impresoras 3D, dispositivos médicos, avances domóticos, proyectos educativos, robots y drones, solo por nombrar algunos ejemplos.

El éxito de Arduino se basa en que se trata de placas económicas, accesibles, que pueden programarse en diferentes plataformas y, que tanto su software como su hardware se distribuye en forma libre.

Gracias a Arduino, pude combinar dos de mis grandes pasiones, la tecnología y la difusión del conocimiento y, en este libro, se plasma un sencillo y práctico camino que llevará a los lectores novatos e iniciados a descubrir todo el potencial que nos ofrece esta placa.

En cada capítulo de este libro, encontraremos material teórico junto a lecciones prácticas, que nos acompañarán a través del proceso de aprendizaje.

Con explicaciones sencillas pero detalladas, será posible experimentar el trabajo con LEDs, sensores, detectores de luz y emisión de sonidos. Pero también aprenderemos sobre las características más interesantes de la plataforma Arduino, el IDE de programación y los diferentes shields disponibles.

Claudio Peña Millahual

Conceptos iniciales

01

El mundo de Arduino es apasionante y, a través de los capítulos que componen este libro, aprenderemos todo lo que necesitamos para poner manos a la obra y lograr espectaculares creaciones. Pero, antes de eso, conoceremos las bases teóricas necesarias para acercarnos a la electrónica.

ELECTRICIDAD

La **electricidad** es un fenómeno físico que tiene como origen las cargas eléctricas y que manifiesta energía, como los fenómenos térmicos, mecánicos, luminosos o químicos, entre otros.

Consiste en un flujo de electrones que puede observarse naturalmente, por ejemplo, en los rayos, que son descargas eléctricas producidas por una transferencia energética entre la ionosfera y la superficie de la Tierra. También observamos electricidad en el funcionamiento del sistema nervioso del ser humano.

Su uso es común en la vida diaria; la aprovechamos en los electrodomésticos o en las máquinas grandes, como los trenes, y además está presente en los dispositivos electrónicos.

Naturaleza de la electricidad

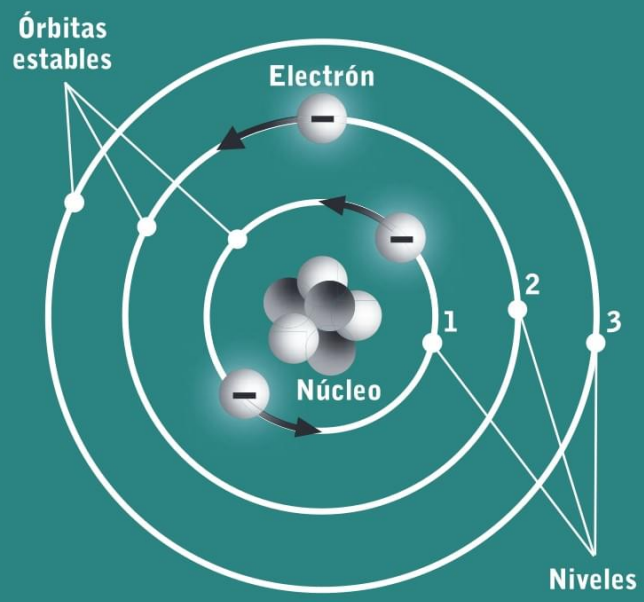
Podemos decir que la **materia** es todo aquello que tiene masa y que ocupa un lugar en el espacio; se compone de átomos, que están formados por partículas subatómicas: electrones, protones y neutrones. La materia también perdura en el tiempo.

Según el modelo atómico de Bohr, el electrón es la carga negativa (-) que se desplaza alrededor del núcleo en forma de capas concéntricas llamadas *órbitas*. Por otra parte, el protón es la carga positiva (+) que compone el núcleo del átomo junto a los neutrones que presentan una carga neutra.

El rayo es una de las manifestaciones más comunes de la electricidad en la naturaleza. También se presenta electricidad en el funcionamiento del sistema nervioso.

La **energía eléctrica** es causada por el movimiento de las cargas eléctricas que permanecen en el interior de los materiales conductores. Por ejemplo, cuando accionamos el interruptor de una lámpara, se cierra un circuito eléctrico y, por lo tanto, se genera el movimiento de electrones a través de cables metálicos. Además del metal, para que exista este transporte y se pueda encender la ampolleta, es necesario un generador o una pila que impulse el movimiento de los electrones en un sentido dado.

+ Modelo de Bohr



El modelo de Bohr explica cómo los electrones pueden tener órbitas estables alrededor del núcleo. En este modelo, los electrones giran en órbitas circulares alrededor del núcleo, y ocupan la órbita de menor energía posible o la más cercana al núcleo.

Diferencia de potencial

En el estudio de la electricidad, la diferencia de potencial entre dos puntos se conoce como **tensión**. Si entre dos puntos de un conductor no existe diferencia de potencial, la tensión entre ambos puntos es cero. Si entre esos dos puntos se ejerce un desequilibrio de cargas o un exceso de cargas negativas en un polo, aparecerá una tensión entre ambos puntos, que será mayor a medida que la diferencia de cargas sea también mayor. Esta tensión es la responsable de generar el flujo de electrones entre los dos puntos del conductor.

Los átomos de los elementos se diferencian por la cantidad de partículas subatómicas que poseen.

Las cargas eléctricas de signo opuesto se atraen y las del mismo signo se repelen, de esta forma cualquier electrón siempre será atraído por una carga positiva equivalente. Así, en un extremo de un material conductor se presenta un exceso de electrones, mientras que en el otro extremo existe una carencia de ellos (carga positiva). Los electrones tenderán a desplazarse a través de ese conductor desde el polo negativo al positivo; a esta circulación de electrones por un material conductor se la conoce como **electricidad**.

La electricidad existe mientras los electrones se desplazan de un extremo a otro del conductor; así el polo negativo será cada vez menos negativo y el polo positivo será cada vez menos positivo, hasta llegar el momento en el que ambos extremos tengan una carga global neutra o estén en equilibrio. En esta situación, el movimiento de los electrones cesará; para evitarlo, utilizaremos una fuente de alimentación externa o generador, para restablecer de manera constante la diferencia inicial de cargas entre los extremos del conductor.

Carga eléctrica

La **carga eléctrica** es una capacidad que tienen las partículas de poder atraer o repeler otras. Es la cantidad de energía que poseen las partículas que componen el átomo; este puede quedar cargado positivamente (si pierde electrones de sus órbitas) o negativamente

Ley de OHM

La ley de Ohm establece la relación fundamental de la electricidad, en la que se tienen tres elementos: tensiones, corrientes y resistencias.
Si se conocen dos de ellos, podemos calcular fácilmente el tercero: **V = R x I**.
De esta forma, si conocemos dos de las tres variables, es posible calcular la tercera:
- Si conocemos la tensión y la corriente, calculamos la resistencia como el cociente entre la tensión y la corriente: **R = V / I**.
- Si conocemos la tensión y la resistencia, calculamos la corriente como el cociente entre la tensión y la resistencia: **I = V / R**.

(si gana electrones). Juntas, generarán fuerzas de atracción y de repulsión tal como se puede observar cuando utilizamos un magneto y un trozo de metal, lo que crea un campo electromagnético.

Además, esta carga es la responsable de originar fuerzas capaces de producir, en su conjunto, fuerzas mecánicas. Se trata de una propiedad conservativa, esto quiere decir que se mantiene en el tiempo, o sea, que la carga inicial será la misma luego de un lapso indeterminado, siempre y cuando todo el sistema se encuentre aislado sin influencias externas.

Potencia eléctrica

La **potencia eléctrica** se define como la cantidad de energía entregada o absorbida por un elemento en un tiempo determinado; la unidad correspondiente en el Sistema Internacional de Unidades es el vatio (*watt*). La potencia eléctrica desarrollada en un cierto instante por un dispositivo es el producto de la diferencia de potencial entre dichos terminales y la intensidad de corriente que pasa a través del dispositivo. De esta forma, la potencia es proporcional a la corriente y a la tensión.

ELECTRÓNICA

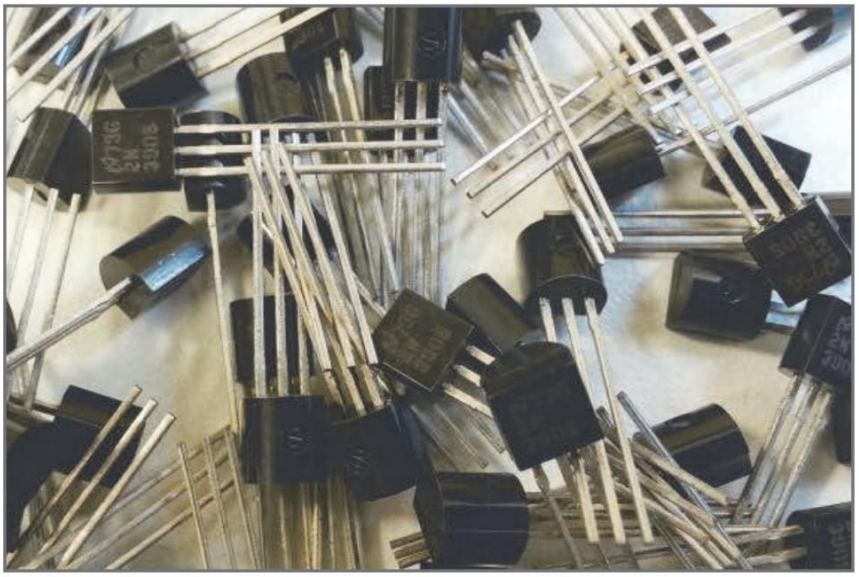
La **electrónica** es una rama de la física cuya finalidad es encargarse del control, la conducción y el flujo de los electrones o de cualquier partícula cargada eléctricamente.

Para simplificar, podemos decir que la electrónica se relaciona con el análisis de los electrones y con la aplicación de sus principios en contextos diferentes. En su noción más básica, la electrónica se relaciona con el **electrón**, una de las partículas esenciales de los átomos.

Los circuitos electrónicos hacen posible la conversión y la distribución de la energía eléctrica, por esta razón, los utilizamos en tareas que se relacionan con el procesamiento y el control de la información.

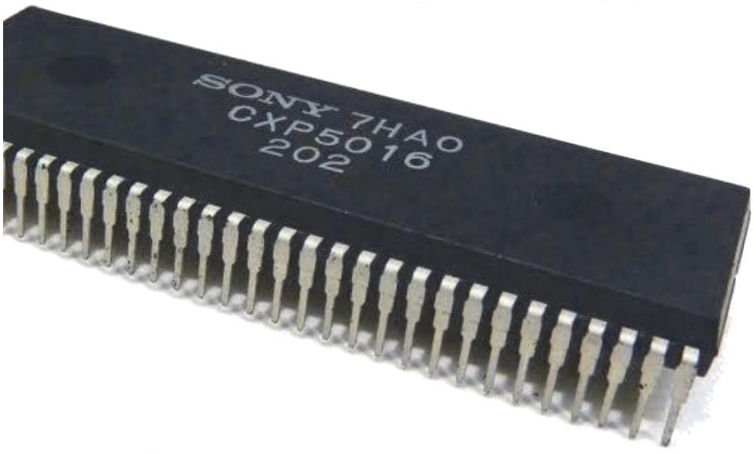
En términos generales, un sistema electrónico se forma por sensores que reciben las señales físicas y las transforman en señales de corriente. Los circuitos presentes en el sistema se encargan de interpretar y convertir las señales de los sensores que llegan hasta los actuadores, que transforman una vez más el voltaje en señales físicas.

En lo que a la historia se refiere, podemos mencionar que la introducción de los tubos de vacío a comienzos del siglo XX ayudó a que la electrónica moderna evolucionara. Los tubos de vacío hicieron posible la manipulación de señales, algo que no permitían los circuitos telegráficos y telefónicos que existían hasta ese momento.



Los transistores lograron reemplazar a los antiguos tubos de vacío, ofreciendo una mayor fiabilidad con menores costos.

Más tarde, el transistor logró reemplazar al tubo de vacío en la mayoría de sus aplicaciones; gracias a la incorporación de materiales semiconductores y contactos eléctricos es capaz de realizar las mismas funciones que el tubo de vacío, pero con un menor costo y una mayor fiabilidad. Luego del transistor, la tecnología ha evolucionado hasta los semiconductores y los circuitos integrados, que pueden contener miles de transistores en un pequeño espacio. Esto hace posible la construcción de circuitos electrónicos complejos, como los que se encuentran en microcomputadoras, equipos de sonido o satélites de comunicaciones.



En electrónica, conocemos al circuito integrado como una combinación de elementos de un circuito que están miniaturizados y que forman parte de un mismo chip o soporte.

Circuitos electrónicos

Para acercarnos a los circuitos electrónicos, debemos repasar los circuitos eléctricos. Cuando utilizamos una batería o un grupo electrógeno para producir electricidad, encontramos tres elementos que no cambian:

EL ORIGEN DE LA ELECTRICIDAD

Tendrá dos terminales: uno positivo y uno negativo.

EL ORIGEN DEL FLUJO ELÉCTRICO

Por ejemplo, un generador o una batería buscará empujar los electrones fuera de su terminal negativo, utilizando un cierto voltaje. Para ejemplificarlo pensemos en una pila, que se encarga de empujar los electrones a 1,5 voltios.

LOS ELECTRONES

Fluirán desde el terminal negativo al positivo por un cable de cobre u otro conductor. Cuando existe un camino desde el terminal negativo al positivo, tenemos un circuito, así los electrones pueden fluir por el cable.

En este punto es posible agregar una carga de cualquier tipo, por ejemplo, una bombilla, un motor, entre otros. De esta forma, la fuente de electricidad se encargará de alimentar la carga, y la carga desempeñará su función para crear luz o arrancar un motor, etcétera.

Trabajo de los electrones

Los electrones que se mueven por un circuito poseen energía, por lo tanto, son capaces de realizar un trabajo; por ejemplo, en una bombilla de filamento incandescente, la energía de los electrones se usa para crear calor y, a su vez, generar luz. Por otra parte, en un motor eléctrico, la energía en los electrones se encarga de crear un campo magnético que, finalmente, origina movimiento.

Aunque los circuitos pueden ser muy complejos, en un nivel básico siempre encontraremos en ellos la fuente de la electricidad o batería, la carga y los cables para conducir la electricidad entre la batería y la carga. Así, los electrones se mueven desde el origen, por la carga y de vuelta al origen.

Teniendo en cuenta lo dicho hasta este momento, podemos mencionar que los circuitos electrónicos son circuitos eléctricos que contienen dispositivos, tales como transistores y válvulas, entre otros. Son capaces de realizar funciones complejas utilizando cargas eléctricas, aunque funcionan con las mismas que los circuitos eléctricos. La importancia de los circuitos electrónicos radica en que conforman una asociación de componentes que pueden realizar un tratamiento de las señales eléctricas para almacenar información.

Los circuitos electrónicos se pueden clasificar en tres grupos:

CIRCUITOS ANALÓGICOS

En este tipo de circuitos, las señales eléctricas varían en forma continua para corresponderse con la información representada. El equipamiento electrónico, como los amplificadores de voltaje o de potencia, radios, televisiones, etcétera, suelen ser analógicos con la excepción de muchos dispositivos modernos que usan circuitos digitales.

CIRCUITOS DIGITALES

En ellos las señales eléctricas obtienen valores discretos para mostrar valores numéricos y lógicos que representen la información que se debe procesar. Algunos ejemplos de equipos con circuitos digitales son: calculadoras, celulares y microprocesadores.

CIRCUITOS MIXTOS

Se trata de circuitos híbridos, pues contienen elementos analógicos y también digitales. Un ejemplo es el convertidor de analógico a digital, o viceversa.

DIFERENCIAS ENTRE CIRCUITOS ELÉCTRICOS Y ELECTRÓNICOS

	CIRCUITOS ELÉCTRICOS	CIRCUITOS ELECTRÓNICOS
Componentes	Excepto el generador, sus componentes son pasivos.	Contiene al menos un elemento activo.
Control	Interruptores y resistencias controlan el flujo de la corriente.	El control se efectúa mediante señales eléctricas.
Uso	Se relacionan con la potencia.	Se relacionan con el almacenamiento de la información.
Tipo de corriente	Dependiendo del circuito, funcionan con corriente alterna o continua.	La mayoría funciona con corriente continua.

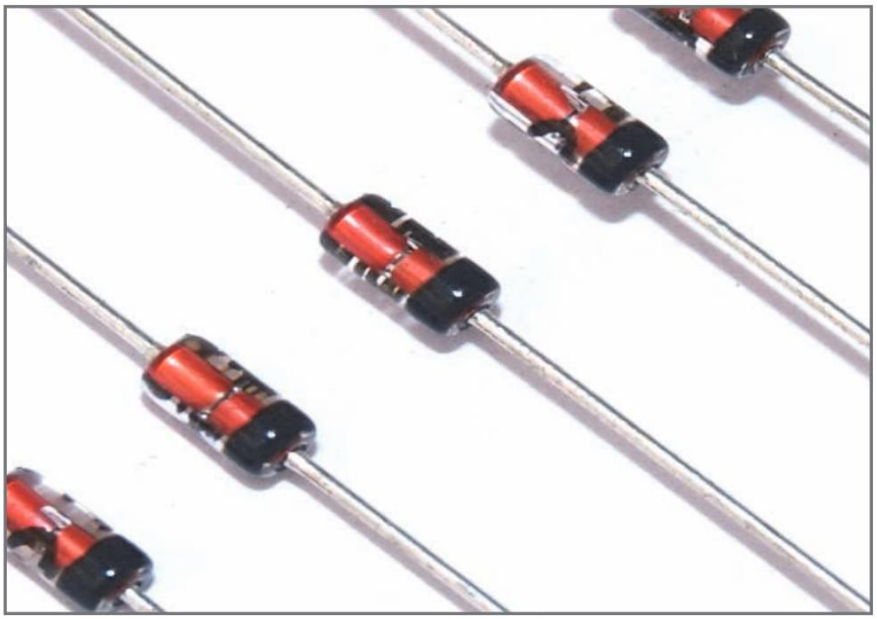
Componentes electrónicos

Los **componentes electrónicos** son aquellos que pueden formar parte de un circuito electrónico; por lo general se encuentran conectados mediante soldadura al circuito impreso.

Existen diferentes formas de clasificar los componentes electrónicos, por ejemplo, según su estructura física (discretos e integrados), según el material base de su fabricación (semiconductores, no semiconductores), según el tipo de energía (electromagnéticos, electroacústicos, optoelectrónicos). En esta ocasión, los clasificaremos según su funcionamiento –activos y pasivos– y su recubrimiento –de cerámica, de metal o de plástico– con la terminación de dos conectores para soldarlos al circuito.

Componentes activos

Se trata de componentes que pueden controlar el flujo de corriente o lograr ganancias. En la primera generación existían las válvulas, utilizadas en radio o televisión. En la segunda generación, aparecen los semiconductores; estos dieron paso a los circuitos integrados, que corresponden a la tercera generación.



El diodo Zener, encargado de la regulación de tensiones, es un ejemplo de componente activo.

COMPONENTES ELECTRÓNICOS ACTIVOS

COMPONENTE	USO
Amplificador operacional	Amplificación, regulación, conversión de señal, conmutación.
PLD	Control de sistemas estables.
Diodo Zener	Regulación de tensiones.
Memoria	Almacenamiento de datos.
Pila	Generación de energía.
Puerta lógica	Control de sistemas combinacionales.
Triac	Control de potencia.

Componentes pasivos

Los componentes electrónicos pasivos se encargan de realizar la conexión entre los componentes activos, de esta forma aseguran que las señales eléctricas puedan transmitirse o que se modifique su nivel.

COMPONENTES ELECTRÓNICOS PASIVOS	
COMPONENTE	**USO**
Inductor	También conocido como **bobina**, se encarga de atenuar o almacenar el cambio de energía.
Condensador	Almacena energía, filtra, adapta impedancia.
Resistor	También conocido como **resistencia**, se utiliza para la división de intensidad o tensión, también para limitar la intensidad.

Un **condensador** o **capacitor** es un componente pasivo capaz de almacenar energía. En la imagen vemos un condensador cerámico.

MICROCONTROLADORES

Los circuitos integrados son estructuras pequeñas, construidas con material semiconductor (generalmente silicio); contienen circuitos electrónicos y se encapsulan en plástico o cerámica para su protección.

Los **microcontroladores** son circuitos integrados programables que pueden ejecutar las tareas que han sido grabadas en su memoria.

Dentro de un microcontrolador encontramos tres unidades funcionales: **unidad central de procesamiento**, **memoria** y **periféricos de entrada/salida**; tal como observamos en una computadora. De esta forma podemos mencionar que un microcontrolador es una microcomputadora que se encuentra encapsulada en un circuito integrado.

Las aplicaciones de los microcontroladores son variadas y amplias, por ejemplo, es común encontrarlos en robótica y automatismo, en las telecomunicaciones, en el hogar y en la industria, etcétera. Si adaptamos la idea del microcontrolador al contenido de este libro, diremos que es posible utilizarlo para aplicaciones tales como manejo de sensores, calculadoras, avisos lumínicos, secuenciador de luces, cerrojos electrónicos, control de motores, robots, entre otros.

MCU y MPU

Aunque es común confundirlos, un microcontrolador (**MCU**) no es igual a un microprocesador (**MPU**).

En términos generales, un MCU usa una memoria flash para almacenar y ejecutar un programa, de esta forma presenta un período de arranque breve y, por lo tanto, es capaz de ejecutar el código más rápido. Si bien parece una ventaja, debemos considerar que esto conlleva una gran limitación práctica: su espacio de memoria es finito.

Por otra parte, un MPU no presenta las mismas restricciones de memoria, pues hace uso de una memoria externa para almacenar los datos. En general, el programa se guarda en una memoria no volátil (NAND o Flash en serie), pero en el arranque se carga en la DRAM externa para ejecutarse.

Teniendo en cuenta lo anterior, el MPU no estará en funcionamiento con tanta rapidez como un MCU, pero puede disponer de una cantidad de memoria mucho mayor, gracias al uso de recursos externos.

Otra diferencia importante entre un MCU y un MPU es que el primero solo necesita un riel de alimentación de tensión único, mientras que el segundo requiere varios rieles de tensión diferentes.

Arquitecturas de construcción

Por una parte, los MPU suelen diseñarse teniendo en cuenta la arquitectura Von Neumann; por otra parte, los MCU incorporan la memoria en su interior para hacer uso preferente de la arquitectura Harvard. En los MPU, los datos y el programa comparten la misma memoria; en los MCU, las instrucciones se ubican en una memoria interna aislada de la memoria de datos, y esto proporciona una mayor seguridad.

Desde los MCU hasta las placas de desarrollo

Si profundizamos en la historia, encontraremos que el microcontrolador comercial apareció en 1971, gracias al **Intel 4004** de 4 bits. Se trató de la segunda CPU completa de un solo chip y la primera comercial. Luego se presentó el **8008** de 8 bits (la base de las computadoras personales). En aquella época, también surgieron los procesadores **Z80** y el **6502**.

En realidad, el MCU **PIC,** de Microchip Technology (1975), fue uno de los más importantes para los fanáticos de la electrónica, pues era de bajo costo y se conseguía con facilidad. Como el PIC, es un MCU, contiene un procesador incorporado, memoria e I/O (in/outs) programables.

Ahora bien, trabajar con un microcontrolador PIC es difícil si no tenemos conocimientos profundos de programación C de bajo nivel, por ello se popularizaron los chips **PICAXE**, pues son capaces de entender lenguajes más sencillos, como BASIC o diagramas de flujos, que son utilizados en educación.

Los microcontroladores PIC o PIC micro son derivados del PIC1650.

En este complejo escenario, hacen su aparición las placas de desarrollo, que en la actualidad proliferan y se vuelven cada vez más accesibles y versátiles. Se trata de plataformas que se encargaron de democratizar el acceso a las herramientas de desarrollo que se encontraban restringidas por el alto costo del hardware y de los sistemas de desarrollo electrónico.

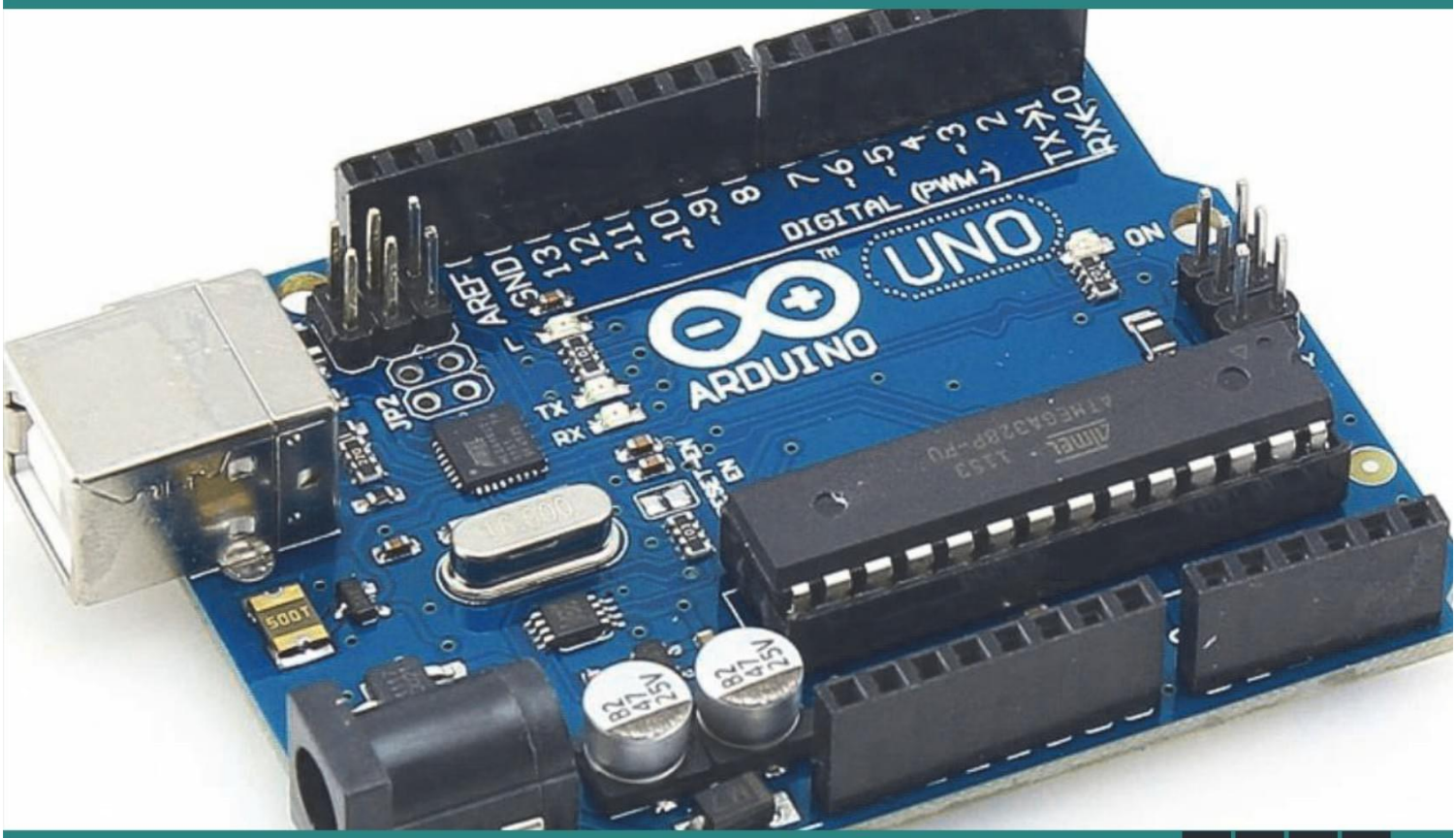
Una de las placas de desarrollo más populares es **Arduino**. La idea principal fue entregar acceso a MCU embebidos, pensando en proyectos de diseño interactivo. Gracias a esto, Arduino permite crear todo tipo de prototipos electrónicos en forma rápida y económica. La importancia de Arduino es tal que todo principiante, entusiasta y experto en el mundo de la electrónica lo utiliza para realizar sus proyectos.

Resumen Capítulo 01

En este capítulo hemos dado el primer paso en la tarea de trabajar con Arduino. Revisamos los conceptos iniciales relacionados con la electricidad y la electrónica, analizamos algunos de los componentes electrónicos más importantes y conocimos los microcontroladores. Para finalizar, realizamos una pequeña descripción en la que recorrimos la evolución de los microcontroladores hasta las placas de desarrollo.

¿Qué es Arduino?

02



Definir a Arduino no es una tarea sencilla, pues se trata de una plataforma que incorpora hardware y software en apoyo de múltiples proyectos de electrónica y, además, se ha convertido en toda una filosofía en la que la premisa del hardware libre es un punto esencial.

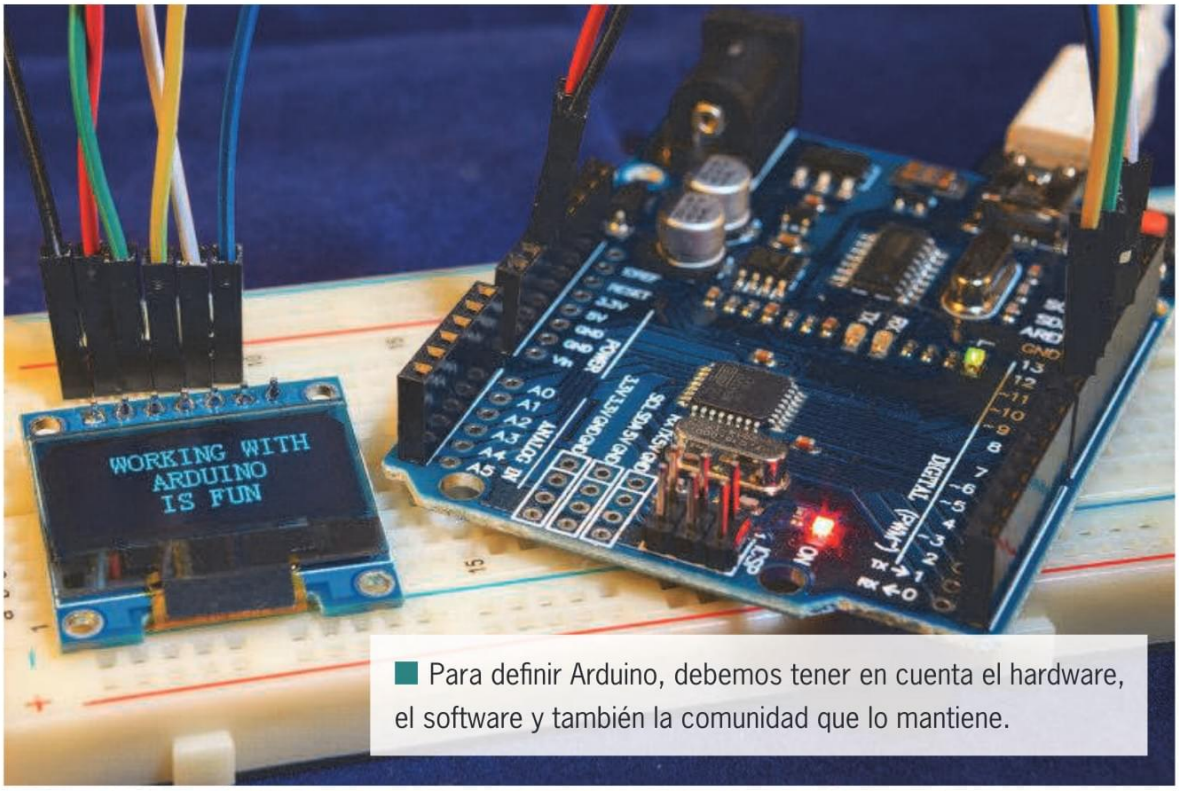
INTRODUCCIÓN

En términos formales, **Arduino** es una plataforma de hardware libre —creada por David Cuartielles y Massimo Banzi— basada en una placa con un microcontrolador y un entorno de desarrollo, y fue ideada para facilitar el uso de la electrónica en proyectos multidisciplinares, tanto para entusiastas como para expertos.

Si desmenuzamos esta definición, extraeremos ciertas ideas muy interesantes. Arduino es, a la vez, un sistema de procesamiento, un microcontrolador, una placa; también integra un entorno de desarrollo y es una plataforma de hardware open source.

Por otra parte, en forma simplificada, podemos mencionar que Arduino es una plataforma de hardware de código abierto, que basa su funcionamiento en una placa con entradas y salidas (analógicas y digitales), con un entorno de desarrollo que incorpora todo lo que necesitamos para crear nuestros programas.

Los componentes esenciales que nos permiten configurar una definición práctica para Arduino son el **hardware**, el **software** y la **comunidad** que lo mantiene.



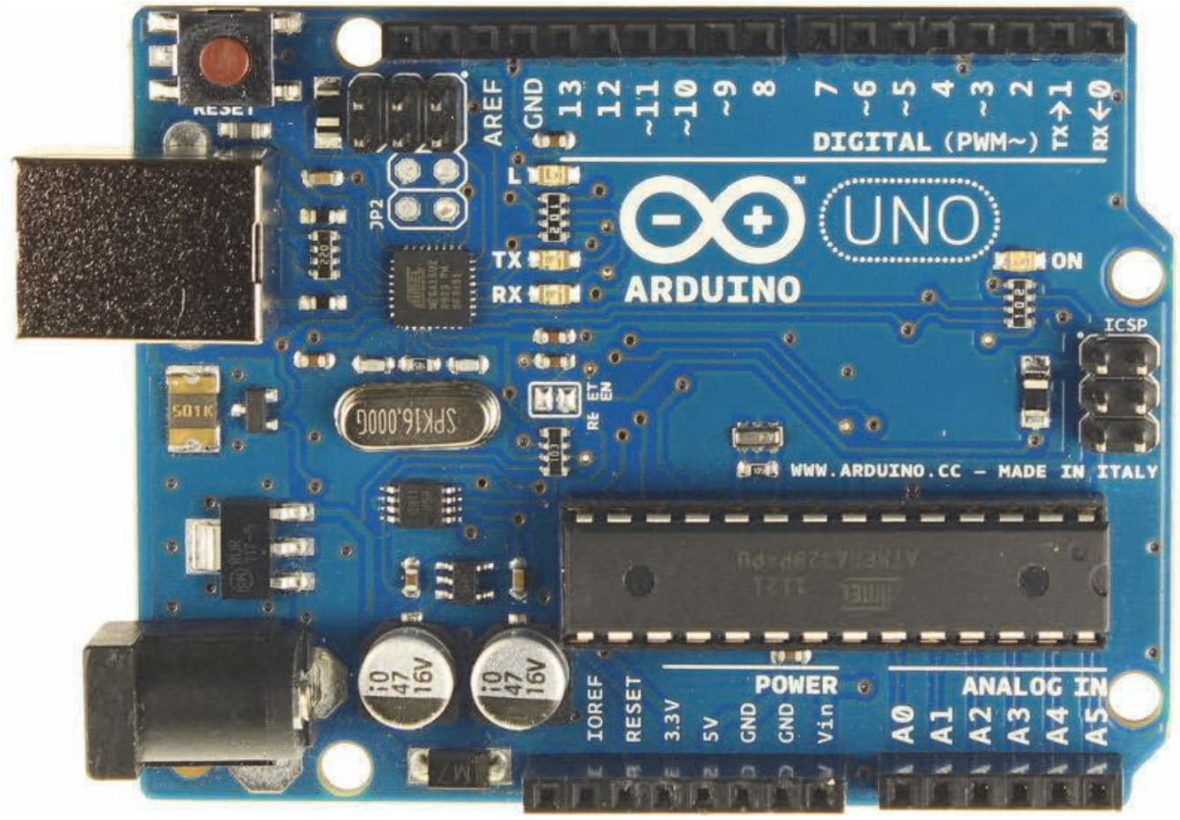
Para definir Arduino, debemos tener en cuenta el hardware, el software y también la comunidad que lo mantiene.

Hardware

En relación con el hardware, Arduino incorpora un microcontrolador que permite la programación con un lenguaje de alto nivel. Se trata del elemento encargado de efectuar los procesos matemáticos y lógicos, así como también de gestionar los recursos para cada componente externo que conectemos a la placa principal.

Una placa Arduino incorpora una serie de entradas analógicas y digitales, gracias a las que podremos conectar distintos sensores y otras placas o *shields*. Todo esto nos permite agregar nuevas funcionalidades sin necesidad de alterar el diseño original de la placa.

Un elemento importante dentro del hardware de Arduino son sus puertos de entrada/salida, mediante los que es posible conectar la placa a la computadora para integrar el trabajo con el software tal como veremos en la siguiente sección.

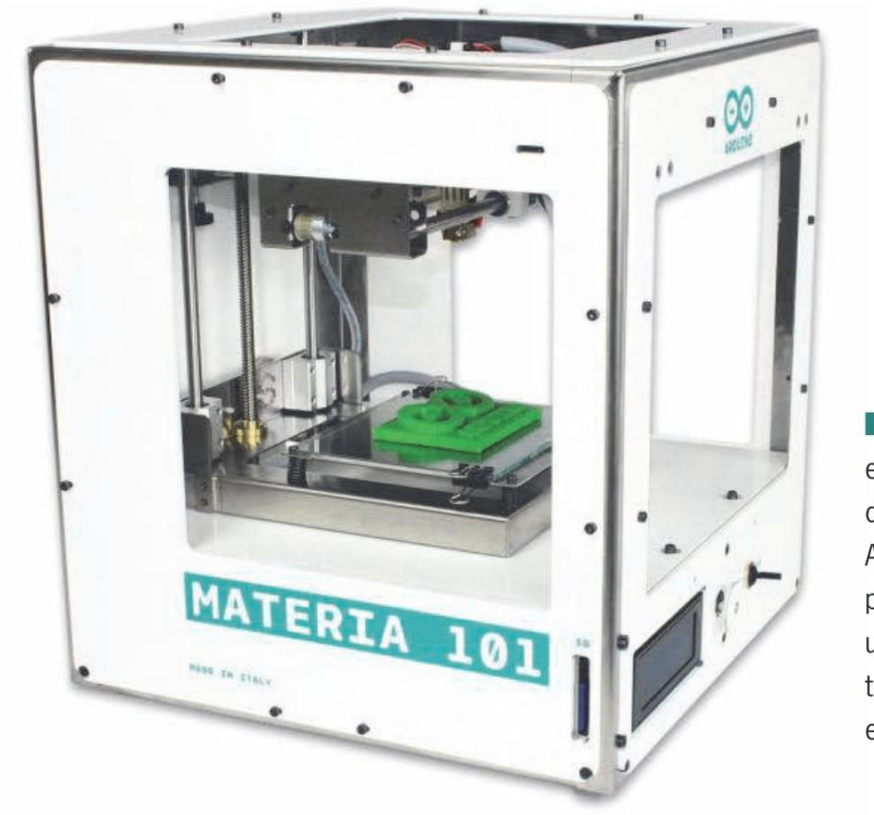


■ Una de las placas Arduino más conocidas es la **Arduino UNO**. En la imagen se puede apreciar el microcontrolador **Atmel**, integrado en esta placa.

En esencia, si analizamos el hardware de Arduino, encontraremos una placa de circuito impreso con un microcontrolador (**Atmel AVR**), y un conjunto de puertos digitales y analógicos de entrada/salida. Además, posee un puerto USB mediante el que se alimenta y se comunica con la PC.

Arduino se presenta en diversas categorías, que utilizaremos dependiendo del tipo de proyecto que deseemos implementar; entre ellas encontramos **placas**, **placas de expansión** o **shields**, **kits** y **accesorios**; adicionalmente, hallamos la categoría de **impresoras 3D**, donde se ubica **Arduino Materia**.

La principal categoría de Arduino son las placas, tanto placas de desarrollo como de expansión; analizaremos algunas de ellas en detalle más adelante, en este mismo capítulo.



Materia 101 es la impresora 3D diseñada y fabricada por Arduino. Al igual que las placas Arduino, se trata de un sistema open source tanto en el software como en el hardware.

Los distintos modelos de placas Arduino poseen especificaciones distintivas, por lo que es necesario conocerlas para saber cuál debemos utilizar en un proyecto concreto. En la siguiente tabla resumimos algunas características de hardware esenciales para ciertas placas Arduino.

PLACAS ARDUINO Y SUS PRINCIPALES CARACTERÍSTICAS

MODELO	MICROCONTROLADOR	ENTRADAS/ SALIDAS DIGITALES	ENTRADAS/ SALIDAS ANALÓGICAS	MEMORIA FLASH
Arduino Leonardo	ATmega 32U4	20	12	32 kb
Arduino UNO R3	ATmega 328	14	6	32 kb
Arduino Mega 2560 R3	ATmega 2560	54	16	256 kb
Arduino Mega pro 3.3V	ATmega 2560	54	16	256 kb
Arduino mini 05	ATmega 328	14	6	32 kb
Arduino Fio	ATmega 328P	14	8	32 kb
Arduino Mega Pro Mini 3.3V	ATmega 2560	54	16	56 kb
Arduino DUE	AT91SAM 3X8E	54	12	512 kb

La importancia de estos datos radica en que condicionarán el tipo de placa en función del proyecto en el que deseamos trabajar. En primer lugar, debemos saber la cantidad de pines analógicos y digitales que necesitaremos para un proyecto específico, y dependiendo de esto elegiremos una u otra placa.

Más adelante tendremos que deducir el tamaño del código que generaremos; esto es importante pues, en programas que utilicen muchas variables o constantes, necesitaremos una mayor cantidad de memoria flash. También debemos considerar la cantidad de RAM disponible y si precisamos un microcontrolador de 8 o de 16 bits; además hay que tener en cuenta cuál es el voltaje que la placa puede manejar. Todo esto resultará en la elección de una placa o de otra; en otros capítulos conoceremos las características específicas de algunos de los modelos más utilizados de Arduino.

Fuente de alimentación

Un tema importante que debemos tener en cuenta a la hora de comenzar a trabajar con Arduino es la necesidad de contar con una fuente de alimentación eléctrica.

En principio utilizaremos la energía proporcionada por la PC, mediante una conexión USB, de esta forma, la alimentación no será un problema cuando estemos programando nuestra placa o mientras esté conectada a la computadora. Pero ¿qué haremos después? Cuando no es una opción tenerla permanentemente conectada a la PC, debemos probar otras alternativas, como por ejemplo, se pueden utilizar adaptadores de corriente, pilas AA o baterías LiPo.

Características generales

Aunque en las diferentes placas Arduino encontramos diversas especificaciones de hardware, en general sus características básicas serán similares a las siguientes: microprocesador ATmega328, 32 kb de memoria flash, velocidad de reloj de 16 MHz, 13 pines para entradas/salidas digitales, 5 pines para entradas analógicas, 6 pines para salidas analógicas (salidas PWM), voltaje de operación 5V, EEPROM 512 byte.

ADAPTADOR DE CORRIENTE

Es una alternativa similar a un cargador para teléfono móvil; resulta una opción adecuada para aquellos proyectos que no se moverán, es decir, que pueden funcionar conectados a un toma corriente de pared.

PILAS AA

Es posible poner varias pilas AA en serie para lograr el voltaje que necesitamos en nuestra placa, teniendo en cuenta que cada una nos proporciona 1.5V. Aunque se trata de una opción recomendable para proyectos que requieren movilidad, debemos considerar que su energía se consume rápido, por lo que tendremos que cambiarlas a menudo.

BATERÍAS LiPo

Es una opción más eficiente pues proporcionan energía por bastante tiempo, aunque en comparación con las tradicionales pilas AA presentan un costo mayor. Son baterías recargables, por lo que también necesitaremos un módulo cargador, ya que es necesario cargarlas adecuadamente para alargar su vida. Las baterías LiPo (polímero de litio) se componen de celdas de 3,7V cada una.

Software

Aunque lo que más nos llama la atención de Arduino es el hardware, la verdad es debemos considerarla mucho más que una placa de circuitos y componentes electrónicos. Es una completa plataforma que nos permite programar el código necesario para controlar el funcionamiento de los sensores que conectamos a la placa.

Gracias al software que integra Arduino, es posible establecer las instrucciones y los parámetros para controlar su funcionamiento y, de esta forma, generar nuestros propios proyectos.

Al igual que el hardware de Arduino, el software que necesitamos para programarlo se distribuye libremente, por eso, solo se precisa descargarlo desde su web oficial. Analizaremos este procedimiento en detalle en el **Capítulo 6** de este libro.

Como vemos, uno de los componentes importantes de Arduino es su software. Se trata de un **IDE** o **Entorno de Desarrollo Integrado**, es decir, un conjunto de herramientas que podemos utilizar para programar o desarrollar aplicaciones.

El IDE de Arduino incorpora todo lo que necesitamos para crear el código que controlará el funcionamiento de los sensores conectados a la placa Arduino, para dar vida a nuestros proyectos.

El IDE de Arduino se distribuye como un programa empaquetado, con todo lo que necesitamos para programar, así encontraremos lo siguiente:

EDITOR DE CÓDIGO

Se trata de un programa diseñado específicamente para que podamos crear y editar código fuente. Aunque es posible utilizar cualquier editor de texto plano para crear este tipo de código, un editor específico integra el reconocimiento del lenguaje de programación que utilizaremos.

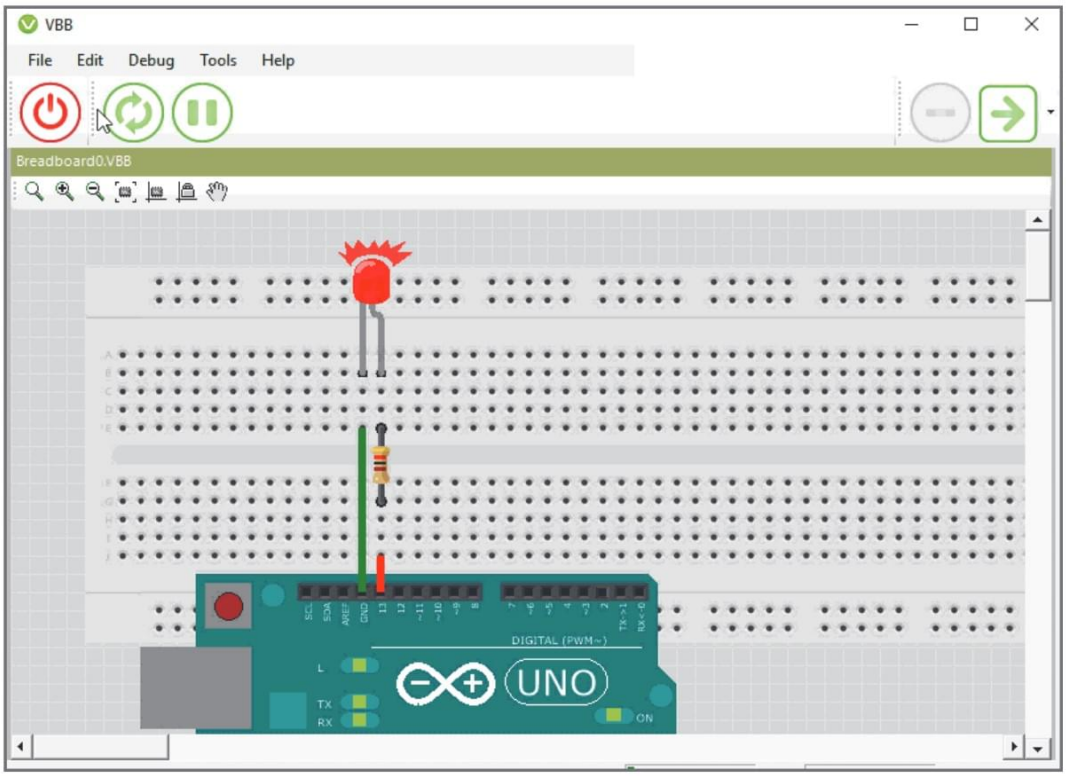
COMPILADOR

Es un programa informático que se encarga de traducir un programa que hemos desarrollado en un lenguaje de programación a un lenguaje diferente; en general traducirá nuestro código a lenguaje de máquina, entendible por el hardware.

DEPURADOR
Este tipo de programa está diseñado para probar y eliminar los errores que puedan existir en el código que desarrollamos. En otras palabras, se trata de un programa que ejecuta el código para detectar posibles errores lógicos.

Por si esto fuera poco, también dispondremos de las herramientas y opciones necesarias para que podamos cargar, a la memoria flash del hardware, los programas que realicemos. Es decir, es posible grabar los programas desarrollados para que Arduino los ejecute.

Antes de trabajar con Arduino, sobre todo si no tenemos experiencia en el manejo de circuitos electrónicos o placas de desarrollo, debemos considerar que, al principio, el manejo de este hardware y de este software podría ser algo complejo. Por esta razón, es una buena idea utilizar un simulador virtual para acercarnos al uso de Arduino, una excelente alternativa es **Virtual BreadBoard**, que encontramos en la dirección **www.virtualbreadboard.com**.



Un simulador virtual nos permite acercarnos al uso de las placas de desarrollo antes de enfrentarlas en forma física.

Comunidad

Como vimos hasta el momento, Arduino puede definirse teniendo en cuenta su hardware y su software. Pero, en realidad, esta plataforma es más que solo placas y código; un punto importante es la filosofía tras Arduino, la que descansa en su amplia comunidad.

La poderosa comunidad de Arduino ha sido muy relevante en el éxito de esta plataforma; se trata del grupo de usuarios, desarrolladores y entusiastas, que comparten contenido, publican proyectos y resuelven dudas, todo esto en pos de la divulgación de Arduino en el círculo de desarrolladores y amantes de la electrónica.

HARDWARE LIBRE

Sin duda, una de las principales características de Arduino es que se trata de una plataforma **Open Source**.

Si bien estamos acostumbrados a escuchar sobre software libre, es menos común encontrarnos con hardware que se clasifique en esta categoría. El hardware open source o libre es aquel para el cual las especificaciones o los diagramas esquemáticos están disponibles y son de acceso público. Entre las cuestiones para tener en cuenta a la hora de clasificar el hardware como libre, debemos considerar lo siguiente:

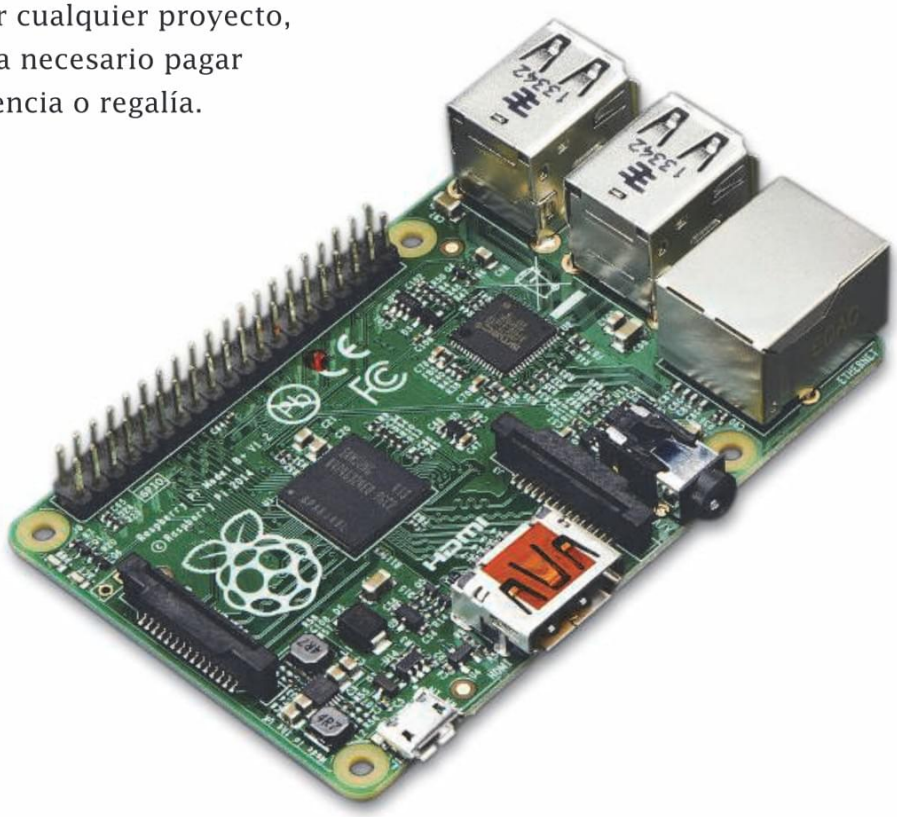
- Es necesario publicar la documentación incluyendo los archivos de los diseños, para efectuar su modificación y distribución.

Movimiento MAKER

Hace algún tiempo, ha surgido un movimiento denominado **Maker**; se trata de una opción que pretende rescatar la necesidad de hacer cosas en forma física sin depender exclusivamente de lo que podemos adquirir listo para utilizar. Este movimiento, relacionado con el mundo de Arduino, se basa en tres pilares: la aparición de herramientas digitales para el diseño y la fabricación, el uso de medios digitales colaborativos y el surgimiento de la fábrica para alquiler. En este sentido, una de las plataformas más amigables para el mundo Maker es Arduino.

- Se debe definir qué porción del diseño es abierta.
- Debe entregar el software necesario para leer el archivo del diseño y la documentación adecuada que se relaciona con sus funcionalidades, así se podrá escribir el código necesario en forma sencilla.
- Ofrecer una licencia que permita producir derivados y modificaciones.
- No se debe restringir la venta o el compartir la documentación necesaria.
- La licencia no debe discriminar ni restringir campos o actividades.
- La licencia no debe restringir otro hardware ni otro software. Además debe ser neutral, sin basarse en tecnologías específicas, partes o componentes, materiales o interfaces de su uso.

Si consideramos estos puntos, podemos definir a Arduino como **hardware libre**. De esta forma estamos frente a una plataforma que, tanto en su diseño como en su distribución, es libre. Es decir, podemos usarla para desarrollar cualquier proyecto, sin que sea necesario pagar alguna licencia o regalía.



Aunque es uno de los más conocidos, Arduino no es el único representante de hardware Open Source. En la imagen vemos a **RaspBerry Pi**, una computadora de placa reducida con propiedad registrada pero de uso libre.

CARACTERÍSTICAS ESENCIALES

Ahora que conocemos algo más sobre el mundo del hardware libre y también hemos ensayado una definición satisfactoria de Arduino, podemos enumerar sus principales características:

PRECIO ACCESIBLE

Sin duda, su bajo costo es una de las principales particularidades de Arduino. Si las comparamos con otras plataformas microcontroladoras, veremos que las placas Arduino son relativamente baratas, además, existen módulos que pueden ensamblarse en forma manual o en versiones reducidas, lo que disminuye su precio en el mercado.

MULTIPLATAFORMA

El paquete de software que acompaña a Arduino puede ser ejecutado en los sistemas operativos Windows, Mac OSX y GNU/Linux. En comparación, otros sistemas están limitados a un determinado sistema operativo.

ENTORNO DE PROGRAMACIÓN

El entorno en el cual podemos programar a Arduino es fácil de usar y de aprender. Pero también es flexible, por lo que puede ser utilizado por usuarios avanzados o educadores.

SOFTWARE ABIERTO

El software de Arduino está publicado como código abierto, por eso puede ser modificado y extendido por quien lo desee. Esto se realiza mediante el uso de librerías C++.

HARDWARE ABIERTO

Arduino se basa en microcontroladores de ATmel; sus planos están disponibles bajo la licencia Creative Commons, por lo tanto, es posible crear versiones propias de los módulos, para extenderlos y mejorarlos. Además, podemos fabricar nuestra propia versión de las placas, ya sea para entender su funcionamiento, para abaratar costos o para redistribuirla.

PLACAS DISPONIBLES

Podemos imaginar las placas Arduino como las distribuciones GNU/Linux, cada una de ellas preparada para atender necesidades de usuarios particulares, o para ser utilizadas en una serie de proyectos o tareas. Es necesario considerar que los modelos oficiales de placas Arduino alcanzan algunas decenas, pero, si sumamos los modelos no oficiales y los Arduino compatibles, con facilidad tendremos cientos.

En este punto hemos introducido un par de conceptos nuevos: **placas oficiales** y **no oficiales** de Arduino. ¿En qué se diferencian?

Por un lado, las placas oficiales son aquellas construidas por la empresa **Smart Projects**, por **SpartFun Electronics** o por **Gravitech**, las únicas que llevan la marca registrada Arduino y que incluyen su logo.

Por otro lado, las placas no oficiales son las que, si bien resultan compatibles, no pueden utilizar el nombre Arduino. Las diseñan otras compañías y, por lo general, se crean para cubrir necesidades específicas donde las placas oficiales no han llegado. En realidad, como hemos comentado en secciones anteriores, cualquiera puede crear su propia placa Arduino; en ese caso, pasaría a formar parte de las placas no oficiales.

Esto es importante a la hora de seleccionar una placa para trabajar en nuestros proyectos. Por ejemplo, es posible que necesitemos una placa compatible por alguna característica que no encontramos en una placa oficial, o que nos interese contar con una placa de desarrollo oficial, y entonces tendremos que elegir entre las manufacturadas por las empresas mencionadas antes. En cualquier caso, una de las más utilizadas es la **Arduino UNO**, sobre todo para quienes recién comenzamos en el mundo de Arduino. A continuación, conoceremos las características de algunas de las placas oficiales.

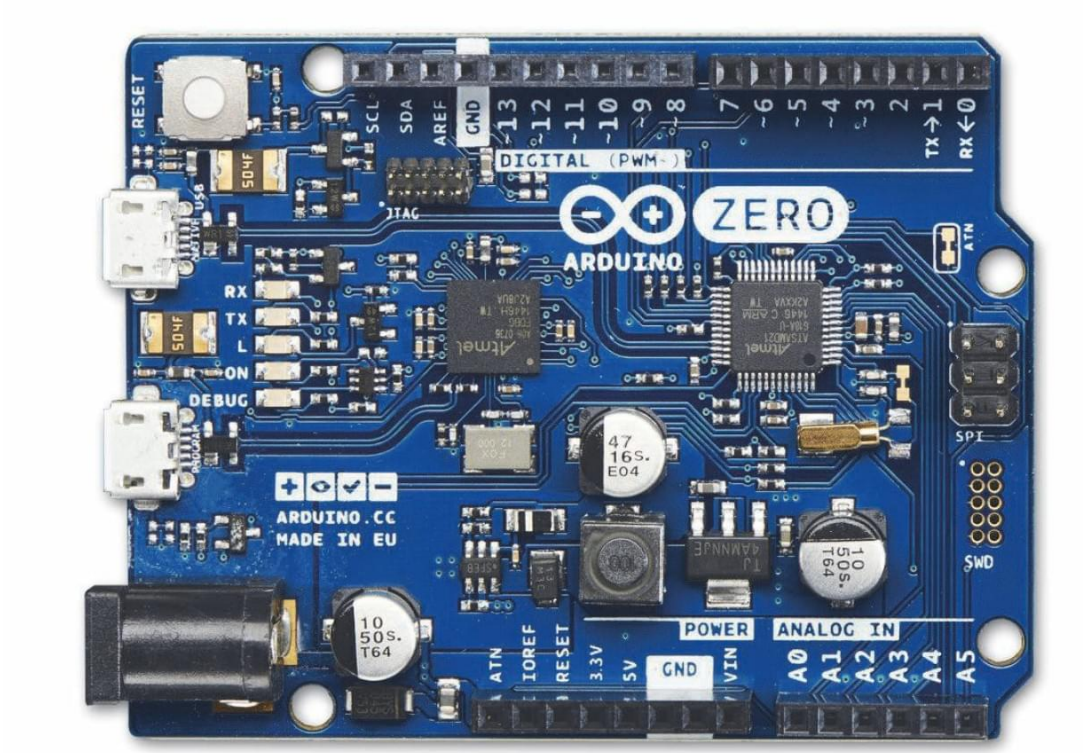
Arduino UNO

Se trata de la placa más extendida, la primera que apareció en el mercado y la más utilizada para todo tipo de proyectos. Sus características generales son las siguientes: un microcontrolador ATmega320 de 8 bits a 16 Mhza 5V. Posee 32 kb para la memoria flash con 0,5 kb reservados para el bootloader, 2 kb de SRAM y 1 kb de EEPROM; además ofrece 14 pines digitales y 6 analógicos. Aunque parece una placa limitada, resulta suficiente para una enorme cantidad de proyectos.

La Arduino UNO es una placa básica, pero contiene suficientes pines analógicos y digitales como para hacer frente a nuestros primeros proyectos. En la imagen vemos la Arduino UNO R3.

Arduino Zero

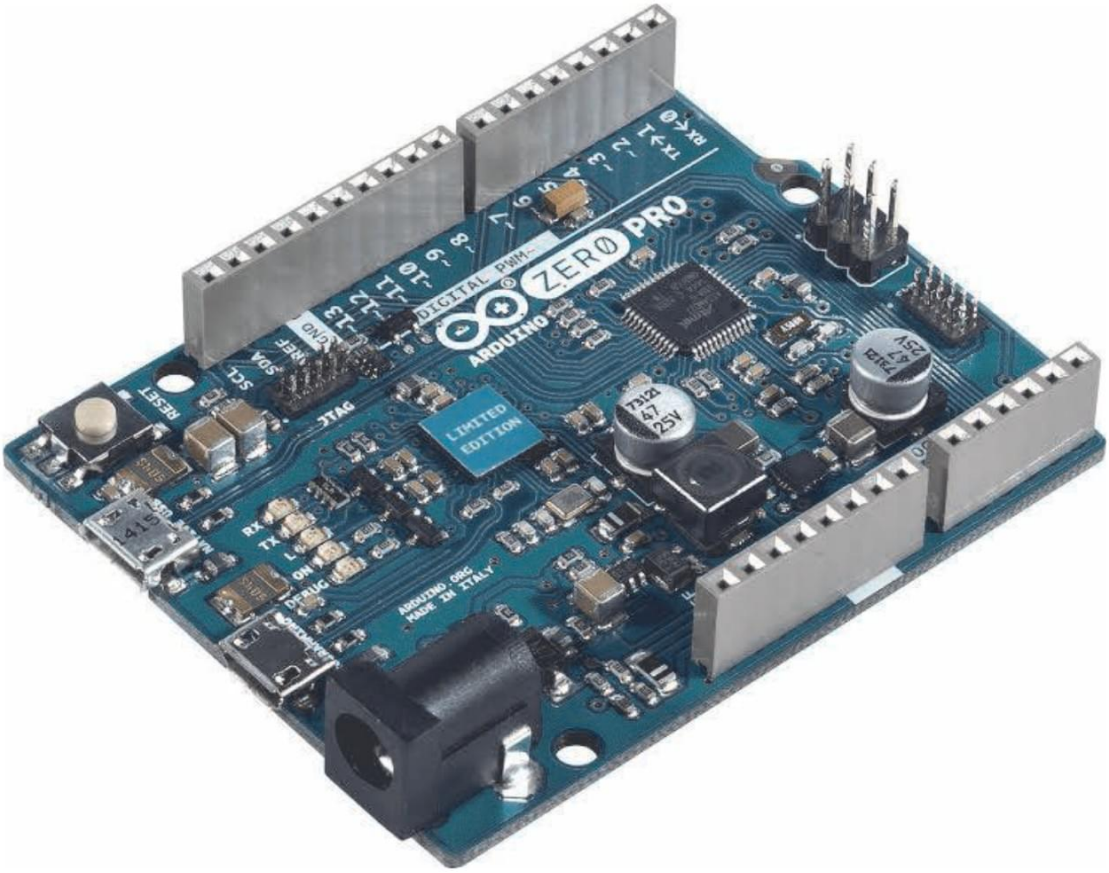
Esta placa es similar a la Arduino UNO, pero, en su arquitectura, utiliza un microcontrolador Atmel SAMD21 MCU de 48 Mhz e integra un core ARM Cortex M0 de 32 bits.
En ella encontraremos 256 kb de memoria flash, 32 kb de SRAM y una EEPROM de más de 16 kb por emulación. Ofrece 14 pines E/S digitales, y 6 entradas analógicas para un canal ADC de 12 bits, y una salida analógica para DAC de 10 bits. Se trata de una placa preparada para aquellos que sienten que Arduino UNO no ofrece lo que necesitan, es decir, nos ayudará a enfrentar proyectos avanzados.



La placa Arduino Zero supera por mucho las prestaciones de la Arduino UNO, por esta razón es la adecuada para acompañarnos en proyectos de mayor exigencia.

Arduino Zero Pro

Se trata de una versión mejorada en muchos aspectos en comparación con la placa Arduino Zero. Esta opción integra un microcontrolador de 32 bits, el Cortex M0+ basado en ARM, que corre a 48 Mhz, y se integra en un Atmel SAMD21 MCU. Sus demás características son similares a la placa Arduino Zero.



La Arduino Zero Pro mejora las prestaciones de cómputo ofrecidas por la placa Arduino Zero.

Arduino Yún

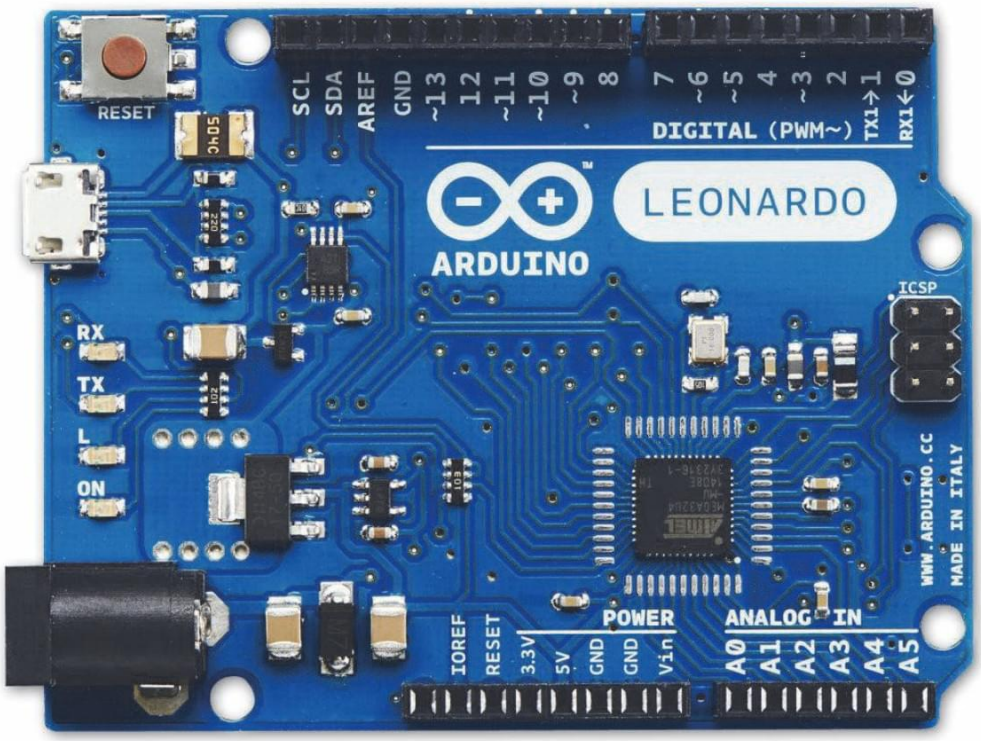
Una forma fácil de describir esta placa sería mencionarla como una opción con características similares a la Arduino UNO, pero que incorpora capacidad de conexión Ethernet, WiFi, USB y microSD sin que sea necesario agregar complementos adicionales.
En forma específica, la Arduino Yún basa su arquitectura en un microcontrolador ATmega32u4 (de 16 Mhz, trabaja a 5V, con una memoria de 32 kb, con 4 kb reservados al bootloader), y en un chip Atheros AR9331, que es el encargado de controlar el host USB, el puerto micro-SD y la red Ethernet/WiFi. Es interesante mencionar que el procesador Atheros soporta distribuciones Linux que se basan en OpenWrt. Esta placa nos ofrece 20 pines digitales, 12 analógicos, se complementa con el AR9331 que funciona a 400 Mhz basado en MIPS, es un chip que contiene RAM DDR2 de 64 MB y 16 MB flash para que podamos utilizar un sistema Linux embebido.

La Arduino Yún nos permite integrar el poder de los sistemas GNU/Linux con la facilidad de uso de Arduino.

Arduino Leonardo

Esta placa puede considerarse como una mezcla entre las Arduino UNO y las Arduino Yún. Por un lado, posee las capacidades de almacenamiento de la Arduino UNO y, por otro, nos ofrece los mismos pines que la placa Arduino Yún. Se basa en un microcontrolador ATmega32u4 de bajo consumo que trabaja a 16 Mhz, su memoria flash es de 32 kb con 4 kb para el bootloader, su EEPROM es de 1 kb. Si la analizamos a nivel de voltajes, encontraremos que es igual a Arduino UNO, pero nos entrega 20 pines digitales y 12 pines analógicos. Como podemos ver, cuenta con los mismos pines que Yún, pero no incorpora sus funciones de red.

Si la comparamos con Arduino UNO, Leonardo ocupa menos espacio, utiliza un conector mini USB, por lo que será eficiente en proyectos en los que necesitemos ahorrar espacio.



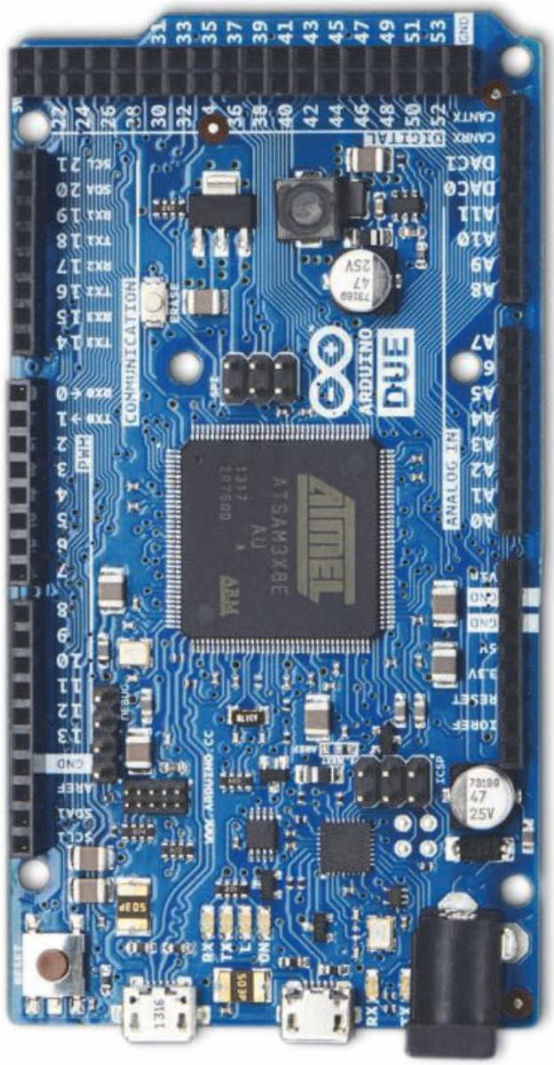
La placa Arduino Leonardo utiliza un conector mini USB para ahorrar espacio. Es ideal para proyectos que requieran dimensiones limitadas.

Arduino Due

Si trabajamos en proyectos que precisen capacidades de procesamiento mayores, la placa Arduino Due es lo que buscamos. Se basa en un microcontrolador Atmel SAM3X8E ARM Cortex-M3 de 32 bits, que trabaja a 84 Mhz entregando una potencia de cálculo superior a otros controladores, y permite realizar operaciones con datos de 4 bytes en un ciclo de reloj.

Presenta una memoria SRAM de 96 kb e integra un controlador DMA para acceso directo a memoria. Si estamos preocupados por la memoria flash, esta placa nos sorprende con 512 kb, un espacio bastante superior a otras placas Arduino.

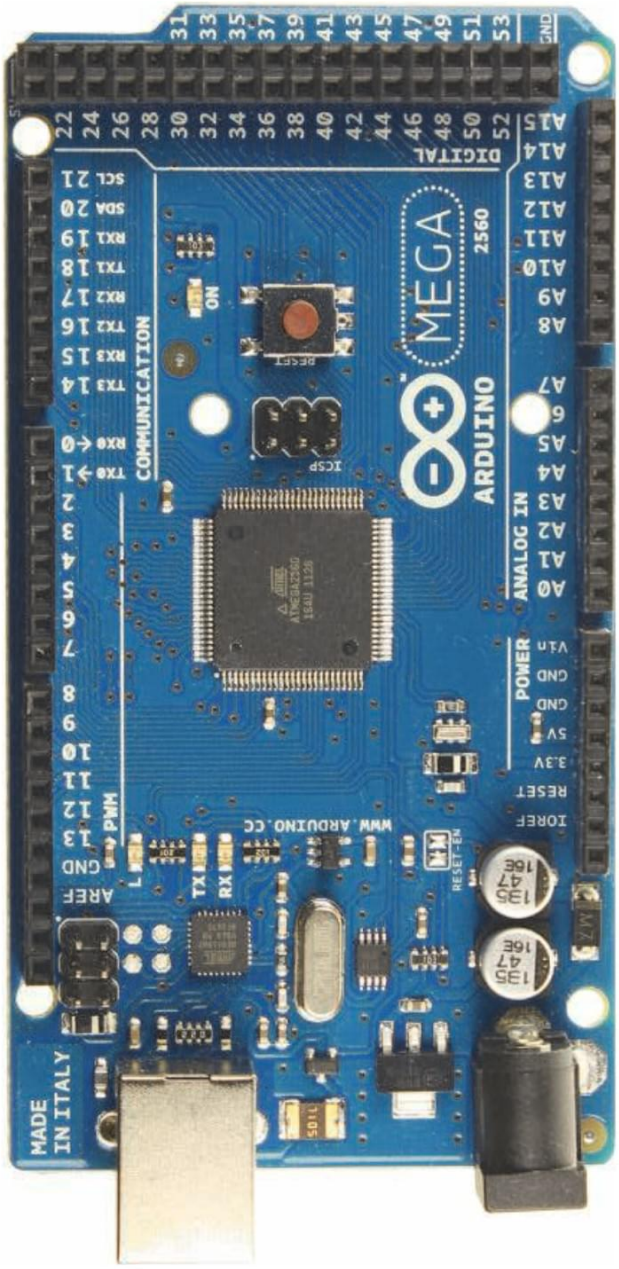
Dispone de 54 pines digitales y 12 analógicos, además posee conexión USB OTG, dos conexiones DAC, 2 TWI, un powerjack, SPI y JTAG.



La Arduino Due se encarga de aumentar la potencia de cálculo y la memoria flash, por lo que es una buena opción para proyectos exigentes.

Arduino Mega

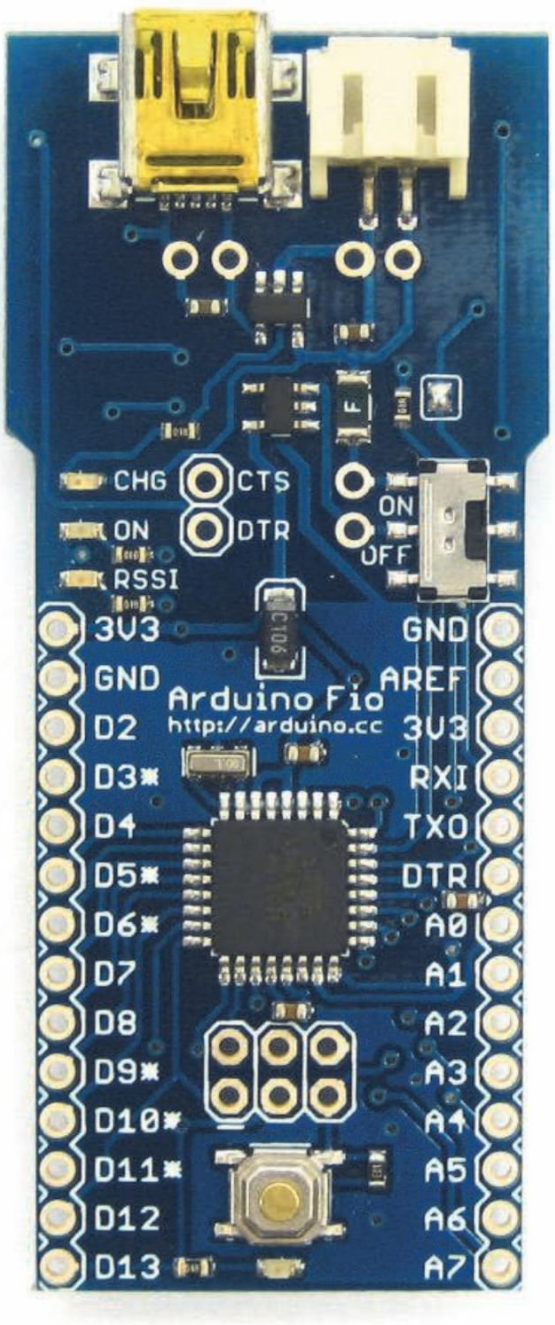
Sus prestaciones son similares a Arduino Due, pero se basa en una arquitectura AVR. Posee un microcontrolador ATmega2560 que trabaja a 16 Mhz, con 5V. Este microcontrolador (8 bits) trabaja con una SRAM de 8 kb, EEPROM de 4 kb y flash de 256 kb (con 8 kb para el bootloader). Además posee 4 pines digitales y 16 analógicos. Es superior al microcontrolador ATmega320 que encontramos en la placa Arduino UNO, pero no llega a compararse con las soluciones que se basan en ARM. La última versión de la placa Arduino Mega utiliza un microcontrolador ATMega8U2 en vez de un chip FTDI. Gracias a esto, ofrece mayor velocidad de transmisión por USB, y no es necesario cargar drivers para Linux o MAC.



La placa Arduino Mega nos ofrece un buen número de entradas digitales, gracias a las que podremos incorporar diferentes módulos para potenciar nuestros proyectos.

Arduino Fio

Si el espacio es una preocupación, la placa Arduino Fio podría ser lo que buscamos. Se trata de una placa que fue reducida al mínimo, pensada para proyectos inalámbricos o que deban ser acomodados en espacios pequeños. Posee un microcontrolador ATmega328P que trabaja a 8 Mhz. Debemos tener en cuenta que, para cargar los programas, será necesario utilizar un cable FTDI o una placa adicional. Posee 14 pines digitales y 8 analógicos, y funciona con 2 kb de SRAM, 32 kb de flash y 1 kb de EEPROM.

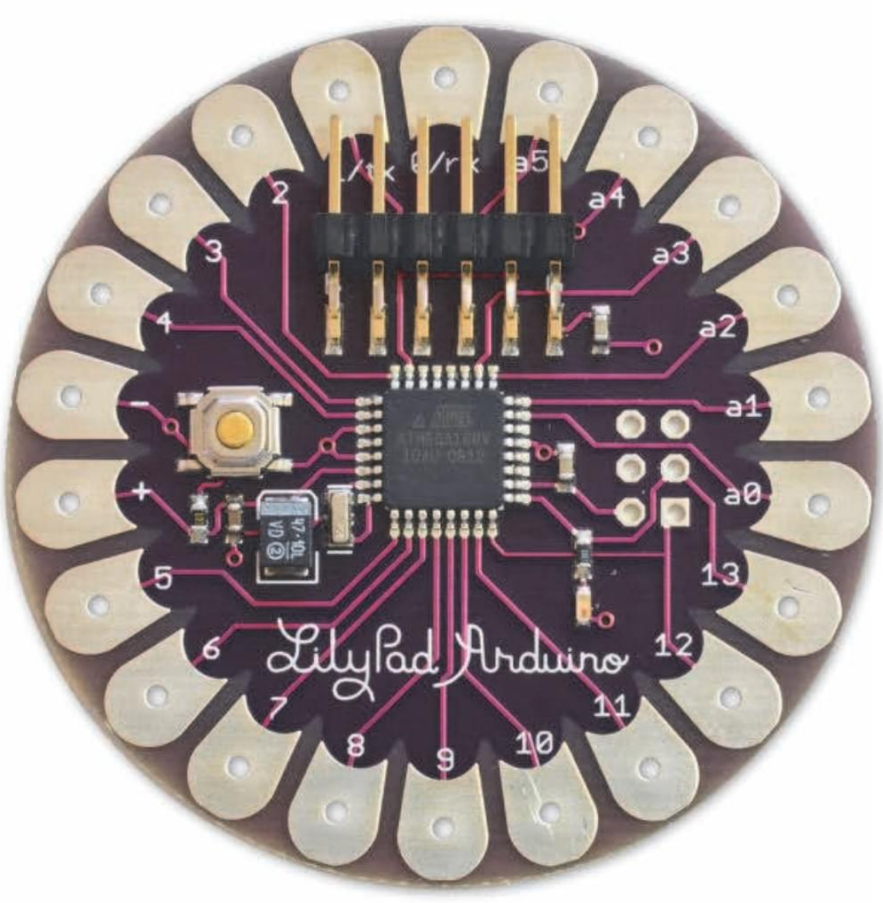


La Arduino Fio, por sus pequeñas dimensiones, es ideal para proyectos que deban utilizar espacios reducidos. Aun así, se trata de una placa Arduino completamente funcional.

Arduino LilyPad

Como veremos con esta placa Arduino, las capacidades de esta plataforma alcanzan límites insospechados. LilyPad es una placa diseñada para ser integrada en prendas de vestir o textiles, es decir, se trata de una versión de Arduino que podemos usar.

Su arquitectura se basa en dos microcontroladores de bajo consumo distintos, Atmega168V y ATmega328V que trabajan a 8 Mhz, a 2,7V y a 5,5V, respectivamente. Posee 14 pines digitales y 6 analógicos, una memoria flash de 16 kb, 1 kb de SRAM y 512 bytes de EEPROM.

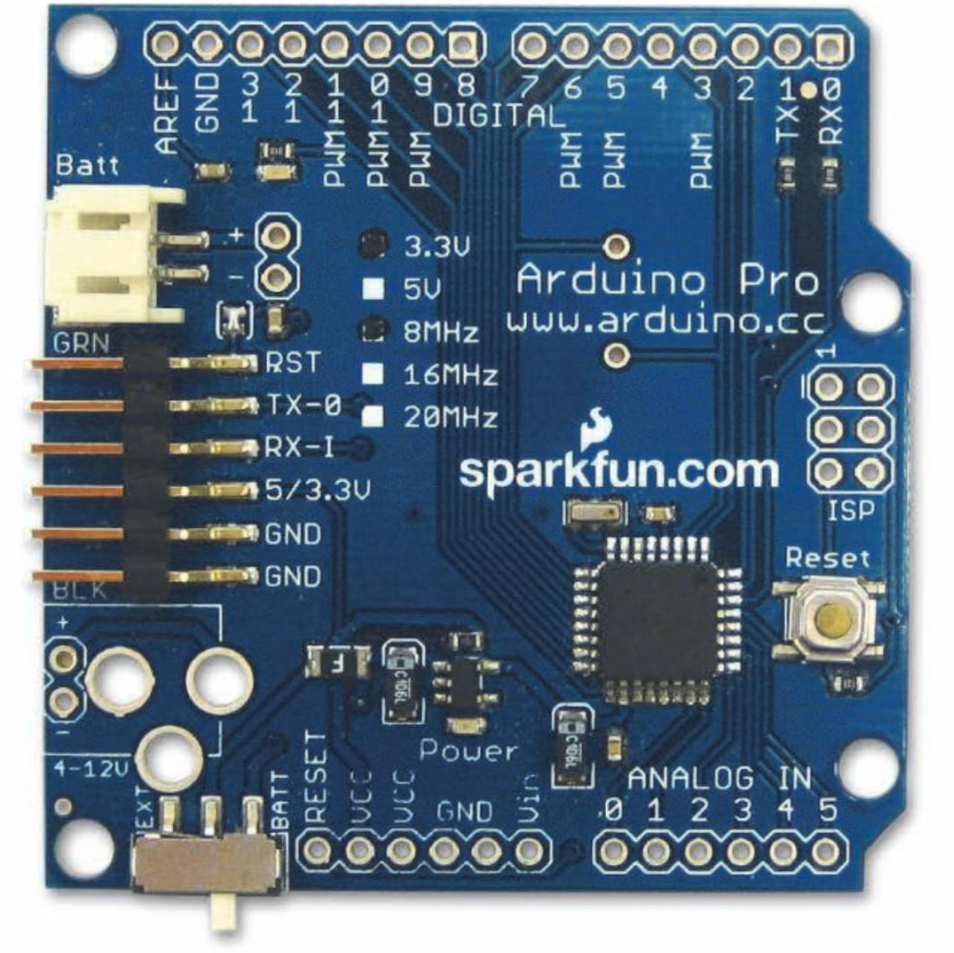


Arduino LilyPad posee una forma distinta, especial para ser incorporada en prendas o artículos que podemos utilizar a diario. Se usa en proyectos de tejidos inteligentes.

Arduino Pro

La placa Arduino Pro utiliza un microcontrolador ATmega168 o un Atmega328, con 8 Mhz o 16 Mhz. Nos ofrece un total de 14 pines de E/S digitales y 6 pines analógicos. Se trata de una placa que posee entre 32 kb y 16 kb de memoria flash, dependiendo del controlador que utilice. Ambos modelos ofrecen 512 bytes de EEPROM.

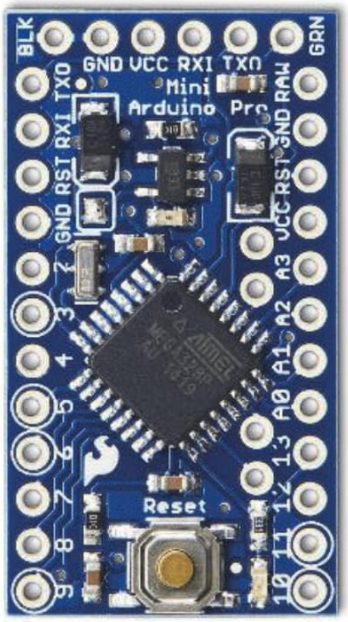
Su nombre puede confundirnos, pero la verdad es que no es una de las placas más potentes de la familia Arduino, aunque resulta una excelente opción para quienes buscan potencia combinada con un bajo costo.



La placa Arduino Pro posee pines laterales de conexión del UART, regulador 5V incorporado, y se encuentra protegida contra inversión de polaridad.

Arduino Pro Mini

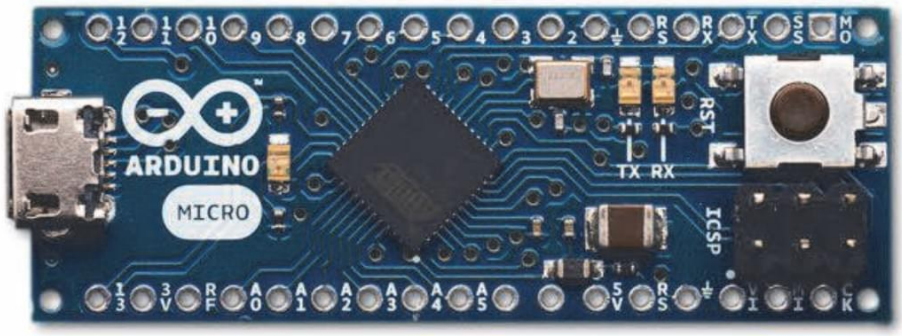
Se trata de la versión mínima de la placa Arduino Pro. Con un precio menor y un tamaño bastante reducido, nos aporta flexibilidad y, por lo tanto, portabilidad para proyectos específicos. Debemos tener en cuenta que, para reducir su tamaño, se ha prescindido de características tales como el conector USB integrado o los conectores de pin.



La placa Arduino Pro Mini posee un tamaño muy reducido, pero con características generales similares a su hermana mayor, la placa Arduino Pro.

Arduino Micro

La placa Arduino Micro ofrece una elevada autonomía junto a un tamaño muy reducido. Su construcción se basa en un microcontrolador ATmega32u4, que funciona a 16 Mhz. Posee un total de 20 pines digitales y 12 pines analógicos. Si queremos compararla, podemos decir que es similar a Arduino Leonardo, aunque agrega la capacidad de comunicación USB built-in, por lo tanto, no precisa de la existencia de un segundo procesador.

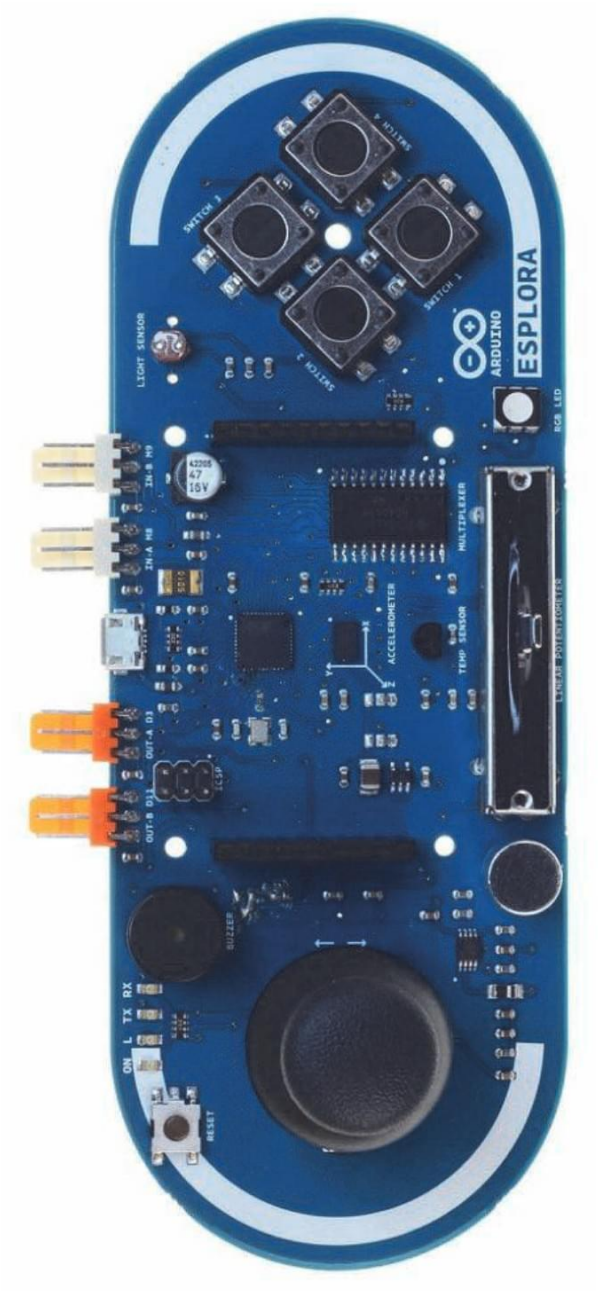


Arduino Micro nos ofrece una potencia similar a la placa Arduino Leonardo, pero en un formato bastante compacto de tan solo 48 x 18 mm.

Arduino Esplora

Es una placa que, a primera vista, llama la atención por su tamaño y también por su forma, pero, si la analizamos más a fondo, veremos que posee otras características que nos sorprenderán. Funciona con un microcontrolador ATmega32u4 que trabaja a 16 MHz, posee una SRAM de 2,5 kb, 1 kb de EEPROM, una memoria flash de 32 kb con 4 kb para el bootloader.

Entre las novedades que presenta, encontramos la integración de diversos sensores: acelerómetro, temperatura, luz; también nos ofrece un zumbador, botones, joystick, micrófono y, por si todo esto fuera poco, se agrega un socket adecuado para conectar una pantalla TFT LCD.



Arduino Esplora incorpora una serie de sensores y, también, una forma distinta de las demás placas Arduino.

Placas no oficiales

Ya mencionamos que cualquier placa que haya sido creada para ser compatible con Arduino o basada en su arquitectura, pero que no pueda utilizar el nombre oficial se considera una placa no oficial. En este segmento existen muchas opciones, algunas de ellas son compatibles a nivel de hardware, por lo que permiten utilizar los shields oficiales, mientras que otras solo son compatibles a nivel de software, por lo que únicamente podremos utilizar el IDE para programarlas. La elección de una o de otra placa dependerá de las necesidades de nuestros proyectos.

Mencionaremos algunas de las placas no oficiales existentes.

PLACAS NO OFICIALES DISPONIBLES EN EL MERCADO

PLACA	EMPRESA	DESCRIPCIÓN
Almond PCB	Open Bionics	Ofrece un microcontrolador Atmega 2560, 9 pines E/S configurables digitales, 2 pines ADC, 256 kb de flash, 4 kb de EEPROM, USB, I2C, UART, SPI, entre otras características.
Banguino	Dimitech	Posee un microcontrolador ATmega328, es compatible a nivel de software, pero no a nivel físico. En cuanto a características, es similar a Arduino UNO.
Boarduino	Adafruit	Es una placa compatible solo a nivel de software con Arduino. Sus características son similares a un Arduino Diecimila, pero con un tamaño y un valor más reducido.
bq ZUM BT-328	bq	Es similar a Arduino UNO, pero incluye un set de tres pines para conectar sin empalmes, botón de encendido y apagado, Bluetooth y soporte de más conexiones por 3.2ª, además posee una conexión micro USB.
ChibiDuino2	TiisaiDipJp	Esta placa es compatible con Arduino UNO, pero incluye dos mini-USB B, un puerto LCD 1602 y un área breadboard.
ChipKIT Lenny	Majenko Technologies	Es compatible con Arduino Leonardo a nivel físico. Posee un microchip PIC32MX270F256D a 40 Mhz, 256 kb de flash, 64 kb de RAM.
Freaduino	Elecfreaks	Su creación se basa en la placa Arduino UNO, y es compatible en un 10% a nivel de hardware y de software.

USOS DE ARDUINO

Ya conocimos algunas de las placas Arduino oficiales disponibles en el mercado y sus características generales, también realizamos un repaso por las placas no oficiales. En este punto, gracias a la información de las diferentes placas, somos capaces de detallar algunos de los principales usos en los que podemos explotar su potencial.

La versatilidad de Arduino hace casi imposible describir en detalle todo lo que podemos lograr, pero en pocas palabras diremos que, gracias a Arduino, es posible hacer prácticamente de todo, el límite es nuestra imaginación. Algunas de sus aplicaciones son las siguientes:

AUTOMATIZACIÓN INDUSTRIAL

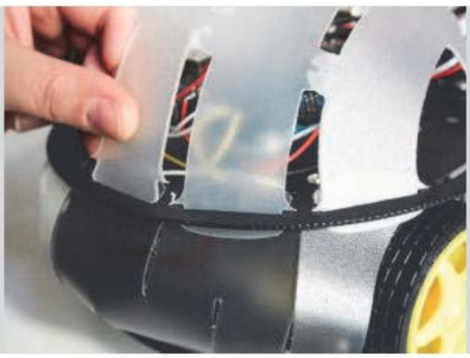
Arduino ofrece muchas posibilidades y ha sido parte importante en proyectos de automatización en industrias, gracias a que puede funcionar como un controlador lógico programable.

DOMÓTICA

Los sistemas de control de domótica son más accesibles y pueden ser creados por entusiastas, gracias a las placas proporcionadas por Arduino.

PROTOTIPADO

La construcción de prototipos para diversos proyectos se ha visto beneficiada gracias a la rapidez que nos ofrece el trabajo con Arduino.

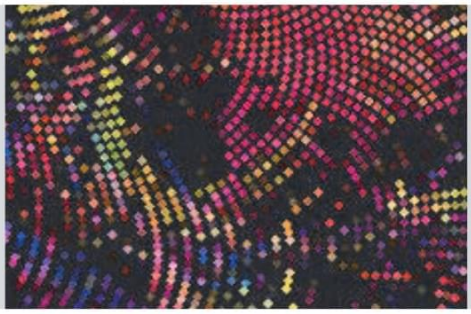


ENTRENAMIENTO ELECTRÓNICO

Arduino es una plataforma excelente para utilizar en centros educativos, ya sea para introducir conceptos electrónicos o para crear proyectos iniciales sencillos.

ARTE

El arte no es una esfera que se aleje de la presencia de Arduino, existen muchas aplicaciones de esta plataforma en la creación de arte.

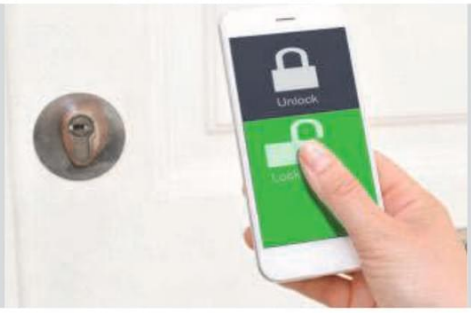


EFICIENCIA ENERGÉTICA

Desde el control del consumo energético en nuestro hogar hasta medidores de consumo, todo lo podemos crear con el apoyo de Arduino.

MONITORIZACIÓN

Arduino ha sido parte de proyectos dedicados a la monitorización a distancia. Ya sea de temperatura, nivel de agua o también de espacios físicos, todo puede ser monitorizado por Arduino.



ADQUISICIÓN DE DATOS

Arduino es capaz de adquirir diversos tipos de datos, para ello podemos agregar a nuestros proyectos algunos de los sensores disponibles en el mercado.

ROBÓTICA

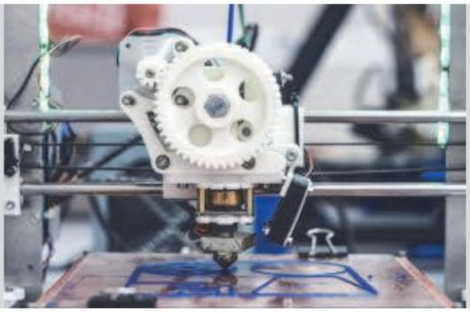
Por supuesto, la robótica no podía quedar fuera de este pequeño listado. Encontramos diversos proyectos que hacen uso de Arduino para construir y programar robots.

INTERNET DE LAS COSAS

Más conocido como IoT, por sus siglas en inglés, se trata de lograr la interconexión de objetos cotidianos con internet.

OTRAS APLICACIONES

Los usos de Arduino son muchos. Algunos sectores donde se ha destacado con proyectos interesantes son la construcción de drones y la de impresoras 3D.



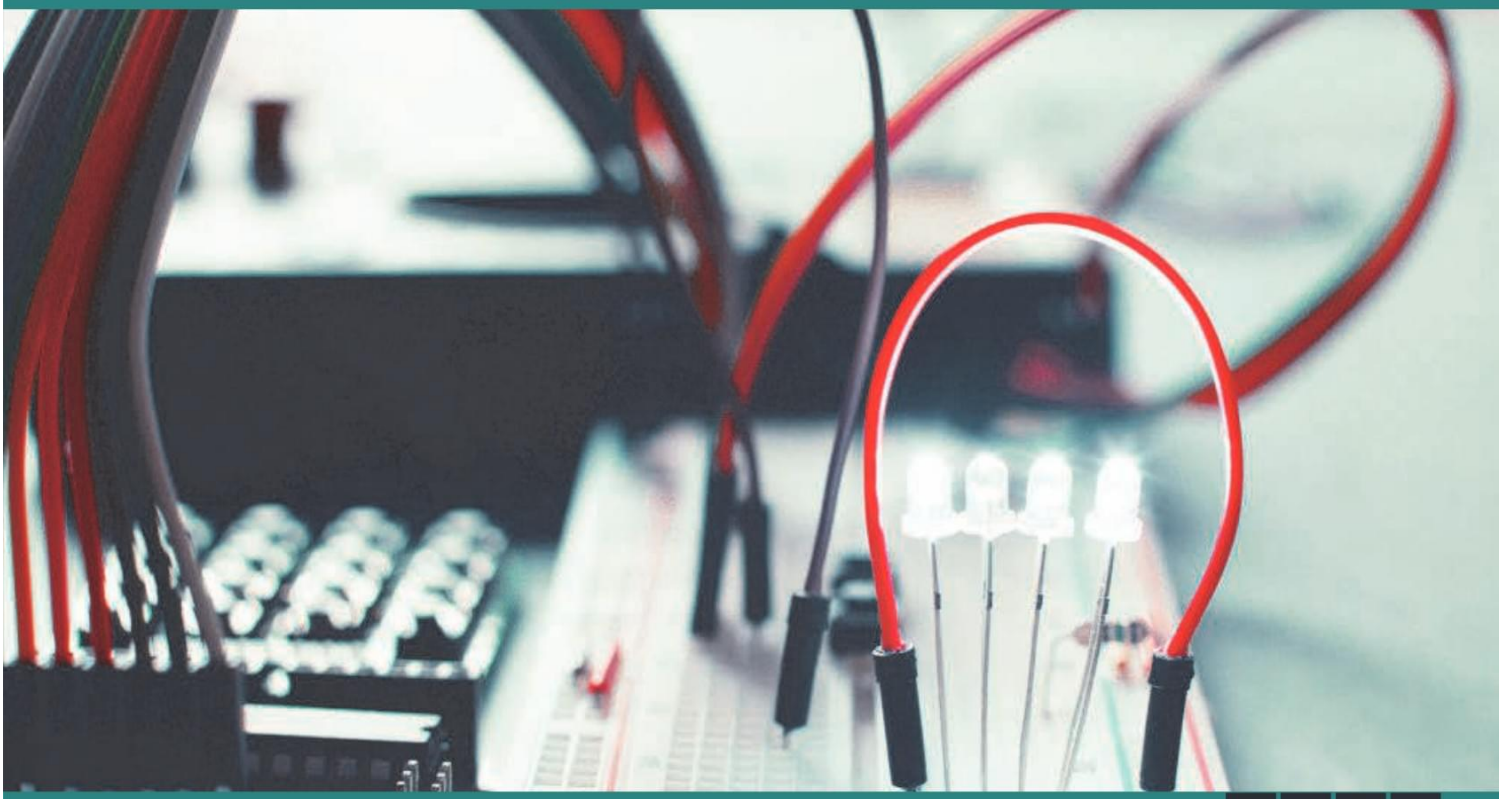
Arduino Robot consta de dos placas circulares apoyadas en ruedas para entregar movilidad.

Resumen Capítulo 02

En este capítulo construimos una definición satisfactoria de Arduino, conocimos sus principales características y, de esta forma, pudimos comprender sus ventajas. Caracterizamos a Arduino a nivel de hardware, a nivel de software y, también, mencionamos la importancia de la comunidad que está a su alrededor. Conocimos qué es y para qué sirve el hardware libre y reconocimos a Arduino como un representante de esta filosofía. Más adelante presentamos y caracterizamos algunas de las placas oficiales Arduino más utilizadas, y conocimos varias de las placas compatibles (no oficiales) que han sido creadas por otras empresas. Para terminar, entregamos un listado de posibles aplicaciones en las que se ha utilizado o se puede utilizar una placa Arduino.

¿Qué se necesita?

03

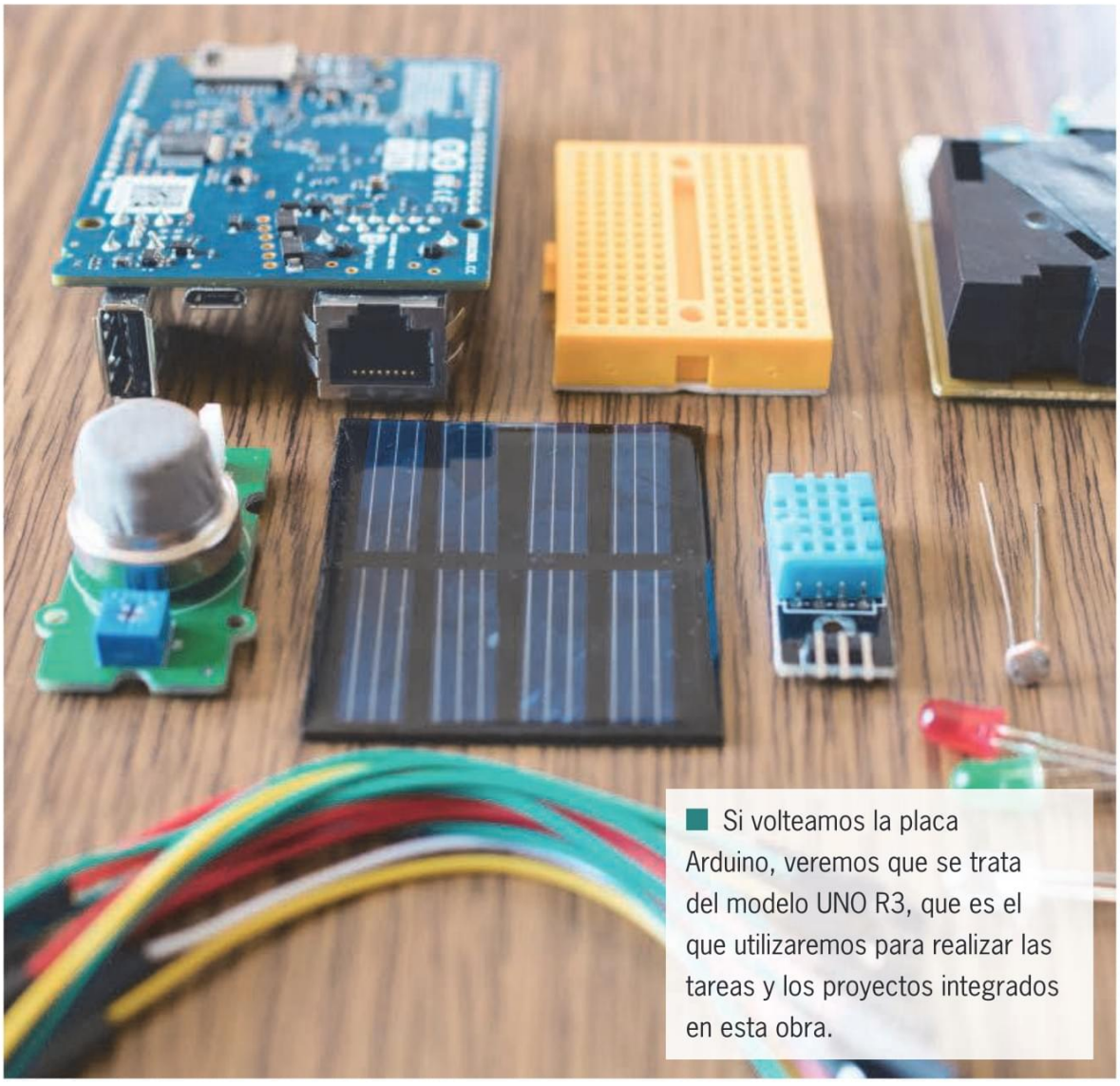


Ya conocimos algunas de las placas Arduino oficiales y también las no oficiales. Ahora es el momento de caracterizar con mayor profundidad a Arduino UNO, la placa que utilizaremos en nuestros primeros proyectos, así como también aquellos componentes básicos que usaremos para iniciarnos en el mundo de Arduino.

COMPONENTES NECESARIOS

Para comenzar a trabajar con Arduino necesitamos, en primer lugar, una placa de pruebas. Podemos utilizar cualquiera de la gama de placas oficiales o, también, una compatible. En este caso usaremos Arduino UNO, por ser la placa más generalizada.

Pero Arduino UNO no puede trabajar sola, es necesario contar con una serie de componentes electrónicos adicionales, cada uno de los que nos servirá para una tarea específica y, en conjunto, para lograr proyectos completos. Para analizar en detalle lo que utilizaremos, los dividiremos en **placa de desarrollo** y **elementos adicionales**.

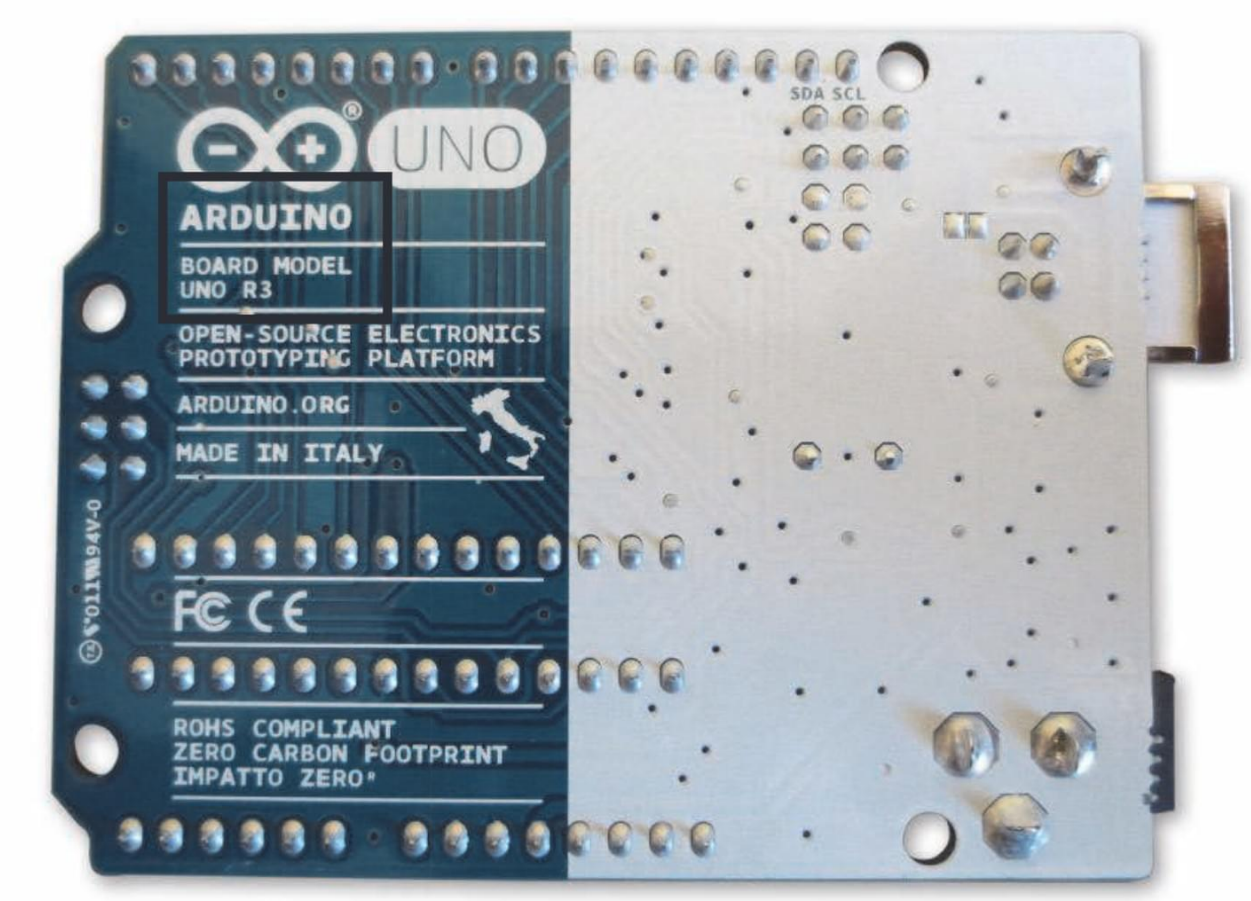


Si volteamos la placa Arduino, veremos que se trata del modelo UNO R3, que es el que utilizaremos para realizar las tareas y los proyectos integrados en esta obra.

PLACA DE DESARROLLO

Como mencionamos, para iniciarnos en Arduino utilizaremos la placa Arduino UNO. Esta es muy versátil y nos ofrece todo lo que necesitamos para proyectos tanto sencillos como avanzados.

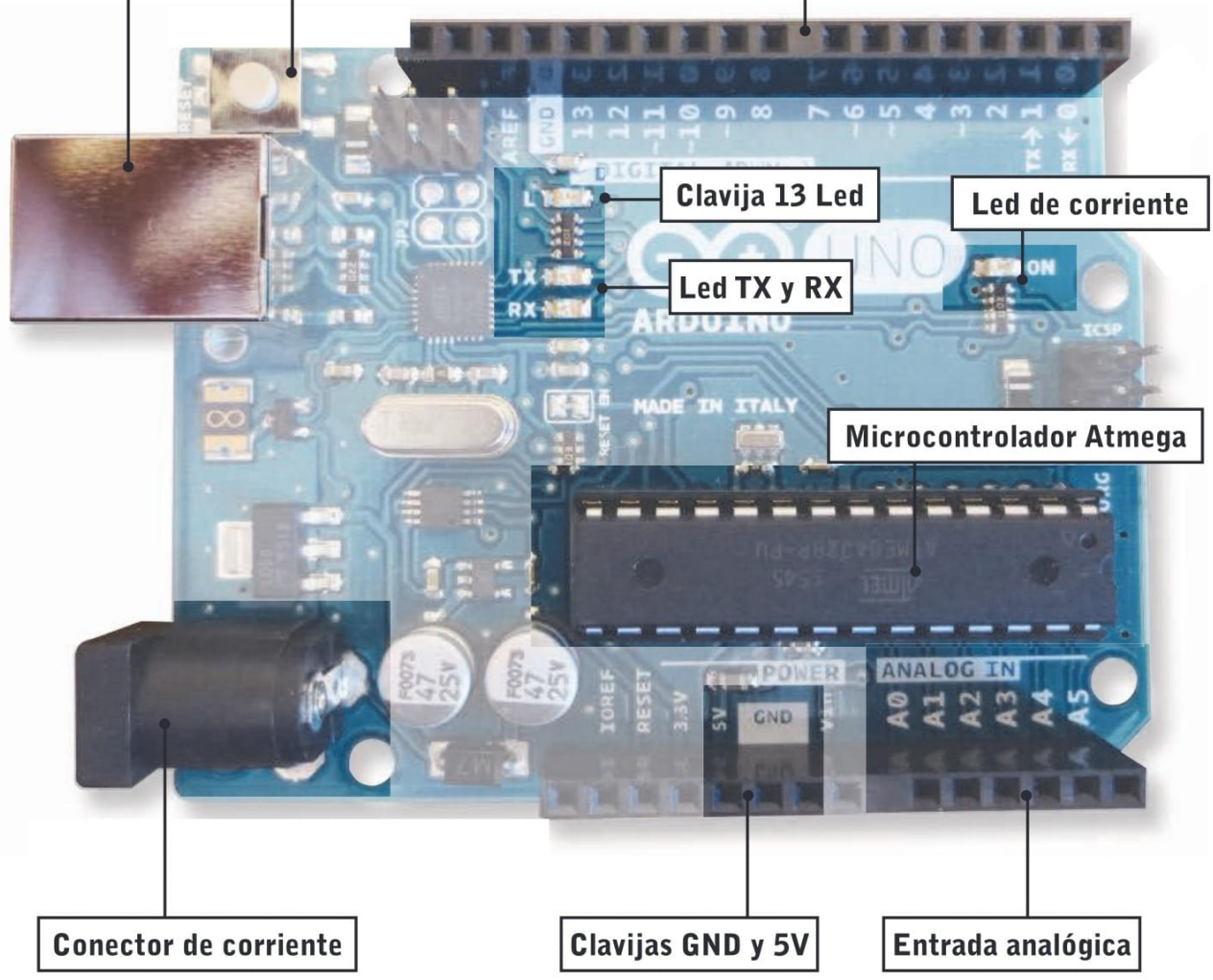
En la actualidad, la última versión de esta placa es la **R3**, por lo tanto, trabajaremos con ella.



Si volteamos la placa Arduino, veremos que se trata del modelo UNO R3, que es el que utilizaremos para realizar las tareas y los proyectos integrados en esta obra.

En Arduino UNO encontraremos diferentes secciones y componentes que es necesario conocer. Tomaremos la placa y la ubicaremos teniendo en cuenta la posición correcta del logo de Arduino, de esta forma, describiremos los componentes en orden. En la siguiente **Guía visual** analizaremos los elementos más relevantes.

Guía visual: Arduino UNO



Puerto USB Se trata de un puerto USB tal como el que encontramos en otros dispositivos, como una impresora. Este lo utilizaremos para entregar energía a la placa Arduino mientras estamos trabajando con la PC. También es necesario para cargar los bocetos o sketchs, en definitiva, se trata de la forma de comunicación de la placa con la computadora.

Botón de reinicio Es un pequeño botón que sobresale de la placa Arduino, su función es permitirnos resetear el microcontrolador ATmega, de esta forma eliminaremos lo que hayamos cargado y podremos comenzar con un nuevo proyecto o sketch. Resulta bastante útil y siempre lo debemos tener en cuenta pues, al principio, podemos equivocarnos bastante mientras cargamos los bocetos.

Led TX y RX Se trata de LEDs que están perfectamente indicados en la placa Arduino, se utilizan para verificar que existe comunicación entre la placa y la computadora. La forma en que verificaremos la comunicación es esperando que parpadeen mientras cargamos el código que hemos generado en el IDE de Arduino o cuando se efectúa una comunicación en serie.

Clavija 13 Led Es un activador que se presenta en forma predeterminada en Arduino UNO, está indicado con la letra L impresa en la placa. Más adelante lo utilizaremos para efectuar la primera comunicación con nuestra placa de desarrollo.

Clavijas digitales Se trata del conjunto de clavijas digitales que ofrece Arduino UNO. Pueden ser utilizadas, por ejemplo, para **digitalRead**() o **analogWrite**(), entre otras opciones.

Led de corriente Este LED se encuentra marcado con el texto ON. Cuando la placa está recibiendo corriente, por ejemplo desde la computadora a través del puerto USB, se encenderá una luz de color verde. Podemos utilizar este LED para verificar que la placa recibe energía en forma correcta.

Microcontrolador Atmega Sin duda, se considera el corazón de nuestra placa Arduino UNO. Es un microcontrolador creado por ATmel, para el caso de esta placa se trata del controlador Atmega328P-PU.

Entrada analógica Se trata del conjunto de clavijas que funcionan como entradas analógicas, presentes en la placa Arduino UNO. Podemos utilizarlas con **analogRead()**.

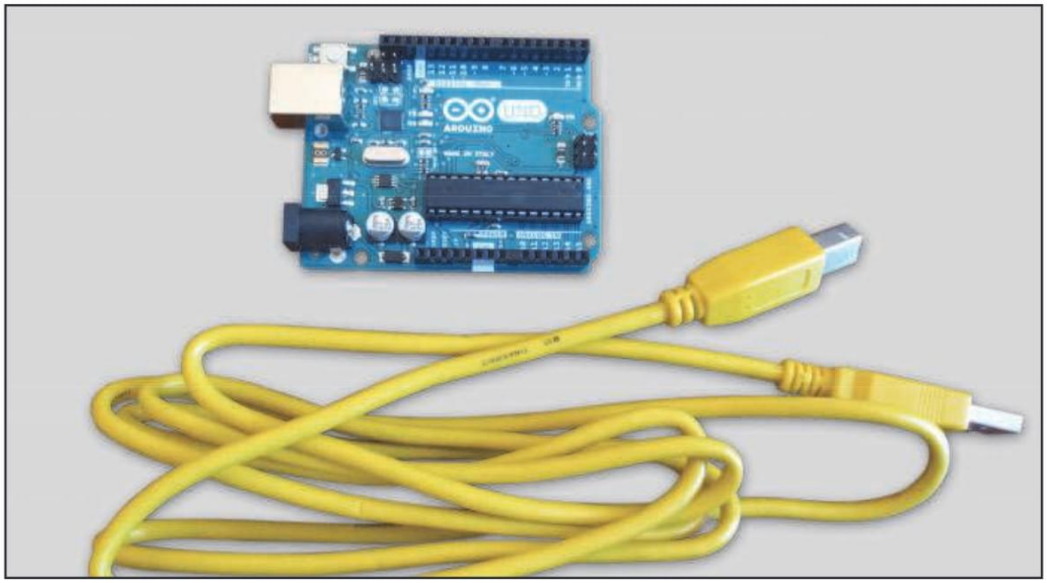
Clavijas GND y 5V Estas clavijas son adecuadas para otorgar, a los circuitos en los que trabajemos, corriente de +5V y también una toma de tierra.

Conector de corriente En el primer punto conocimos una forma de energizar nuestra placa Arduino, mediante el puerto USB. Pero este puerto solo proporcionará energía a la placa mientras la mantengamos conectada a la computadora. Cuando esto no suceda, utilizaremos el conector de corriente para energizar la placa; este conector puede trabajar con voltajes que van desde los 7V hasta los 12V.

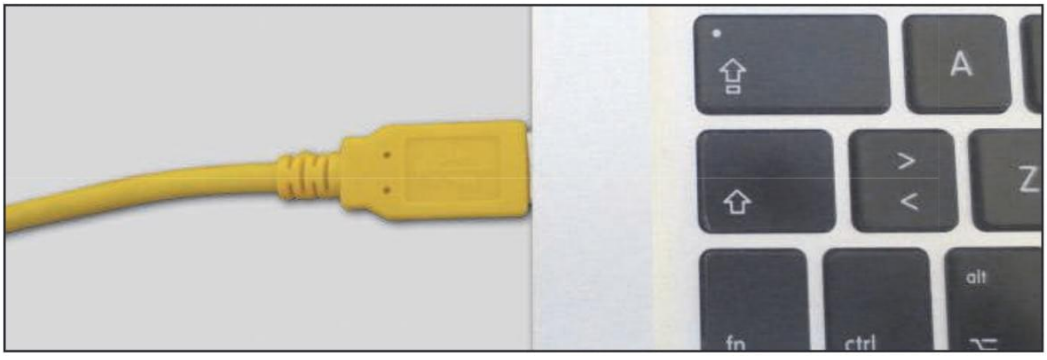
Con los principales elementos de la placa Arduino UNO ya descriptos, podemos realizar el primer test de funcionamiento. Para eso, conectamos la placa a la computadora y verificamos que se le proporcione corriente. Tal como mencionamos en la **Guía visual**, utilizaremos algunos de los LEDs que existen en la placa.

Paso a paso: Prueba de conexión con Arduino UNO

01 Para realizar esta primera prueba de conexión, necesitará una computadora, una placa Arduino UNO y el cable USB que se proporciona con la placa o que se adquiere por separado.

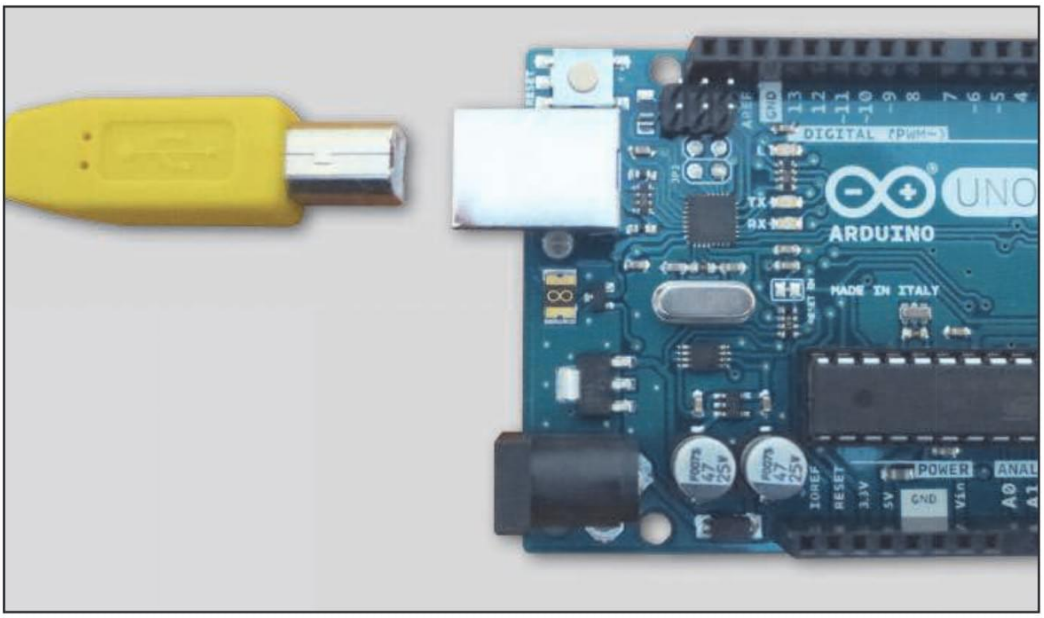


02 En primer lugar, conecte un extremo del cable USB a la computadora, utilizando uno de los puertos USB habilitados.



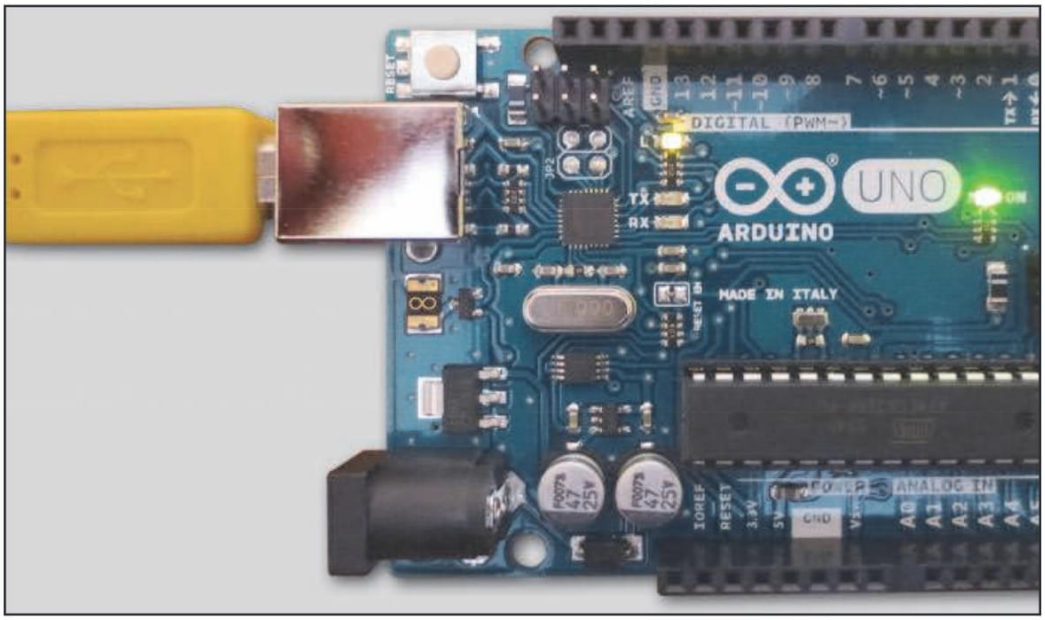
03

Luego, conecte el siguiente extremo a la placa Arduino UNO. Para ello use el puerto USB que se describe en la Guía visual.



04

Con la placa conectada a la PC, verifique el LED de corriente y la clavija 13 LED. Si la conexión y el paso de corriente son correctos, el LED de corriente, indicado en la Guía visual, deberá mostrar una luz constante de color verde, mientras que la clavija 13 LED, indicada en la Guía visual, mostrará una luz parpadeante de color anaranjado.



Funcionamiento

Ahora es momento de profundizar aún más en las características de la placa Arduino que utilizaremos:

- **Microcontrolador: ATmega328**
- **Voltaje: 5V**
- **Voltaje de entrada recomendado: entre 7V y 12V**
- **Límite de voltaje de entrada: entre 6V y 20V**
- **Pines digitales I/O: 14, 6 de ellos son salida PWM.**
- **Entradas analógicas: 6**
- **Memoria flash: 32 KB (ATmega328), de ellos 0.5 kb son usados para el arranque.**
- **SRAM: 2 kb (ATmega328)**
- **EEPROM: 1 kb (ATmega328)**
- **Velocidad de reloj: 16 MHz**

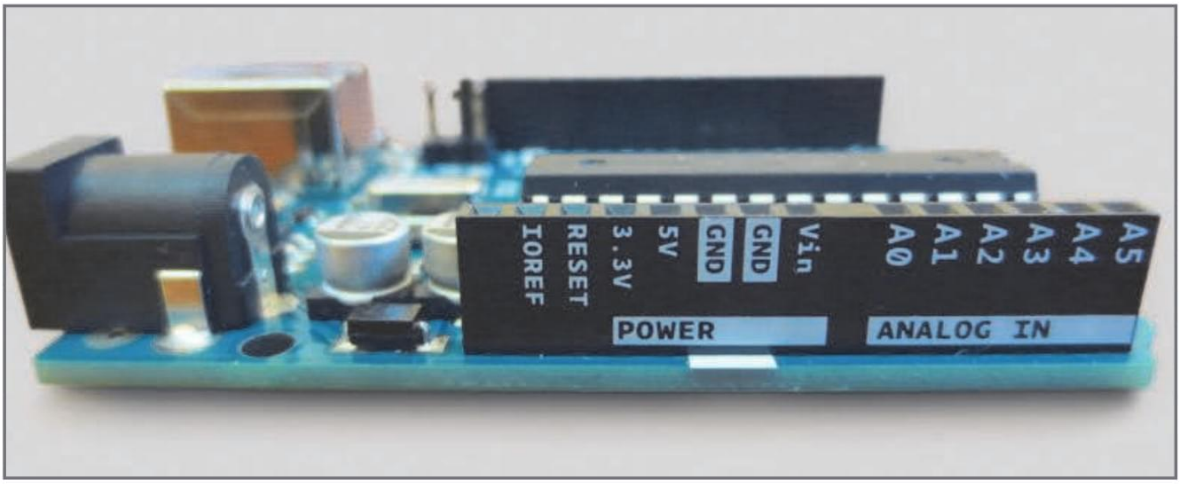
Las placas Arduino nos ofrecen un conjunto de terminales digitales que podemos aprovechar como entradas y salidas generales mediante comandos tales como **pinMode()** o **digitalRead()**, entre otros. Estos terminales poseen una resistencia que puede activarse con **digitalWrite()** al estar configurado como entrada. En la siguiente tabla, conoceremos los terminales digitales generales de una placa Arduino y sus usos.

Pero no solo existen pines digitales en la placa Arduino, también encontramos otros, que mencionamos a continuación:

- **Pines analógicos**: estos pines soportan la conversión analógico-digital (ADC) de 10 bits mediante analogRead().
- **I2C**: permiten la comunicación I2C (TWI) gracias a la librería Wire.
- **VIN**: mediante este pin, proporcionaremos voltaje a la placa Arduino. Debemos tener en cuenta que las diferentes placas soportan distintos rangos de voltaje de entrada por lo que, al utilizarlo para alimentar la placa, debemos ser cuidadosos.
- **5V**: es posible proporcionar alimentación regulada para alimentar el microprocesador y otros componentes.
- **3V3**: se trata de una fuente de 3.3V generada por el chip FDTI.
- **GND**: corresponden a los pines de tierra.
- **AREF**: se trata de la referencia de voltaje para las conexiones analógicas.
- **Reset**: podemos ponerlo en LOW para resetear el microcontrolador.

PINES DIGITALES EN PLACAS ARDUINO

TERMINALES	NÚMEROS	DESCRIPCIÓN
Serial	0 RX y 1 TX	Se utilizan para recibir (RX) y transmitir (TX) datos serie TTL.
Interruptores externos	2 y 3	Podemos utilizar estos terminales para disparar una interrupción con un valor bajo, un pulso de subida o bajada, y también un cambio de valor.
PWM	3, 5, 6, 9, 10 y 11	Nos entrega salidas PWM de 8 bits; para ello utilizamos la función analogWrite(). Con el microchip Atmega8, estas salidas estarán en los pines 9, 10 y 11.
Reset BT	7	En Arduino BT, se encuentra conectado a la línea de reset para el módulo Bluetooth.
SPI	10, 11, 12, 13	Se trata de terminales que soportan comunicación SPI.
LED	13	En algunas placas, un LED se encuentra conectado al pin 13.



Al examinar una placa Arduino UNO, verificaremos que el nombre de cada pin se encuentra indicado en el costado de los conjuntos de entradas.

Comparación con otras placas

Aunque ya elegimos a Arduino UNO como la indicada para comenzar a trabajar, es una buena idea compararla con otra placa oficial, así notaremos sus ventajas y, también, sus puntos débiles.

Para efectuar esta comparación hemos elegido a Arduino Leonardo, que ya conocimos en el **capítulo 2** de este libro, porque se trata de placas que, a simple vista, tienen un gran parecido.

Veamos primero las similitudes. En realidad se trata de dos placas que presentan el mismo tamaño y poseen la misma cantidad de pines, que se disponen de igual forma en ambas placas. Estos presentan los mismos requerimientos de alimentación, de 7V a 12V, con 6V y 20V como límites. Además, poseen la misma frecuencia de operación (16 Mhz) e igual voltaje de operación (5V).

Pero no todo son similitudes, analicemos ahora sus principales diferencias.

Microprocesador

Sin duda se trata de la mayor diferencia entre Arduino UNO y Arduino Leonardo. Arduino UNO utiliza un Atmega328 que no ofrece comunicación USB integrada, por lo tanto, debe emplear un microcontrolador adicional, el ATmega 16u2. En cambio, Arduino Leonardo utiliza un microcontrolador ATmega32u4 que sí integra la posibilidad de comunicación USB.

Comunicación I2C

Aunque ambas tarjetas son compatibles con la comunicación I2C o TWIN, existe una diferencia importante entre ellas, se trata de los pines que deben utilizarse para la línea de datos seriales y para la línea de reloj.

Arduino UNO utiliza los pines A4 y A5, respectivamente, mientras que Arduino Leonardo hace uso del pin 2 y del 3.

Puede parecer una diferencia mínima, pero la ubicación diferente de los pines para este tipo de comunicación hace que algunos shields no sean compatibles con las dos tarjetas. En beneficio de Arduino UNO, debemos considerar que la mayor parte de los shields han sido diseñados para ella, por lo tanto, Leonardo puede requerir la realización de algunas modificaciones.

Entradas analógicas y salidas PWM

Como consecuencia del uso de microcontroladores distintos, UNO y Leonardo ofrecen un número diferente de pines, que pueden ser configurados como entradas analógicas y salidas PWM.

Arduino UNO ofrece 6 canales de entrada analógica en los pines del A0 al A5, mientras que Arduino Leonardo posee 12 canales de entrada analógica, del A0 al A5 y del A6 al A11. Además, presenta una salida adicional de PWM en el pin 13.

Interrupciones externas

En la tarjeta Arduino UNO encontramos los pines 2 y 3, para las interrupciones 0 y 1, respectivamente. Arduino Leonardo nos ofrece cinco pines para interrupciones externas: 3, 2, 0, 1 y 7.

Memoria

Según la información oficial sobre ambas placas, también hallamos una pequeña diferencia en el apartado de memoria. En Arduino UNO, el ATmega328 posee 32 kb de memoria flash, con 0.5 kb para el bootloader; también tiene 2 kb de SRAM y 1 kb de EEPROM. Por otra parte, en la Arduino Leonardo, con un microprocesador ATmega32u4, tenemos 32 kb de memoria flash, con 4 kb que son utilizados para el bootloader; 2.5 kb de SRAM y 1 kb de EEPROM.

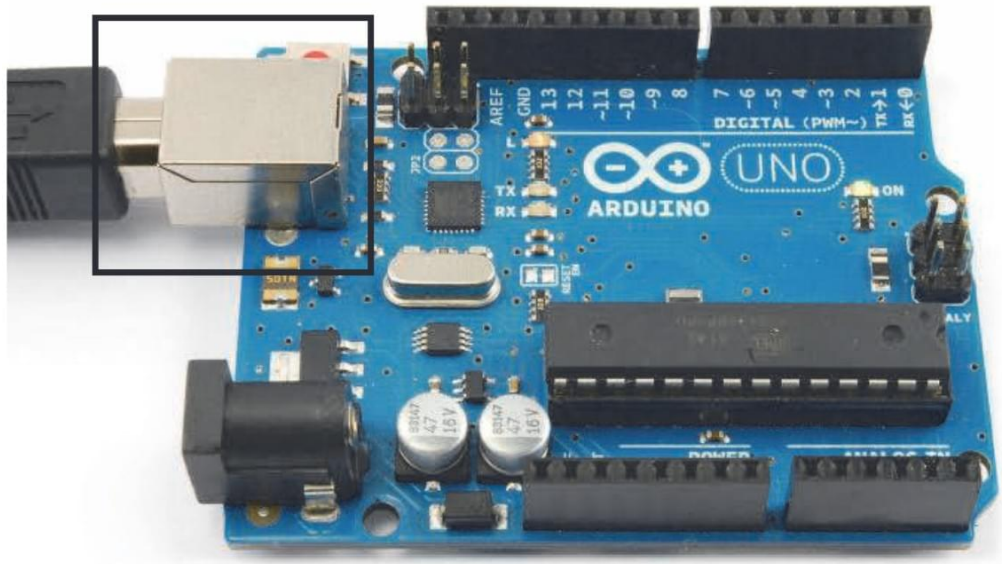
Comunicación SPI

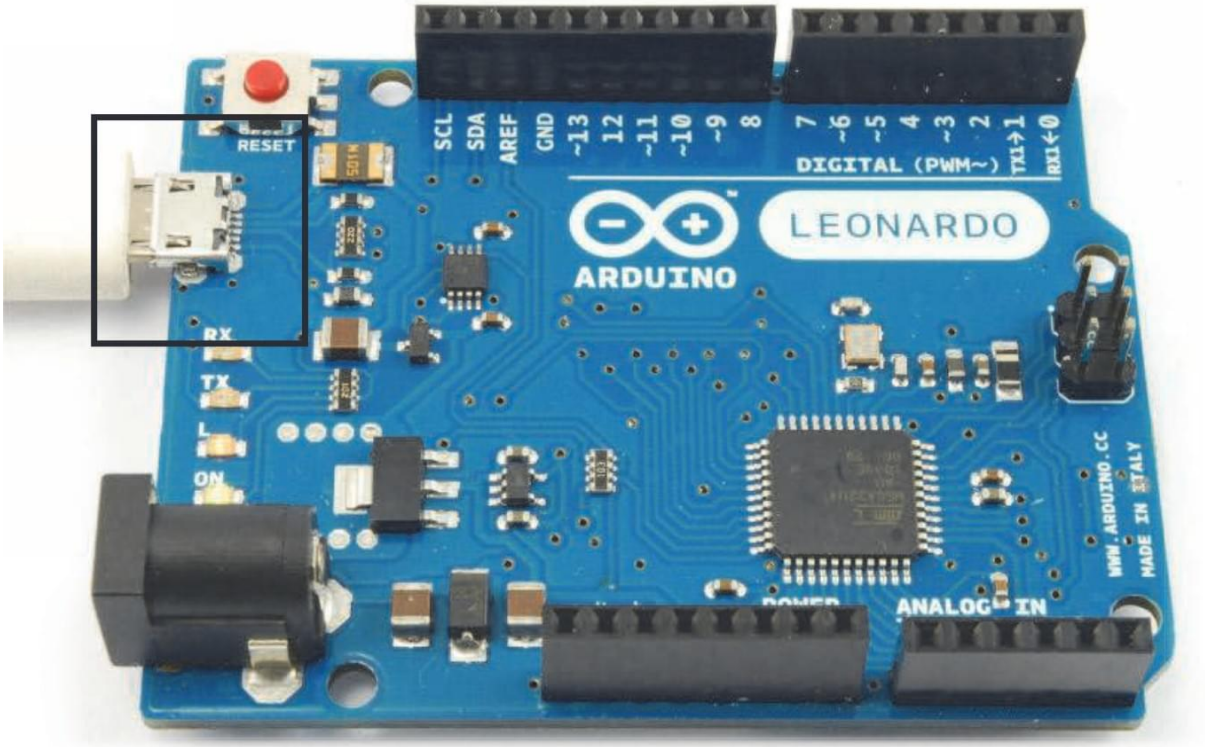
El protocolo de comunicación SPI o interfaz serial periférica es soportado tanto por UNO como por Leonardo. En Arduino UNO se utilizan los pines 0, 11, 12 y 13 para las líneas SS, MOSI, MISO y CK, mientras que en Leonardo se utiliza el conector ICSP, que se encuentra en uno de sus extremos. Al igual que lo que ocurre con la comunicación I2C, encontraremos shields que no son compatibles con la tarjeta Arduino Leonardo.

"La placa Arduino Leonardo utiliza un microcontrolador que integra conectividad USB: el Atmega32U4."

Cable de conexión

Un punto importante entre las diferencias que se encuentran entre UNO y Leonardo es el cable que necesitamos para conectarlo a la computadora. En Arduino UNO es necesario usar un cable A–B, como el que utiliza la mayoría de las impresoras; en cambio, para Leonardo necesitaremos un cable A-micro B, como el que se usa para los teléfonos inteligentes.





■ En la Arduino UNO se utiliza un cable USB A-B mientras que, para conectar Leonardo a la PC, será necesario un cable A–micro B.

ELEMENTOS ADICIONALES

Ya elegimos la placa Arduino UNO como la compañera en nuestros primeros proyectos, ahora es tiempo de detallar los elementos anexos que utilizaremos. Todos estos elementos pueden ser adquiridos en tiendas de electrónica o también en establecimientos especializados; además, si compramos un kit de iniciación a Arduino, seguro obtendremos todo lo necesario. Aunque presentaremos un listado de componentes básico, es posible que se necesiten otros elementos, esto dependerá del tipo de proyecto que deseemos implementar.

Protoboard

El **protoboard** no es más que una placa que utilizaremos para construir circuitos electrónicos. A simple vista, se trata de un panel lleno de agujeros, en el que podremos conectar los cables y componentes electrónicos, por ejemplo, condensadores y diodos emisores de luz.

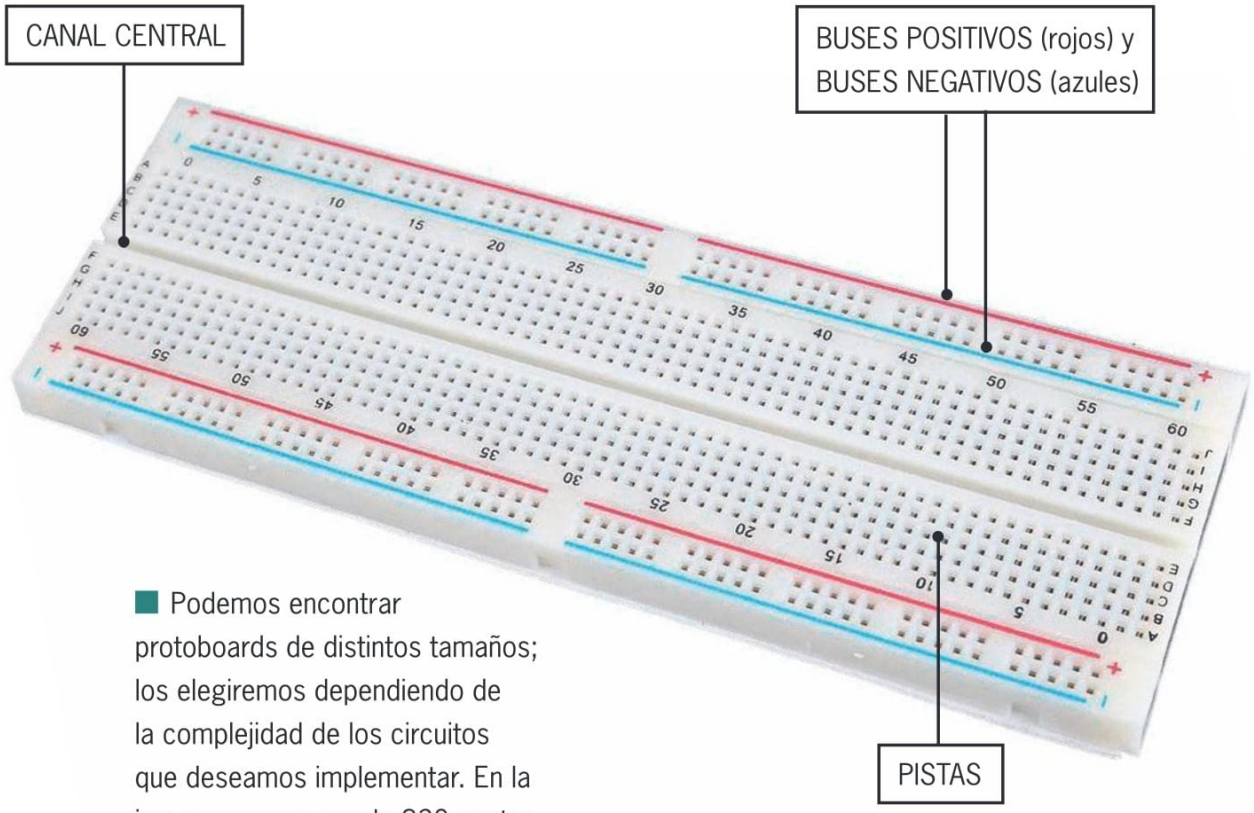
Existen diferentes tipos de protoboards, algunos necesitarán el uso de soldadura mientras que otros se diseñaron para ser usados en modo libre de soldadura.

Por ser una alternativa más sencilla, es recomendable utilizar un protoboard libre de soldadura.

En su construcción, el protoboard funciona como un tablero con orificios que se encuentran conectados eléctricamente entre sí, siguiendo una distribución lineal para que podamos ubicar y conectar diversos componentes en forma sencilla y rápida.

Podemos dividir el protoboard en tres secciones bien delimitadas:

- **BUSES**: son los caminos que se ubican en ambos costados del protoboard. Están marcados con líneas rojas y azules, para indicar los buses positivos o de voltaje, y los buses negativos o de tierra, respectivamente. Por lo general, utilizamos los buses para conectar la fuente de poder.
- **CANAL CENTRAL**: es la ranura o sección ubicada justo en el centro del protoboard, allí podemos colocar los circuitos integrados.
- **PISTAS**: en la parte central, traspasada por el canal central, se ubican las pistas; están indicadas por números y letras, que nos sirven para ubicarlas.



Podemos encontrar protoboards de distintos tamaños; los elegiremos dependiendo de la complejidad de los circuitos que deseamos implementar. En la imagen vemos uno de 830 puntos.

La importancia del protoboard radica en que nos permite montar circuitos o prototipos en forma temporal, es decir, es posible crear un circuito y probar su funcionamiento, para luego desmontarlo y dejar el protoboard listo para el siguiente experimento. Se pueden crear circuitos conectando diversos componentes, por ejemplo, resistencias, condensadores, LEDs, transistores, pulsadores y circuitos integrados, entre otros. Para efectuar las conexiones, utilizaremos cables específicamente diseñados para ser usados en el protoboard; los conoceremos más adelante.

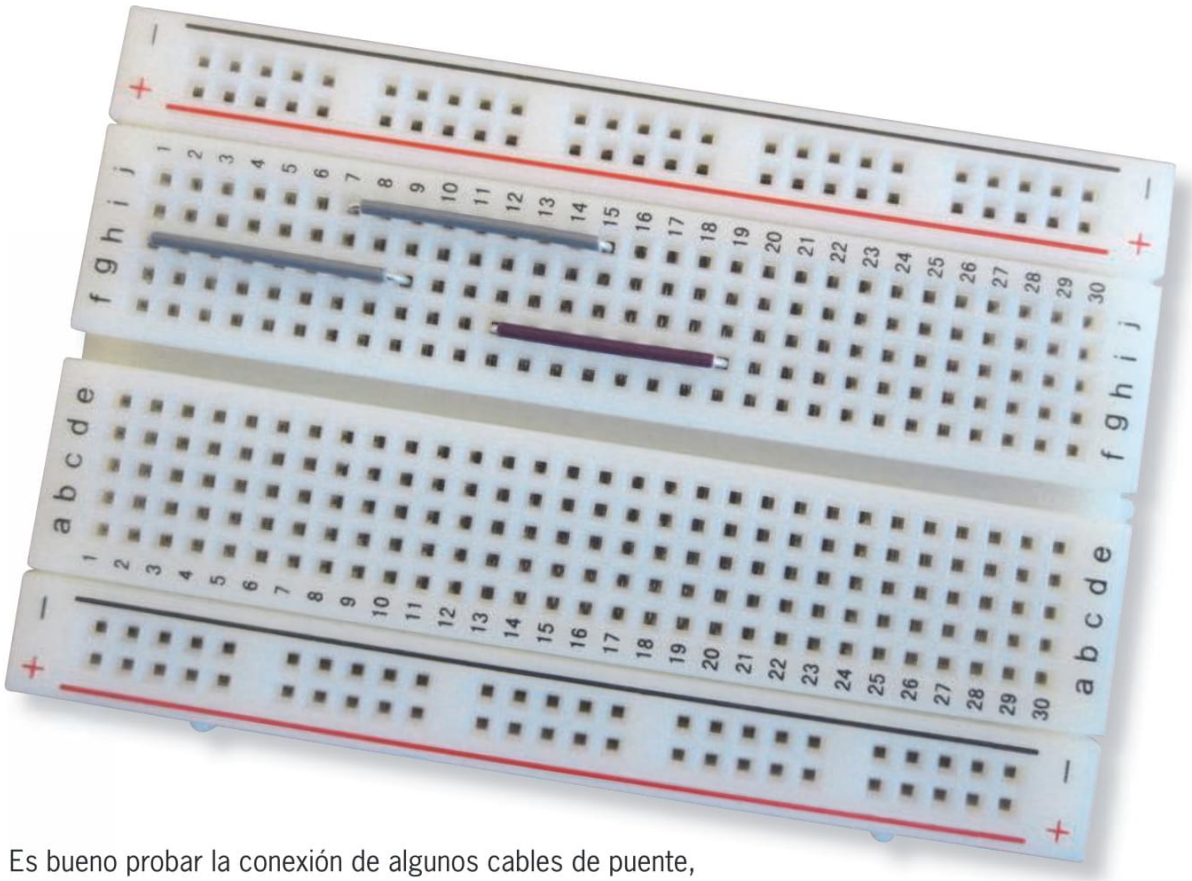
Cables de puente

Para trabajar en proyectos simples y también complejos, necesitaremos un conjunto de **cables de puente**. Se trata de filamentos conductores, recubiertos de material aislante en parte de su extensión, pero que deja libres los extremos, mediante los que realizaremos las conexiones adecuadas mediante conectores terminales.

Estos cables se usan para lograr la conexión de componentes en el protobord y también con la placa Arduino UNO.

Los cables de puente pueden adquirirse en distintas longitudes, ya que la conexión de componentes deberá realizarse a distancias diferentes. Los cables están representados con colores variados, de esta forma, se pueden identificar de manera más rápida y sencilla dentro de un circuito.

Si estamos utilizando un protoboard que no requiere soldadura, solo tendremos que introducir los extremos del cable de puente en los orificios adecuados, dependiendo del proyecto en el que estemos trabajando. Antes de energizar el protoboard, podemos utilizar un par de cables de puente para probar la forma en que debemos conectarlos. Para lograr una conexión correcta, presionamos el cable de puente hasta vencer la resistencia inicial.

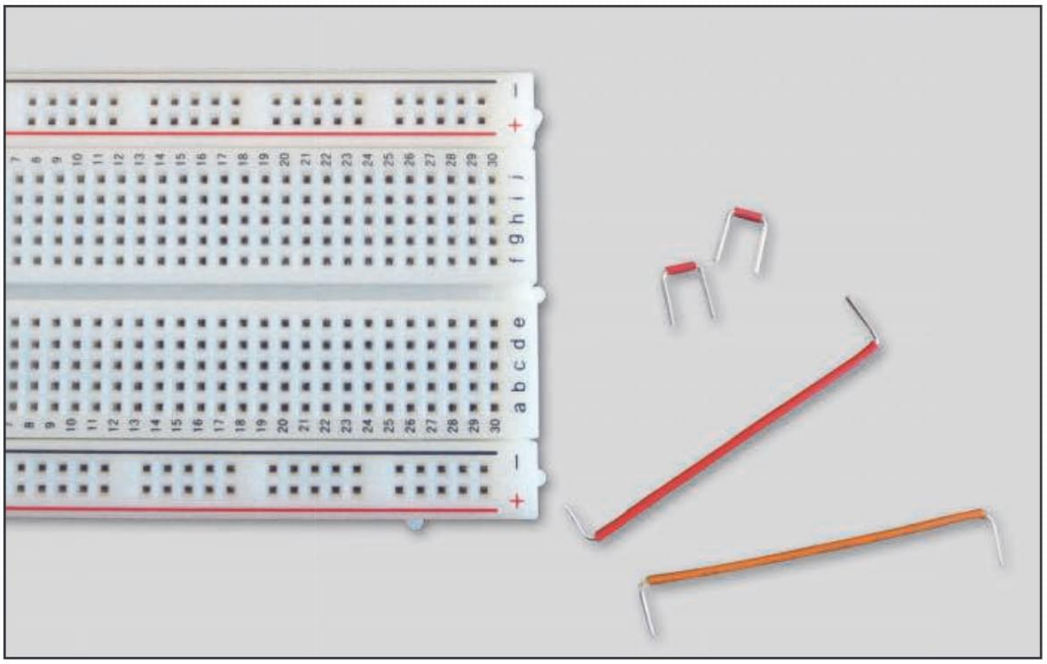


Es bueno probar la conexión de algunos cables de puente, de esta manera, nos acercaremos a la forma de trabajo que implementaremos más adelante y así podamos conocer el uso de estos elementos en el protoboard no energizado.

En este punto, aprenderemos a realizar una conexión básica, para ello utilizaremos los dos elementos que conocemos: el protoboard y los cables de puente. Sigamos las instrucciones indicadas en el siguiente **Paso a paso**.

Paso a paso: Conexiones básicas

01 Para esta actividad básica, necesitará un protoboard y algunos cables de puente, tal como se ve en la imagen.

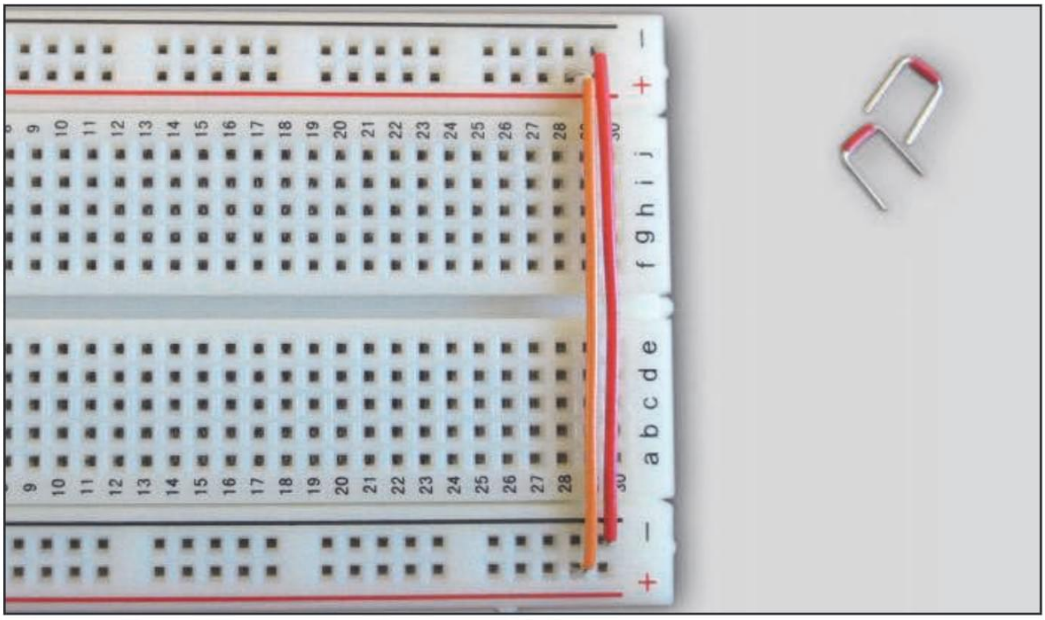


Protoboard no energizado

Es importante recordar que el protoboard no debe estar energizado en el momento de realizar las pruebas de conexión iniciales con los cables de puente. Si trabajamos con la placa energizada y efectuamos una conexión errónea, podríamos dañar la placa al exponerla a la circulación de corriente por caminos no adecuados. Las primeras conexiones solo las realizaremos en forma ilustrativa, para conocer de qué manera debemos conectar los cables de puente; más tarde analizaremos la forma correcta de realizar las conexiones para cada proyecto.

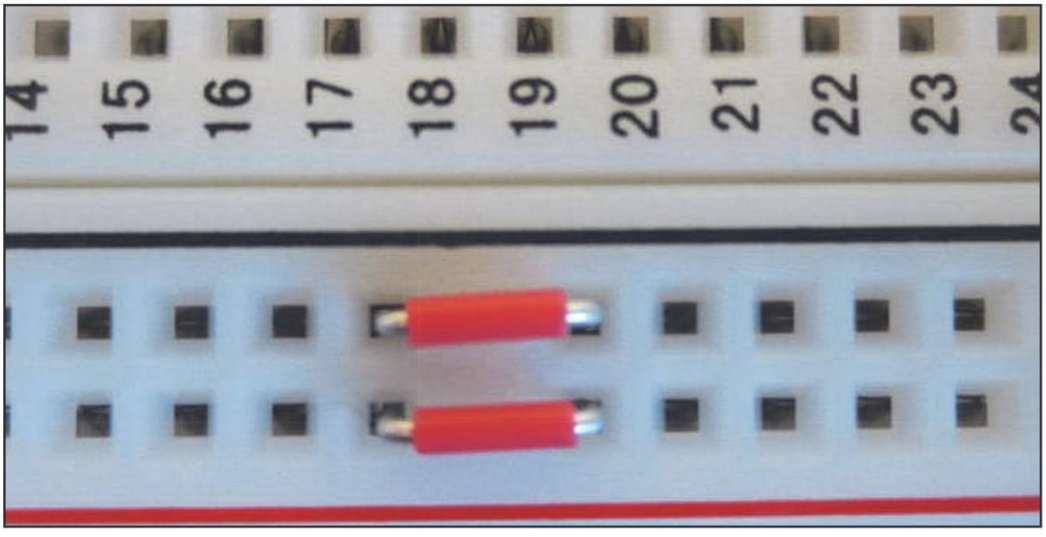
02

En primer lugar, realice una conexión que servirá para que ambos pares de buses sean capaces de conducir corriente cuando les agregue una fuente de poder, de esta forma será más sencillo manipular los circuitos integrados. Conecte ambos buses teniendo en cuenta sus sectores positivos y negativos.



03

En algunos protoboards encontrará que la parte media de sus buses está separada, en estos casos será necesario establecer un puente que permita darle continuidad a la corriente suministrada. En caso de requerirse, debe realizar esta conexión como se aprecia en la imagen de ejemplo.



Condensador

Los **condensadores** son elementos capaces de almacenar y emitir energía en un circuito. Por lo general, son utilizados entre una toma de corriente y la toma a tierra, por ejemplo, cerca de un servomotor; gracias a esto es posible suavizar las fluctuaciones de voltaje.

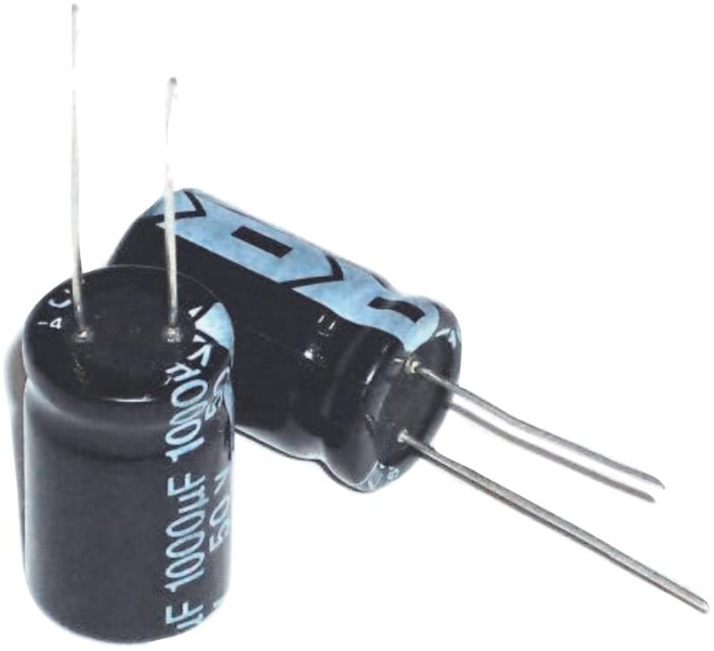
El funcionamiento de un condensador es sencillo, utiliza dos placas o láminas conductoras que se encuentran separadas por un material aislante. Estas láminas se cargarán eléctricamente al conectar una fuente de tensión o batería.

Ambas placas se llenarán con la misma cantidad de carga, pero con diferentes signos, es decir, una tendrá carga positiva mientras que la otra placa tendrá carga negativa. Una vez cargadas, se presenta una tensión entre ellas y pueden soltar la carga almacenada cuando se conecte un receptor de salida.

Entre ambas láminas existe un material aislante o dieléctrico diferente, por ejemplo: tantalio, papel, aire, aluminio, cerámica o algunos plásticos, dependiendo del tipo de condensador del que se trate.

La cantidad de carga que es capaz de almacenar el condensador está definida por la fórmula **C=q/V**.

- **C**: capacidad del condensador.
- **q**: carga de una de las placas.
- **V**: tensión entre las dos placas o la tensión del condensador.



Los condensadores o capacitores electrolíticos ofrecen, por lo general, más capacidad por unidad de volumen, en comparación con otros tipos de condensadores.

Tanto la carga como la descarga de un condensador no se realiza en forma instantánea, por esta razón pueden utilizarse como temporizadores. El tiempo de carga se relaciona con su capacidad y con la resistencia en serie, puesto que esta hace más difícil el paso de la corriente. De la misma forma, su descarga dependerá de la capacidad y de la resistencia de salida.

Existen diversos tipos de condensadores, a continuación, describimos los más utilizados.

VARIABLE

Este tipo de condensador puede cambiar su capacidad en forma mecánica o electrónica. Existen los trimmers, que permiten elegir entre varios valores de capacidad, y también los de sincronización, que presentan una capacidad entre límites establecidos.

ELECTROLÍTICO

Se trata de un condensador que posee una de sus placas formada por líquido iónico. Se destacan por sobre otros condensadores por su mayor capacidad.

CERÁMICO

Este tipo de condensador posee cerámico revestido en láminas metálicas, como material dieléctrico. Gracias a que la constante cerámica es bastante alta, estos condensadores poseen una gran capacidad.

DE PAPEL

Los condensadores de este tipo utilizan papel como elemento dieléctrico. Este papel se encuentra cubierto por cera mineral, aceite sintético o aceite mineral.

PLÁSTICO

Estos utilizan una delgada placa plástica como material dieléctrico; entre ellos se encuentran diferentes clases, dependiendo del tipo de plástico utilizado, por ejemplo, polipropileno, policarbonato, poliestireno, poliéster, teflón, poliparaxileno, entre otros.

Diodo

Gracias a este elemento, nos aseguramos de que la corriente pase en una dirección. Si lo conectamos en una dirección, permitirá que la corriente lo atraviese mientras que, en otra dirección, bloqueará el paso de la corriente.

Si lo observamos, notaremos que se trata de un componente central que es traspasado por un filamento largo. En general, debemos conectar el ánodo al punto de mayor energía en el circuito, mientras que el cátodo se conectará a la toma de tierra o al punto de energía más bajo. ¿Cómo sabemos cuál es el cátodo? Lo reconocemos pues suele estar marcado con una banda indicativa en un lado del cuerpo central. Así, la corriente fluye desde el ánodo hacia el cátodo, es decir, desde el terminal positivo hacia el negativo.

El lado que corresponde al cátodo en un diodo está indicado por una franja en el cuerpo principal del componente.

La principal función de los **diodos** es impedir que la corriente fluya en dos direcciones, por esta razón pueden ser aplicados en los siguientes casos:

FLUJO DE CORRIENTE

Un diodo permite impedir que la corriente fluya en un sentido no deseado, por ejemplo, si queremos que Arduino entregue una tensión sin exponerse a una corriente de entrada, podríamos utilizar un diodo en la placa Arduino y el componente externo.

CORRIENTE TRANSITORIA

Si utilizamos bobinas o inductores, podrían producirse corrientes transitorias, capaces de afectar partes del circuito en el que trabajamos. Para suprimirlas, podemos utilizar un diodo en forma de protección.

CAÍDAS DE VOLTAJE

Es posible utilizar la caída de voltaje que se produce entre el ánodo y el cátodo (0.7 voltios) al conectar un diodo. Esto es útil, por ejemplo, para la conversión análogo/digital.

MEDIA ONDA DE CORRIENTE ALTERNA

Para utilizar la corriente alterna del suministro eléctrico normal, es necesario tratarla de forma especial (transformación de corriente alterna a corriente directa); en este proceso utilizamos el diodo. El diodo es capaz de lograr que la corriente fluya en un solo sentido; como la corriente alterna va en dos sentidos –en el semiciclo positivo va hacia un lado, y en el semiciclo negativo, hacia otro–, el diodo nos permite suprimir una parte de la onda de corriente alterna para lograr una corriente directa que permanece inestable, es decir, que presenta un rizo, pero no alterna su signo.

Diodo emisor de luz (LED)

Al igual que los diodos que conocimos en el apartado anterior, este tipo de componente permite el paso de la electricidad en un solo sentido. En este caso, veremos que el LED se ilumina cuando pasa la corriente.

En su apariencia visual, se destacan dos filamentos unidos a una cabeza de color; por lo general, el ánodo que se conecta a la corriente es el filamento más largo mientras que el cátodo posee una extensión menor.

Una definición correcta de este elemento sería la siguiente: un **diodo LED** es aquel que emite luz cuando está polarizado directamente. Su funcionamiento es bastante sencillo: al conectarlo con la polarización directa, el semiconductor de la parte superior permite que pase la corriente por el cátodo y el ánodo; cuando la corriente circula por el semiconductor, este emitirá luz. Los colores emitidos dependerán del material con que se fabrique el semiconductor, esta variedad ja permitido el desarrollo de nuevas pantallas multicolores.

Es importante tener en cuenta que un diodo LED debe ser protegido; si se encuentra con una pequeña corriente en sentido inverso, no le pasara nada, pero, si existen picos inesperados, podría llegar a dañarse. Una forma de protegerlo, podría ser la instalación de un diodo de silicio común, en paralelo y apuntando en la dirección opuesta.

En un diodo LED, el dispositivo semiconductor se encuentra encapsulado en una cubierta de plástico que puede estar coloreada, aunque es solo por razones estéticas, ya que no influye en el color de la luz que emite.

Los diodos LED pueden aprovecharse en diferentes aplicaciones, por ejemplo:

- **CONTADORES**: se utilizan para desplegar contadores.
- **CORRIENTE CONTINUA**: podemos utilizarlos para indicar la polaridad de una fuente de alimentación continua.
- **CORRIENTE ALTERNA**: son adecuados para indicar la actividad que se presenta en una fuente de alimentación de corriente alterna.
- **ALARMAS**: es posible usar su emisión de luz en dispositivos de vigilancia y alarmas.

Puente H

El **puente H** es un circuito diseñado para controlar la polaridad del voltaje que se aplica a una carga, por ejemplo, a un motor. En otras palabras, se trata de un circuito que utilizaremos para hacer que un motor gire en ambos sentidos.

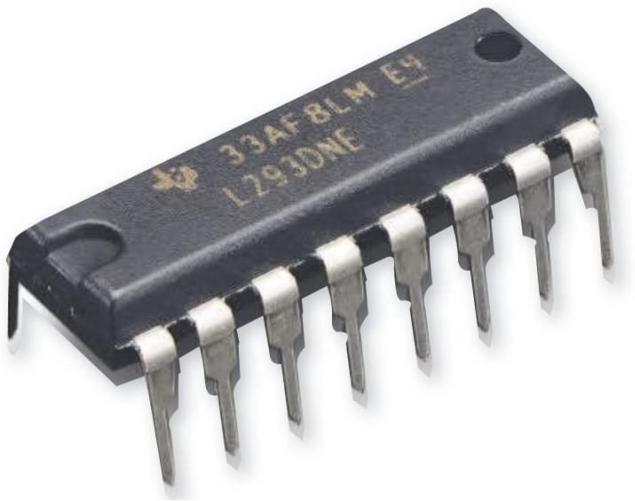
Existen dos posibilidades para contar con un puente H, podríamos construirlo utilizando algunos pocos componentes individuales o también conseguir un circuito integrado. Para comenzar, es recomendable esta segunda opción.

Para entender el funcionamiento de este componente, debemos tener en cuenta que un motor de corriente continua determina la dirección de giro dependiendo de la tensión que apliquemos en sus terminales. Es decir, si conectamos el terminal 1 del motor al positivo, y el terminal 2 al negativo de la pila, como resultado obtendremos un

sentido de giro específico; pero, si realizamos la conexión del positivo y el negativo en forma opuesta, tendremos como resultado un giro en sentido contrario. Como no es eficiente estar cambiando la conexión de los terminales cada vez que deseamos invertir el giro del motor, podemos utilizar un puente H.

El puente H no es más que una disposición específica de componentes, como transistores y diodos, que nos permiten controlar la polaridad de los terminales de salida, teniendo en cuenta algunas funciones lógicas.

Como mencionamos antes, un puente H se puede fabricar en forma manual, utilizando transistores y otros componentes, pero también podemos usar circuitos integrados, tales como el L293B y L293D. De ellos, el L293D, que presentamos en la imagen adjunta, posee diodos de protección y dos puentes H, y proporciona 600 mA al motor; además, soporta voltajes de entrada que van desde 4,5V y 36V.

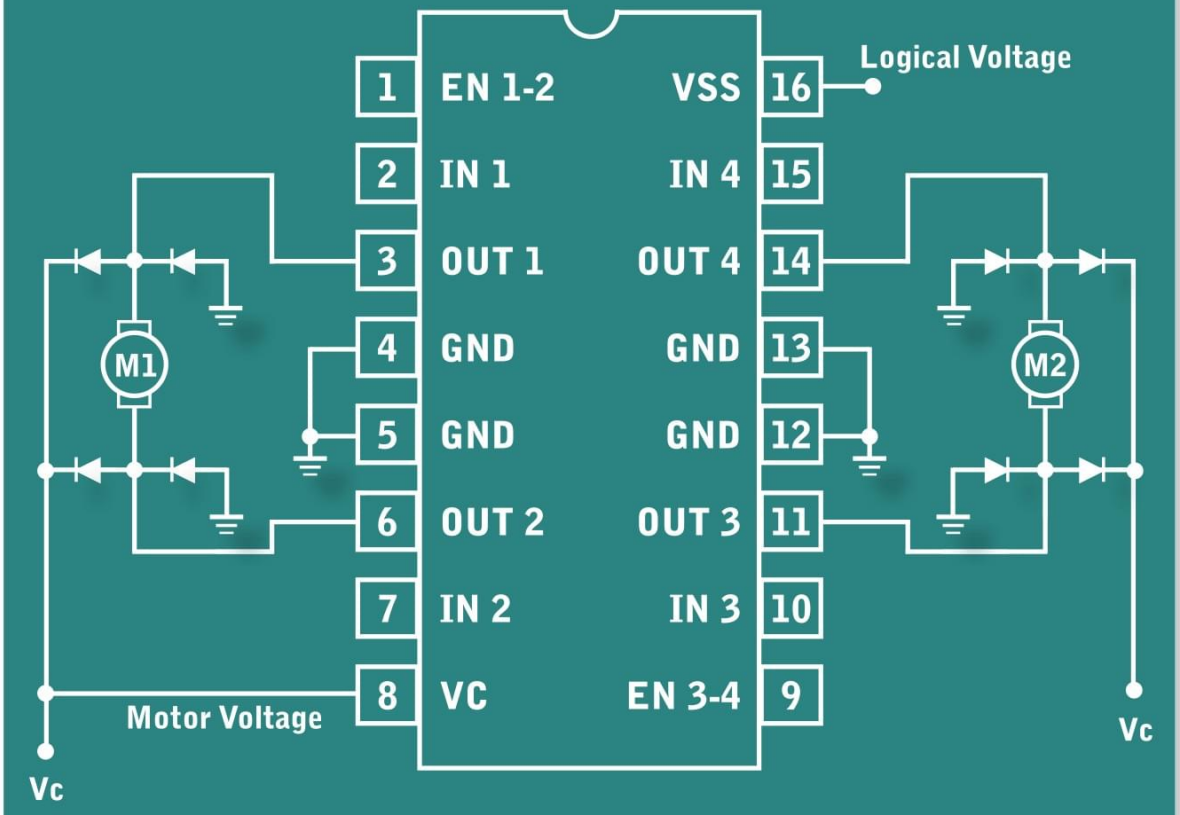


El L293DNE es un puente H que nos servirá para impulsar cargas inductivas, por ejemplo, relés, solenoides, motores paso a paso DC y, también, bipolares.

Primer LED

El primer LED fue desarrollado por Oleg Vladimirovich Losev en el año 1927, aunque llegó a la industria en la década del 60. Actualmente se utilizan en muchas aplicaciones, por ejemplo, en indicadores de estado (encendido/apagado) de diversos componentes electrónicos, en dispositivos de señalización, en paneles informativos, o para alumbrar pantallas de cristal líquido, como las de las calculadoras, las agendas electrónicas y los teléfonos móviles.

+ Esquema de un circuito integrado con dos puentes H



Gracias a los pines 2 y 7, podemos controlar el sentido de giro del motor M1, mientras que los pines 10 y 15 controlarán el sentido de giro del motor M2. Los pines 1 y 9 se encargan de activar o desactivar cada puente H en forma independiente.

Broche de presión de pila

Aunque se trata de un componente muy sencillo, lo necesitamos para conectar una pila de 9V a clavijas de corriente, que a su vez conectaremos al protoboard o a la placa Arduino.

También podemos encontrarlo con el nombre de **conector de batería 9V**. Se presenta como una estructura que posee dos broches adecuados para las baterías de 9V, unidas a dos filamentos que serán los que debemos conectar al protoboard o a la placa.

En este punto debemos tener en cuenta que, para realizar la alimentación eléctrica de Arduino, podemos proceder de varias formas:

MEDIANTE EL PUERTO USB

Se trata de la forma más habitual; conservamos el cable USB conectado a la placa y a la computadora, de esta forma la mantendremos con corriente. Lo malo es que necesitamos tener una computadora cerca en todo momento.

UTILIZAR UNA BATERÍA 9V Y EL CONECTOR DE CORRIENTE

En este caso, hacemos uso de una batería 9V; para ello necesitaremos un broche de presión de pila que, en uno de sus extremos, presente el conector adecuado para la entrada de corriente de la Arduino UNO.

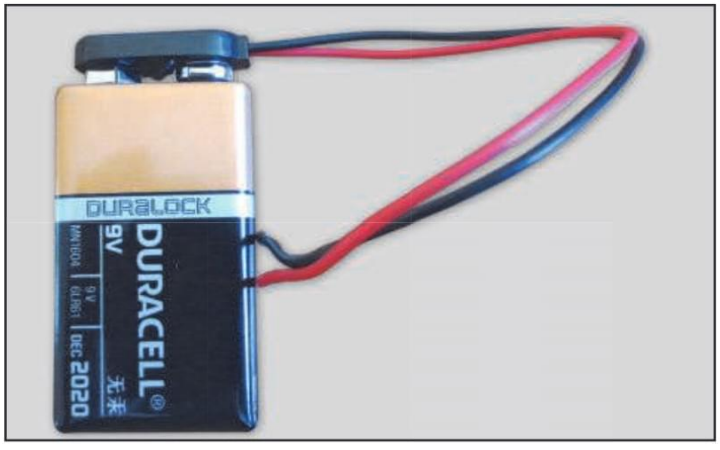
UTILIZAR UNA BATERÍA 9V Y LOS PINES DE ARDUINO

Mediante esta alternativa, también utilizaremos una batería 9V y un broche de presión de pila, pero, en lugar de un conector para la entrada de corriente, solo tendremos los filamentos positivo y negativo. Para realizar la conexión, usaremos los pines Vin y GND de la placa.

Esta tercera opción es bastante útil, por ejemplo, cuando necesitamos la misma tensión de 9V para algún componente del circuito; en este caso es posible hacer la conexión a través de la protoboard. Para conectar de esta manera procederemos como se indica a continuación.

Paso a paso: Alimentar Arduino mediante el protoboard

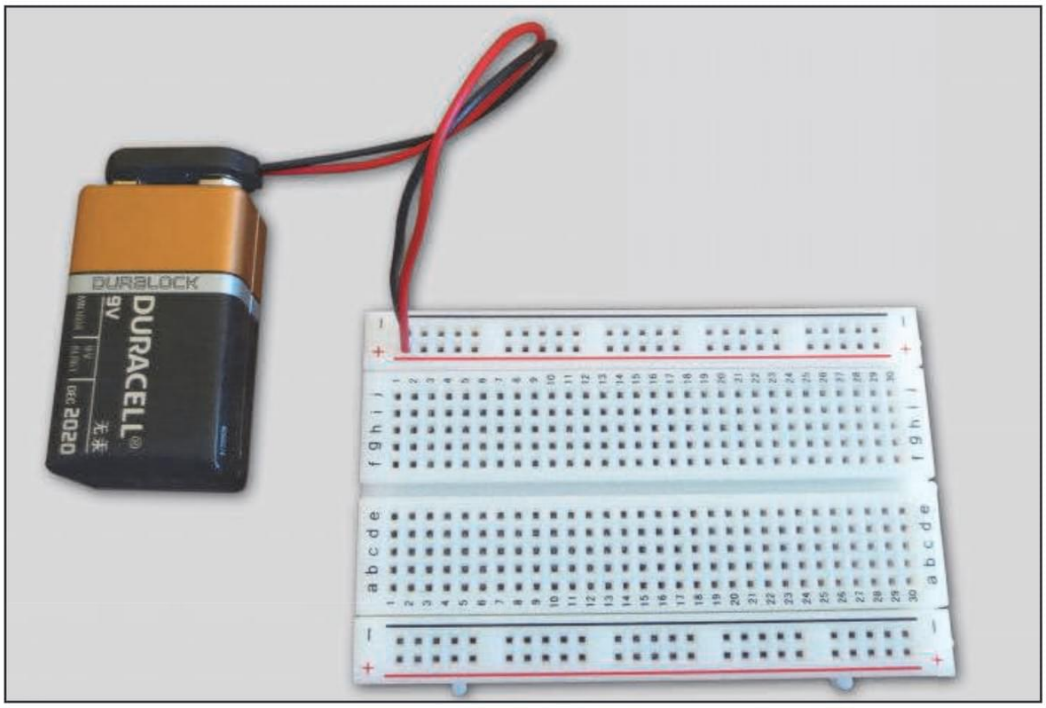
01



En primer lugar, conecte la batería 9V al broche de presión de pila. Debe efectuar la conexión a la pila hasta escuchar el clic que indica una postura correcta.

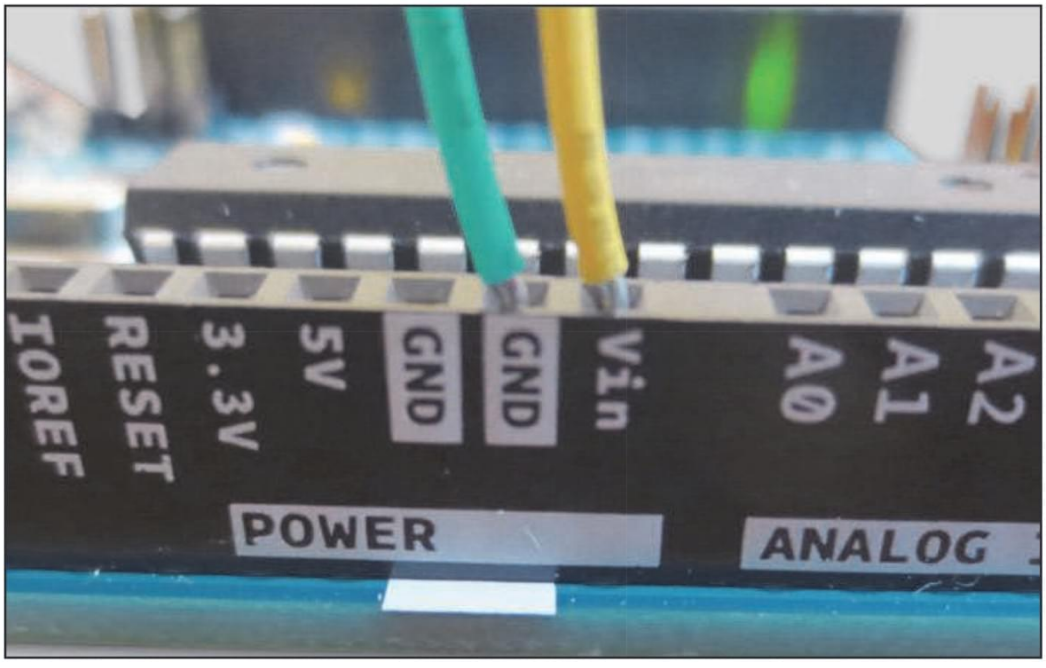
02

Tome los filamentos rojo y negro que corresponden al positivo y al negativo del broche de presión y conéctelos en los lugares adecuados del protoboard.

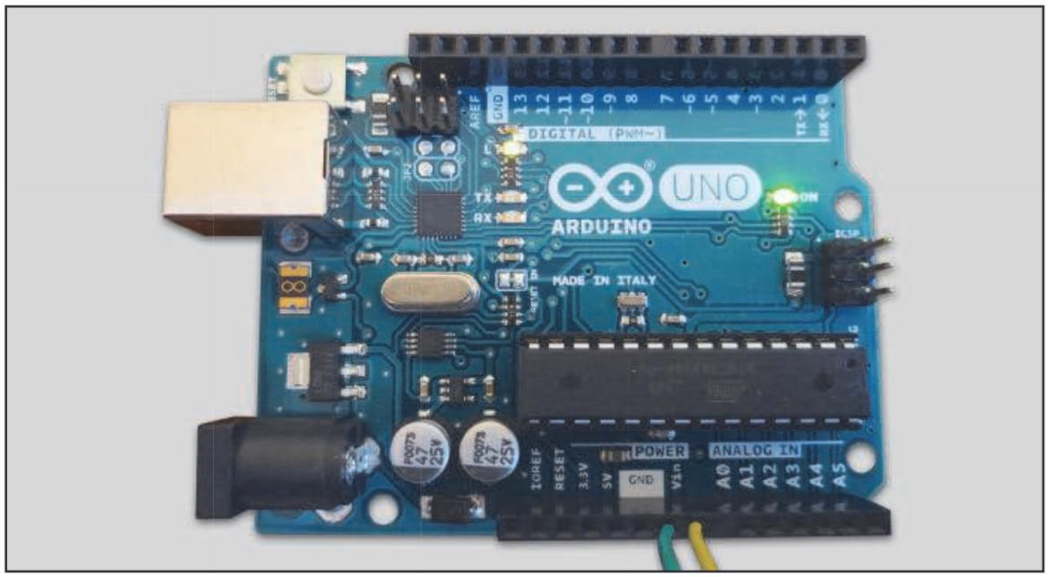


03

Para continuar, utilice dos cables de puente para conectar el protoboard con la placa Arduino. Conecte el polo positivo a Vin y el polo negativo a GND.



04 Si la conexión se realizó en forma correcta, la placa Arduino UNO estará energizada, puede verificarlo mediante una inspección del LED de corriente.



Potenciómetro

Es una resistencia variable que posee tres terminales o conectores. Dos de ellos se conectan a una resistencia fija, mientras que el tercero se puede mover consiguiendo valores diferentes. Lo importante del **potenciómetro** es que podemos elegir el valor por tomar; de esta forma, controlaremos la intensidad de corriente que fluye por el circuito o la diferencia de potencial, según esté conectado en paralelo o en serie, respectivamente. La variación de la resistencia va desde un valor mínimo, que generalmente es de 0 ohmios, hasta un valor máximo Rmax (5k, 10k o 20k ohmios).

Considerando su forma, encontramos dos tipos de potenciómetros:

LINEALES

Este tipo de potenciómetro tiene forma rectangular, con un control deslizante que debemos mover para establecer la posición adecuada.

ROTATIVOS

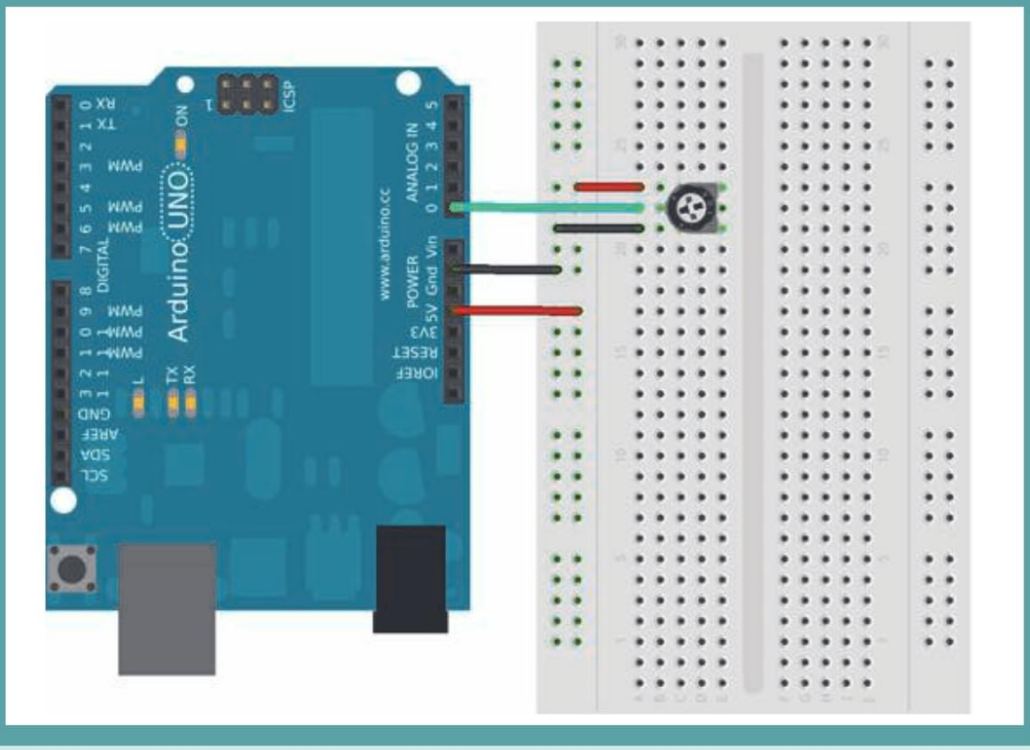
Son los más comunes, y son los potenciómetros que utilizaremos para nuestros proyectos con Arduino. Visualmente se presentan como un dispositivo con un mando giratorio (perilla) que nos permitirá seleccionar el valor por utilizar.

Si pensamos en la relación existente en la posición y la resistencia, encontraremos potenciómetros lineales, parabólicos o exponenciales.

El potenciómetro rotativo es el más común en proyectos de Arduino. Se trata de un elemento pasivo que, técnicamente, funciona como un interruptor.

Conectar un potenciómetro a una placa Arduino UNO requiere el uso del pin analógico 0, además de los pines 5V y GND. Podemos proceder tal como se indica en la siguiente imagen.

+ Conectar Arduino a un potenciómetro



En este esquema vemos la forma en que podemos conectar Arduino a un potenciómetro.

Luego necesitaremos un sencillo código que será el encargado de leer e interpretar el valor de tensión:

```
const int analogPin = A0;
int value;
int position;
void setup() {
}
void loop() {
value = analogRead(analogPin);
   position = map(value, 0, 1023, 0, 100);
   // código para trabajar con el valor medido
   delay(1000);
}
```

Como vemos, es una pequeña porción de código que utilizaremos para trabajar con el potenciómetro. Aprenderemos en detalle la forma de trabajar con programación en el **capítulo 5** de esta obra, por ahora solo la consideraremos un sencillo ejemplo.

Pantalla de cristal líquido

Aunque existen pantallas de cristal líquido en variados tamaños, nos bastará con una que disponga de dos filas y 32 caracteres en total.

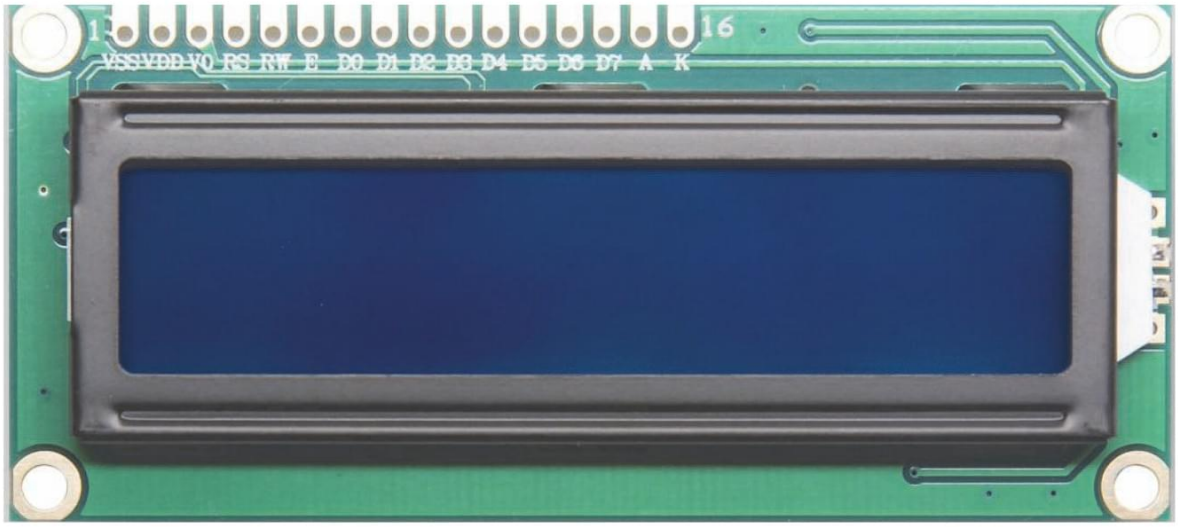
Estas pantallas se encuentran unidas a una pequeña placa de circuito con pines para entrada/salida de datos y están diseñadas para que podamos mostrar información de manera gráfica. Lo importante es que, mediante Arduino, podremos desplegar información utilizando estas pantallas.

Para enviar datos a la pantalla LCD, utilizaremos la librería **LiquidCrystal** que encontramos en el IDE. Esto es a través de los pines de entrada/salida de datos que poseen las placas que vienen unidas a la pantalla de LCD.

"Las pantallas de cristal líquido son capaces de representar cualquier carácter alfanumérico."

Por ejemplo, en una pantalla de dos filas y 16 caracteres en cada una, hallaremos un listado de pines, como el siguiente (de izquierda a derecha):

- **Pin 1**: VSS o GND
- **Pin 2**: VDD o alimentación +5V.
- **Pin 3**: voltaje de contraste, se debe conectar a un potenciómetro.
- **Pin 4**: se trata de la selección de registro, para elegir el dispositivo para su uso.
- **Pin 5**: pin de lectura/escritura, podemos establecer el estado para escribir o leer datos (HIGH o LOW).
- **Pin 6**: mediante este pin, podemos habilitar o deshabilitar el LCD.
- **Pin 7 hasta el 14**: se trata de los pines de datos, mediante ellos se envía o recibe la información.
- **Pin 15**: es el ánodo del LED de iluminación para la luz de fondo (+5V).
- **Pin 16**: es el cátodo del LED de iluminación para la luz de fondo (GND).



Aquí vemos una pantalla de cristal líquido de 16 caracteres por 2 filas. Este modelo utiliza caracteres blancos acompañados por una luz azul de fondo.

Una configuración básica para utilizar la pantalla de cristal líquido con una placa Arduino requiere el uso de un potenciómetro de 10 k. El conector de la derecha se conecta a 5V en la placa Arduino, el conector de la izquierda se conecta en GND de Arduino, mientras que el conector central se conecta al tercer pin en el LCD, que corresponde al voltaje de contraste. Luego seguimos las siguientes instrucciones:

- El pin 1 del LCD se conecta al GND de Arduino.
- El pin 2 del LCD se conecta a 5V en Arduino.
- El pin 4 se conecta al pin 12 de Arduino.
- El pin 5 se conecta a GND.
- El pin 6 del LCD se conecta al pin 11 en Arduino.
- Los pines 7, 8, 9 y 10 del LCD se dejan sin conectar.
- Los pines 11, 12, 13 y 14 del LCD deben ser conectados en orden a los pines 5, 4, 3 y 2 de Arduino.
- Para terminar, conectamos el pin 15 del LCD a 5V y el pin 16 a GND.

Una vez realizadas las conexiones, cargamos un código de ejemplo para ver la pantalla en funcionamiento. En el **capítulo 5**, analizaremos en detalle la forma en que podemos cargar programas de ejemplo en el IDE de Arduino.

Motor de corriente continua

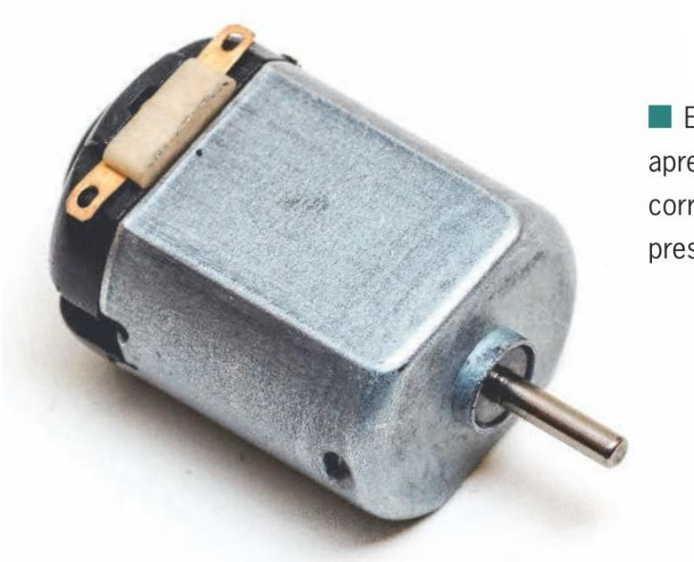
Es un dispositivo que puede convertir la energía eléctrica en mecánica, esto sucede cuando aplicamos electricidad en sus terminales.

La verdad es que estamos rodeados de motores eléctricos, de todos los tipos y tamaños, por ejemplo, el motor que mueve el automóvil, o los que hacen girar el plato del microondas o proporcionan el movimiento para reproducir un disco compacto.

En nuestros proyectos con Arduino, generalmente trabajaremos con pequeños motores de corriente continua (motor de corriente directa o motor DC -*direct current*-)por lo que es necesario conocerlos.

En primer lugar debemos recordar que los imanes poseen un polo positivo y uno negativo, y, mientras los polos opuestos se atraen, los iguales se repelen. Esto es importante pues se relaciona estrechamente con la corriente eléctrica ya que, cuando hacemos circular una corriente eléctrica a través de un conductor que se encuentra en un campo magnético, este se verá sometido a la fuerza electromotriz (fuerza mecánica), que resulta ser la base para el funcionamiento de un motor eléctrico.

En otras palabras, si una corriente circula por un conductor que está entre los polos de un imán, se presentará una fuerza mecánica opuesta a los cambios de la corriente, que intentará hacer girar el conductor para compensarlos.



En esta imagen podemos apreciar un pequeño motor de corriente continua de 3V, que presenta 100 RPM.

Algunas de las partes principales que encontramos en un motor de corriente continua son:

ESTÁTOR
Es la parte inmóvil que, por lo general, incluye imanes potentes fijos, o genera un campo variable mediante corriente alterna.

ROTOR
Aquí se encuentran espiras o arrollamientos de hilos de cobre alrededor de un núcleo, de esta forma la fuerza que se ejerce sobre el rotor se multiplica en forma proporcional.

Pulsador

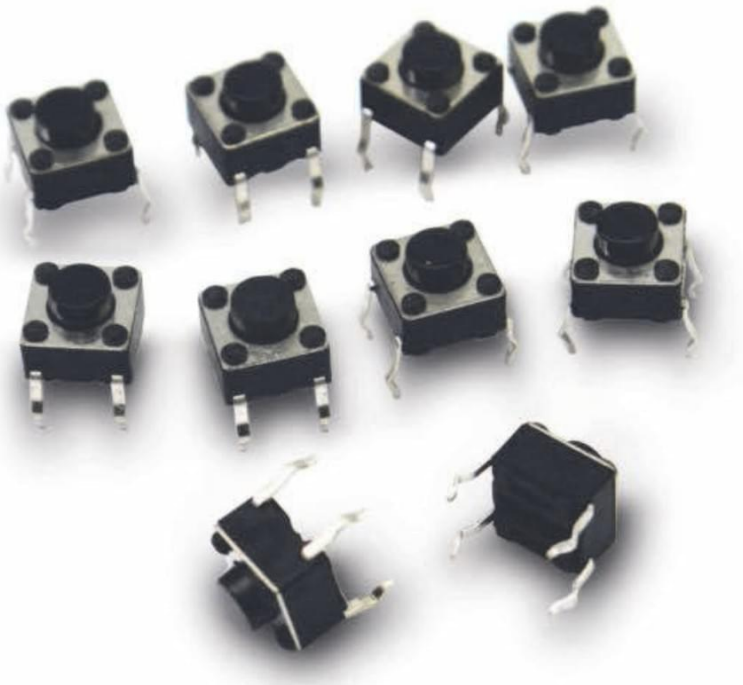
Se trata de un elemento sencillo que es capaz de cerrar un circuito cuando lo presionamos, podemos utilizarlo para abrir o cerrar el paso a una señal.

El **pulsador** es, en realidad, un interruptor que mantendrá una posición de cerrado mientras lo mantengamos pulsado. Por el contrario, cuando dejemos de pulsarlo, conservará la posición de abierto.

A simple vista nos encontramos con un pequeño elemento cuadrado que posee un botón en su parte superior, esto parece evidente en un pulsador, pero, al examinarlo con más detalle, veremos que en su parte inferior se presentan cuatro patillas de conexión, esto es así porque ambos pares están internamente conectados por lo que, en realidad, funcionan como dos patillas de conexión.

Como sabemos, la función de un pulsador consiste en abrir o cerrar un circuito, por lo tanto, es posible imaginar muchos usos para este elemento, por ejemplo, controlar un LED, un motor o un zumbador, permitiendo que estos trabajen o dejen de hacerlo.

Sin embargo, debemos considerar que el pulsador presenta un funcionamiento comparable con el de un timbre, es decir, que cerrará el circuito y permitirá el funcionamiento de otro elemento solo mientras lo mantengamos pulsado, por esta razón no sería práctico utilizarlo para hacer funcionar un motor, pues tendríamos que mantenerlo pulsado en todo momento.



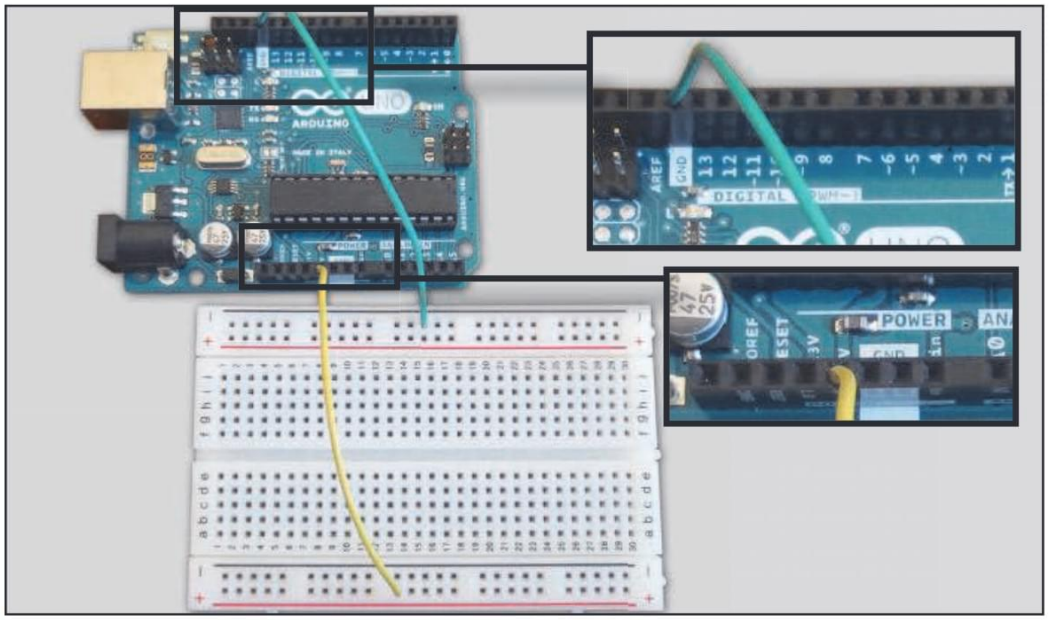
Aunque puede parecer que el pulsador posee 4 terminales, la verdad es que internamente ambos pares se encuentran unidos, y se comportan como dos terminales.

En este sentido, un uso más práctico sería utilizarlo para indicar, por ejemplo, que se encienda o apague un LED o un motor, con una sola pulsación, dependiendo del estado original del componente. Para esto, necesitamos que la placa detecte que el pulsador fue presionado y actúe en consecuencia, encendiendo o apagando, según si el componente por controlar se encuentra apagado o encendido, respectivamente. Para lograrlo, podemos acompañar el pulsador y su correcto conexionado con un programa que le permita decidir la acción por ejecutar en cada caso. En el siguiente **Paso a paso**, utilizaremos un pulsador para controlar el encendido de un LED, conectaremos la placa Arduino UNO a la computadora mediante USB; por ahora solo usaremos esta conexión para obtener la energía que necesitamos.

Paso a paso: Control sencillo de un LED

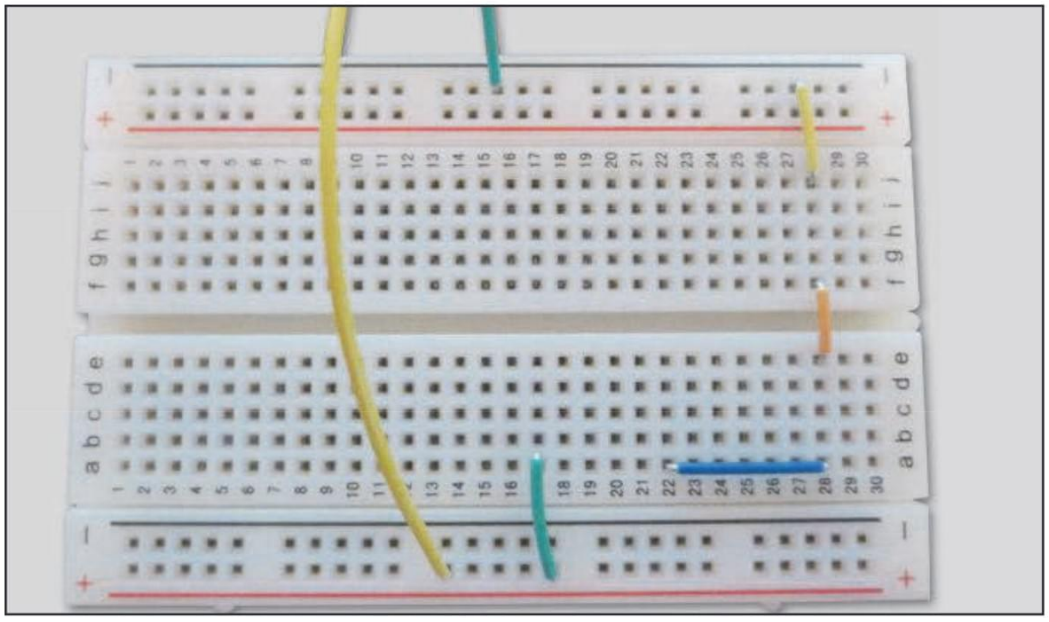
01

Conecte la placa Arduino UNO al protoboard. Para lograr el comportamiento de un circuito eléctrico de corriente, use un cable que se comporte como ánodo, y otro, como cátodo, utilizando los pines GND y 5V. De esta forma energizará el protoboard.



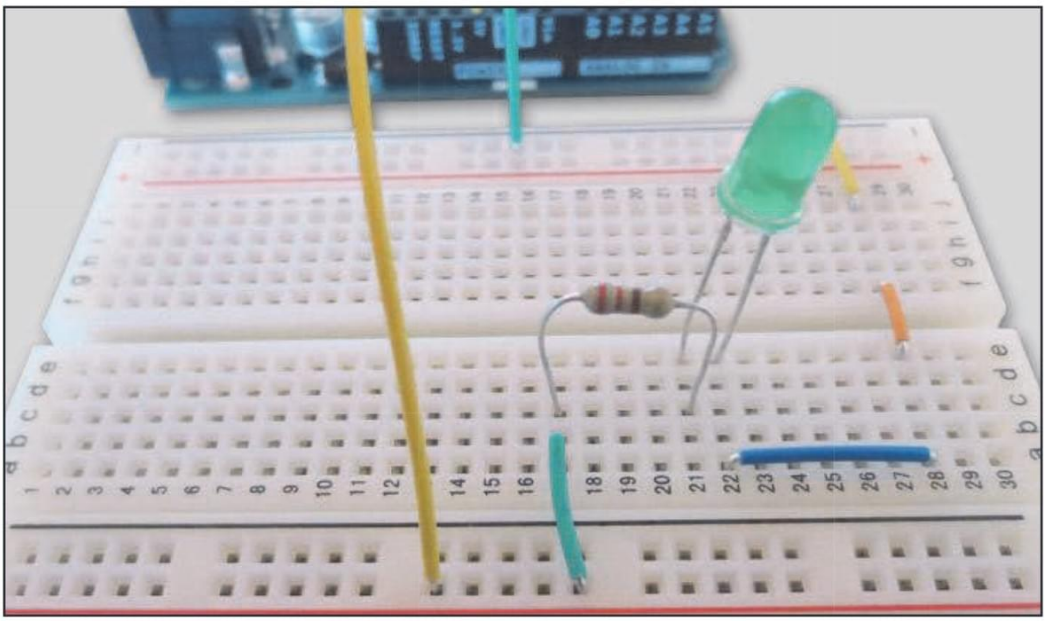
02

En primer lugar, prepare las conexiones para que se encienda un LED sin necesidad de utilizar un pulsador. Coloque las conexiones de puente tal como indica la imagen.



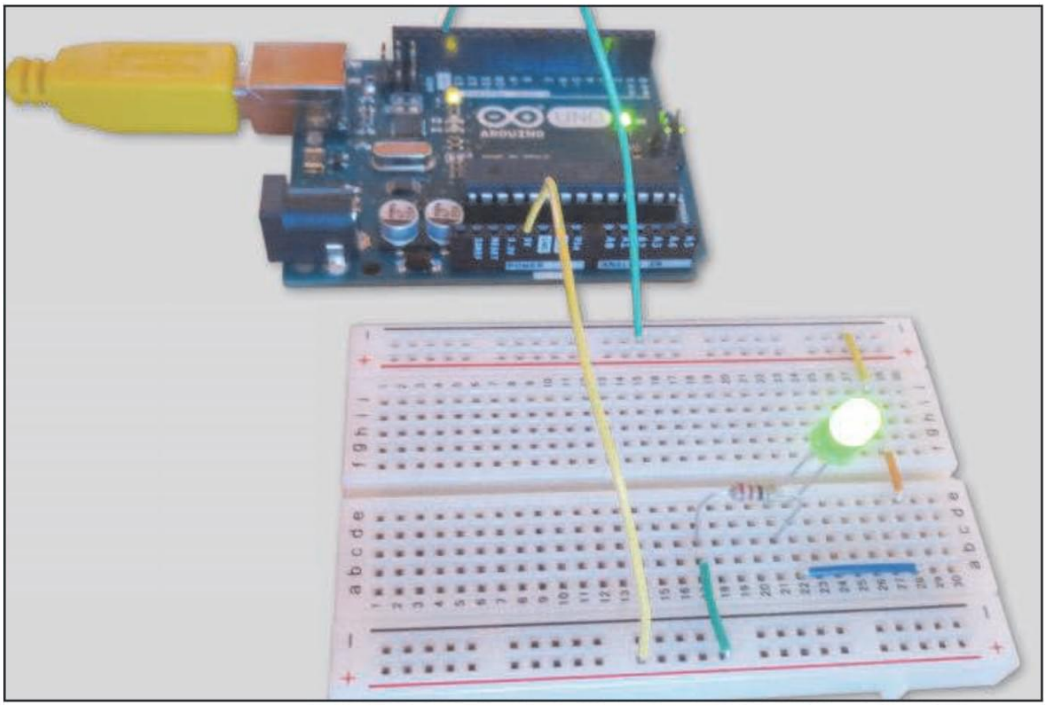
03

Para continuar, conecte el LED que utilizaremos, en este caso se trata de un LED verde, y también una resistencia.

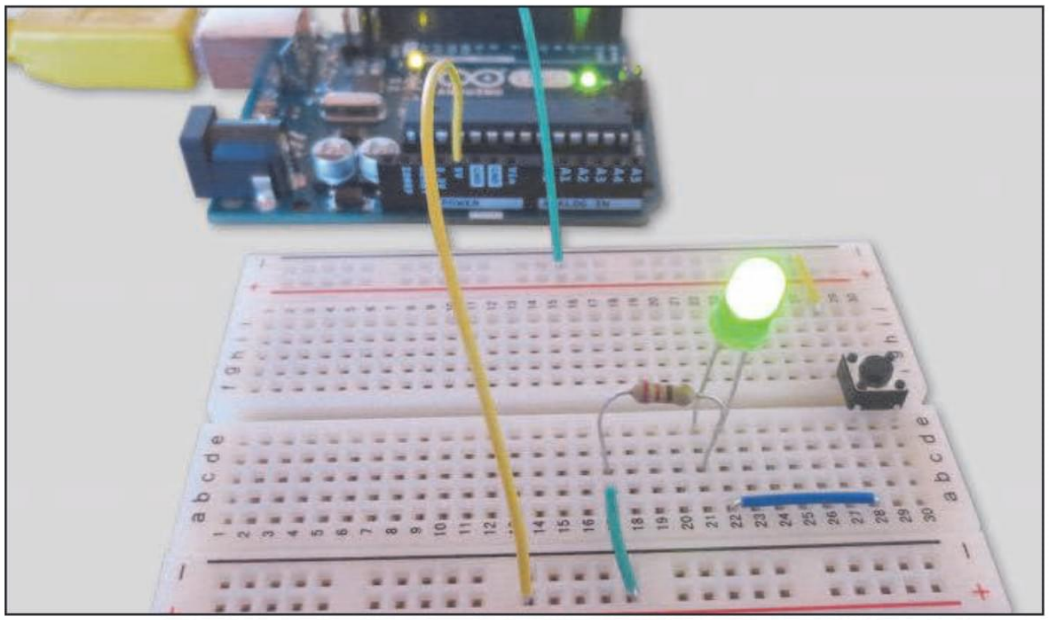


04

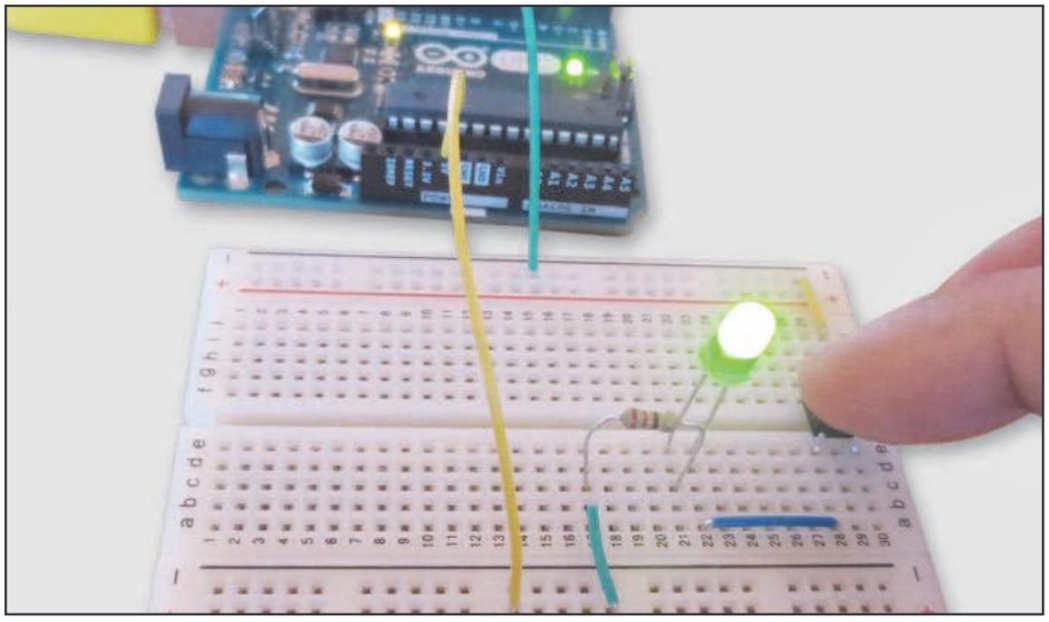
Al terminar de montar este circuito, conéctelo a la PC mediante USB, verá que el LED se enciende de forma continua. Las salidas 5V y GND permiten hacer un circuito igual que como lo haría si utilizara los cables conectados a una pila.



05 Ahora, sustituya uno de los puentes por un pulsador, tal como se ve en la imagen. Con esta configuración, el pulsador solo se encarga de cerrar el circuito, por lo que no es operativo.



06 Por el contrario, si modifica el circuito para unir la esquina inferior izquierda con la superior derecha, el pulsador será operativo y tendrá que presionarlo para encender el LED. Esto se logra moviendo el cable de puente que ubicó en la parte superior del protoboard.

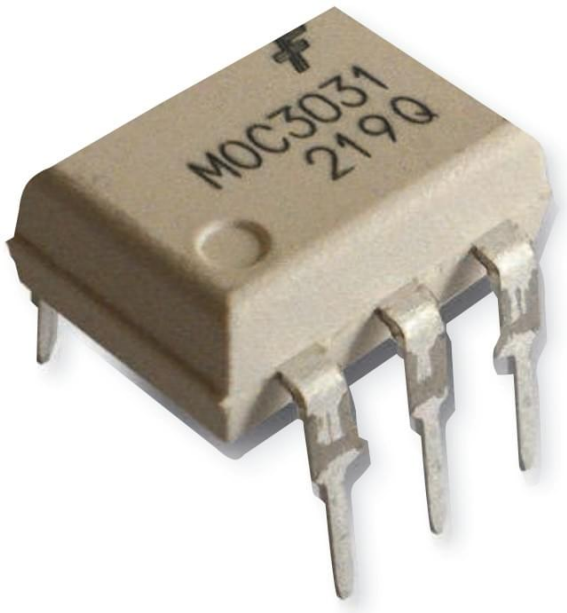


Como vimos en el apartado 4, es posible controlar un LED sin necesidad de agregar código, aunque se trata de un ejemplo muy sencillo es adecuado para entender el funcionamiento de un pulsador.

Optoacoplador

El **optoacoplador** u **optoaislador** es un dispositivo que se encarga de aislar circuitos, es decir, puede separar circuitos dentro de un proyecto para evitar que algunas de sus partes se vean afectadas por voltajes o corrientes excesivos, que podrían causar daños a algunos componentes.

Pero un optoacoplador no solo puede ser utilizado para aislar circuitos, ya que también se aplica cuando es necesario realizar el control de potencia, cuando necesitamos hacer uso de un relé, cuando precisamos interfaces en circuitos lógicos o entre señales de corriente alterna y circuitos lógicos, entre otras tareas.



Un optoacoplador es un componente diseñado para aislar ciertas partes de un circuito. De esta forma, se evitan daños por la presencia de voltajes o corrientes excesivas.

En su construcción, se trata de un circuito integrado sencillo, generalmente compuesto por un LED y un fototransistor. Cuando hacemos circular una señal eléctrica a través del LED, se encenderá; la luz emitida es detectada por el fototransistor que trabajará en modo saturación. De esta forma puede actuar como interfaz entre dos circuitos, que estarán unidos ópticamente y así se protegerán contra los picos de tensión.

Resistencias

Una **resistencia** es un componente capaz de oponerse al paso de la corriente, por lo tanto, entrega un cambio en la tensión y en la corriente. Existen diferentes tipos de resistencias; si tenemos en cuenta su funcionamiento, podemos agruparlas en tres categorías:

- **RESISTENCIAS FIJAS**: presentan un valor que no es posible modificar.
- **RESISTENCIAS VARIABLES**: poseen un valor que nosotros podemos variar modificando la posición de un contacto deslizante; se las conoce como **potenciómetros**.
- **RESISTENCIAS ESPECIALES**: varían su valor en función de la estimulación que reciben de un elemento externo, por ejemplo, luz o temperatura.

Por lo general, las resistencias o resistores poseen bandas de color que nos indican su valor.

Optoacoplador 4N35

El **4N35** es uno de los optoacopladores más utilizados, por lo que seguro nos toparemos con él en muchos proyectos. Las principales características de su LED son las siguientes: una caída de voltaje típica de 1.15V para una corriente de 10 mA, la corriente máxima que soporta es de 60 mA y la disipación de potencia máxima es de 120 mW. Por otra parte, las características de su fototransistor son: caída de voltaje de colector a emisor de 45V para una corriente de colector de 1 mA, caída de voltaje del emisor a la base de 7.8V para una corriente de emisor de 100 µA, caída de voltaje del colector a la base de 100V para una corriente de colector de 100 µA, ganancia de corriente (Hfe) para una corriente de colector de 2 mA, y una caída de voltaje de colector a emisor de 5V es de 400.

No todas las resistencias son iguales. El valor que las diferencia se mide en ohmios y las identificamos por medio de las bandas de color que poseen. Cada una de estas bandas de colores representa un número que podemos utilizar para obtener el valor final de la resistencia o resistor. Las dos primeras bandas hacen referencia a las dos primeras cifras del valor; la tercera banda nos indica los ceros que debemos aumentar al valor anterior para saber el valor final de la resistencia. La cuarta banda se encarga de representar la tolerancia; si vemos una quinta banda de color, se refiere a su confiabilidad. En la siguiente tabla, analizamos esta lista de colores.

CÓDIGOS DE COLOR

COLOR	BANDA 1	BANDA 2	BANDA 3 (Multiplicador)	% TOLERANCIA
Negro	0	0	x1	
Café	1	1	x10	1%
Rojo	2	2	x100	2%
Naranja	3	3	x1000	
Amarillo	4	4	x10000	
Verde	5	5	x100000	0,5%
Azul	6	6	x1000000	
Violeta	7	7	x10000000	
Gris	8	8	x100000000	
Blanco	9	9	x1000000000	
Dorado				5%
Plata				10%

Códigos de color que nos permiten identificar el valor de una resistencia o resistor.

Fotorresistencia

Una **fotorresistencia**, **resistencia dependiente de la luz** o **fotocélula** es una resistencia del tipo variable, capaz de cambiar el valor de su resistencia dependiendo del nivel de luz que toque su superficie. De esta forma, cuanto mayor sea el nivel de luz que toque su superficie, menor será su resistencia, por el contrario, su resistencia aumentará mientras menos luz toque su superficie.

La construcción de una fotorresistencia requiere el uso de materiales que sean fotosensibles, por esta razón se utilizan sulfuro de talio, sulfuro de cadmio, sulfuro de plomo y seleniuro de cadmio, entre otros.

Las fotorresistencias son utilizadas en proyectos relacionados con iluminación, apagado y encendido de alumbrado, alarmas, en medidores de luz, etcétera.

El funcionamiento de una fotorresistencia o **LDR** procede de la siguiente forma: por una parte, cuando no se expone a radiaciones luminosas, sus electrones se encuentran unidos firmemente en los átomos que la conforman; por otra parte, cuando se expone a radiaciones luminosas, la energía libera los electrones haciendo que el material sea más conductor y, por lo tanto, disminuye su resistencia.

Es necesario considerar que las LDR reducen su resistencia cuando son expuestas a una radiación luminosa que se encuentra en una determinada banda de longitudes de onda. Por ejemplo, las fotorresistencias construidas con sulfuro de cadmio son sensibles a todas las radiaciones luminosas visibles, pero las fotorresistencias que han utilizan sulfuro de plomo son sensibles solo a las radiaciones infrarrojas.

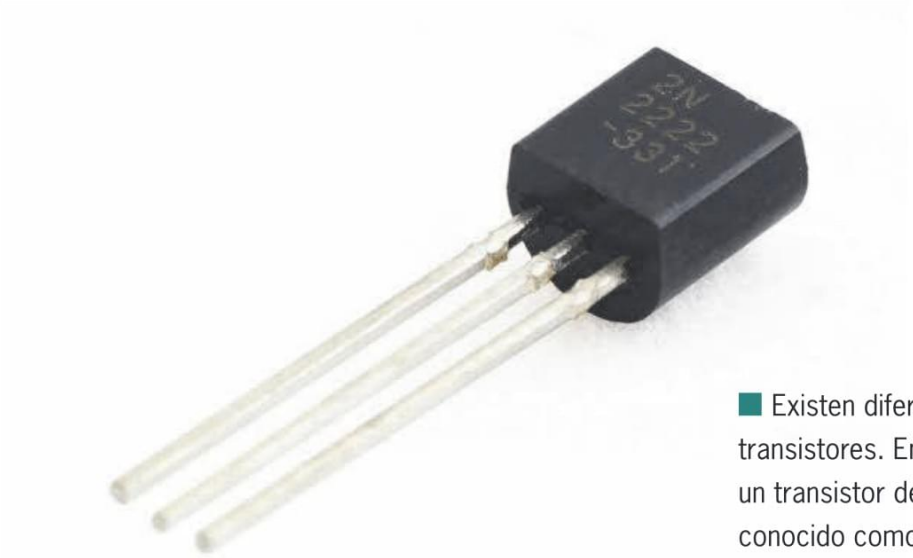
Transistor

Es un elemento semiconductor capaz de trabajar como un interruptor electrónico. Posee tres terminales y puede regular el flujo de corriente o de tensión, por esta razón se utiliza como interruptor o amplificador en las señales electrónicas.

Posee tres partes esenciales: el emisor (emite electrones), el colector (recibe o recolecta electrones) y la base (modula el paso de los electrones). Si aplicamos una pequeña señal eléctrica entre la base y el emisor, se modulará la corriente que circula entre el emisor y el receptor.

Las funciones de un transistor son, básicamente, dos:

- **INTERRUPTOR**: permite que pasen o se corten las señales eléctricas, dependiendo de una señal de mando. Se abre o se cierra, para dejar pasar o cortar la corriente en el circuito.
- **AMPLIFICADOR**: es capaz de recibir una pequeña señal y aumentarla.



Existen diferentes tipos y formas de transistores. En la imagen, apreciamos un transistor de unión bipolar, más conocido como **BJT**.

Tipos de transistores

Existen diferentes tipos de transistores, una de las clasificaciones más utilizadas los divide en: transistores bipolares o BJT (*Bipolar Junction Transistor*) y transistores de efecto de campo o FET (*Field Effect Transistor*). Los transistores de efecto de campo incorporan muchos transistores, por ejemplo, los JFET, MOSFET y MISFET.

Un transistor puede presentar tres estados posibles:

- **ACTIVO**: puede dejar pasar más o menos corriente. Actúa como un amplificador.
- **CORTE**: impide el paso de la corriente. No circula intensidad por la base, por ello la intensidad de colector y emisor es nula. El transistor entre colector y emisor se comporta como si se tratara de un interruptor abierto.
- **SATURACIÓN**: deja que pase toda la corriente. Cuando por la base circula una intensidad, se aprecia un incremento de la corriente de colector bastante considerable. Aquí el transistor entre colector y emisor se comporta como si fuese un interruptor cerrado.

Zumbador piezoeléctrico

Es un elemento bastante interesante, pues podemos utilizarlo para producir ruidos o detectar vibraciones. Un **buzzer** o **zumbador piezoeléctrico** hace uso de una propiedad denominada **piezoelectricidad**.

Esta propiedad se relaciona con algunos cristales que, cuando son sometidos a tensiones mecánicas, adquieren polarización eléctrica, por lo que presentan tensiones eléctricas. También, cuando son sometidos a un campo eléctrico, se deforman por la acción de fuerzas internas.

Lo importante es que, al someterlos a una tensión eléctrica variable, vibrarán. Un ejemplo de su aplicación lo encontramos en circuitos electrónicos digitales, pues disponen de un reloj interno que vibra, basado en cristales de cuarzo piezoeléctrico. El cristal que corresponde a Arduino (microcontrolador ATmega 328) late a 16 Mhz por segundo.

Un buzzer piezoeléctrico ofrece una calidad de sonido que no podemos clasificar como alta fidelidad, pero es suficiente para generar tonos audibles.

Teniendo esto en cuenta, al conectar un piezoeléctrico a una señal digital, vibrará a una frecuencia que se corresponde con la variación eléctrica a la que se expone. Si la frecuencia es audible, podremos escuchar el zumbido que produce.

Realizar una conexión básica de un zumbador piezoeléctrico a una placa Arduino es bastante sencillo, solo necesitamos conectar el negativo a GND y el positivo al pin 9 de la placa. En este punto debemos ser cuidadosos, pues los buzzer poseen polaridad, por lo tanto, no sonarán si los conectamos incorrectamente.

Asimismo, es necesario considerar el uso de un pin PWM (por ejemplo el pin 9), porque la alternancia entre HIGH y LOW es la que entregará el efecto piezoeléctrico, y será más cómodo usar un pin PWM que programar ese efecto en un pin normal.

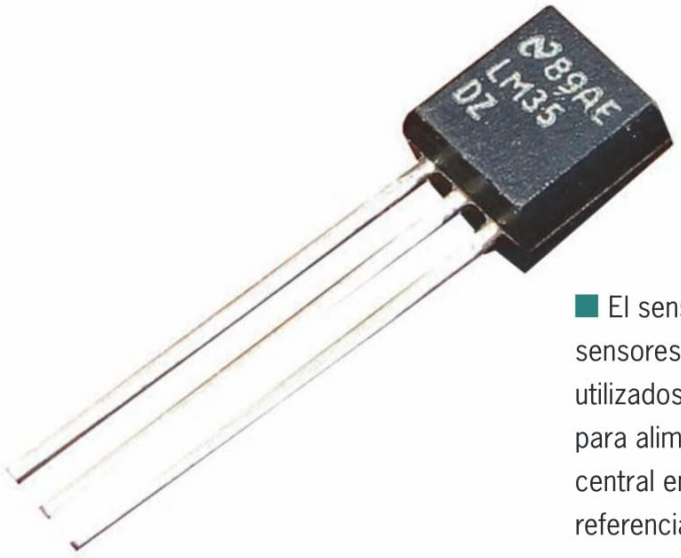
Un código de prueba para hacer funcionar el buzzer es el siguiente, lo analizaremos y ampliaremos en capítulos posteriores:

```
void beep(unsigned char pausa)
        {
            analogWrite(9, 20);
            delay(pausa);
            analogWrite(9, 0);
            delay(pausa);
        }
```

Sensor de temperatura

Es un sensor que posee la capacidad de cambiar a tensión de salida dependiendo de la temperatura que sea detectada por su encapsulado. Los sensores de temperatura, por ejemplo el LM35, poseen su propio circuito de control, por lo que pueden entregar una tensión proporcional a la temperatura detectada.

Entre los sensores de temperatura, los LM35 son los más utilizados por los aficionados a la electrónica, y resultan adecuados para nuestros primeros proyectos. En ellos la tensión entregada es lineal con la temperatura, por lo que aumentan el valor de la tensión en unos 10 mV por cada grado centígrado detectado. Poseen un rango de medición bastante amplio que va desde los -55 ºC hasta los 150 ºC; para estos límites entregarán -550 mV y 1500 mV, respectivamente.



El sensor LM35 es uno de los sensores de temperatura más utilizados. Sus pines extremos sirven para alimentación, mientras que el central entrega la medición en una referencia de tensión a 10 mV/ºC.

Sensor de inclinación

Otro de los componentes importantes para trabajar en proyectos con Arduino es el **sensor de inclinación**. Se trata de un dispositivo que funciona como un interruptor, abriéndose o cerrándose, dependiendo de su orientación.

Los sensores de inclinación, también conocidos como **inclinómetros**, se crearon para convertir una magnitud física en una eléctrica. Para estos sensores la magnitud física es la inclinación, y son capaces de medir desde unos pocos grados hasta los 360º. La magnitud de salida puede ser en corriente, tensión o una señal digital.

Estos sensores se presentan como cilindros de pocos milímetros que, en su interior, contienen una o dos bolas conductoras, capaces de cerrar el circuito con los pines presentes en la parte inferior del cilindro. Al hacer contacto, la corriente fluirá, pero, a partir de un ángulo de inclinación, el contacto cesará y también el paso de la corriente.

Servomotor

Se trata de un motor que puede girar 180º. Podemos controlarlo mediante el uso de señales eléctricas en forma de pulsos, que son enviadas mediante una placa Arduino.

Los **servomotores**, más conocidos como **servos**, son utilizados en tareas específicas, por ejemplo, relacionadas con la robótica, la automatización, paneles solares, entre otros.

En una sección anterior de este mismo capítulo, conocimos los motores DC, los que poseen la característica de que pueden girar sin detenerse. No podemos utilizarlos para que den determinadas vueltas o que se detengan en una posición fija, solo se encargan de girar sin parar hasta que interrumpimos el suministro de corriente. Por esta razón, generalmente no se utilizan en robótica, pues necesitamos efectuar movimientos precisos y también mantener posiciones fijas.

Para tareas relacionadas con la robótica u otras similares, utilizamos motores paso a paso y servomotores. Estos últimos son un tipo especial de motor que permite el control de posición, se trata de un sistema que posee elementos electromecánicos y electrónicos.

Los servomotores son dispositivos electromecánicos que contienen un motor eléctrico, engranes y una tarjeta de control, en una carcasa de plástico.

Podemos clasificar a los servomotores dependiendo de sus características de rotación:

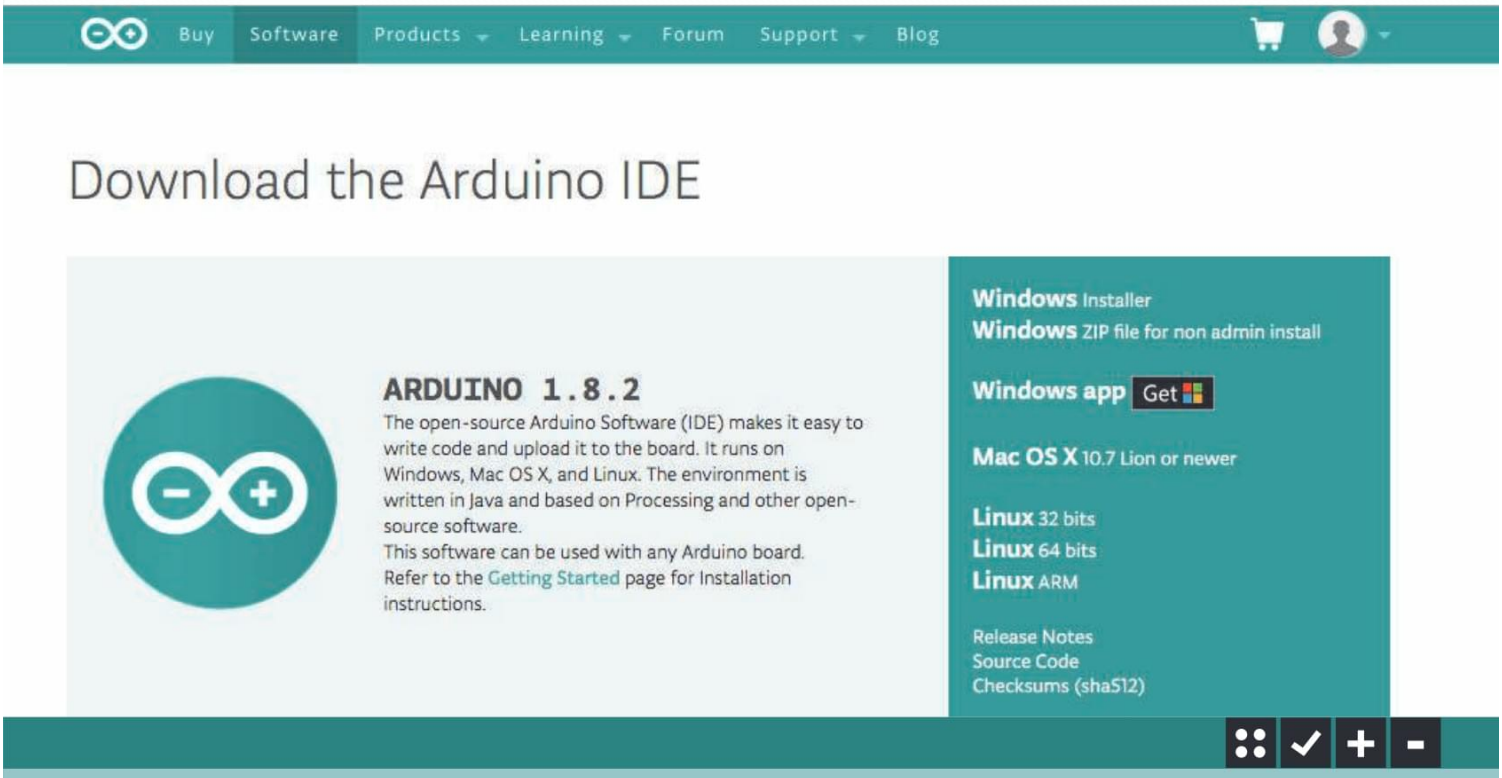
- **GIRO LIMITADO**: son los más comunes. Permiten una rotación de 180°, por lo que no pueden realizar una vuelta completa.
- **ROTACIÓN CONTINUA**: son capaces de girar 360°, es decir, pueden dar una vuelta completa. Son similares a un motor convencional, pero poseen las características adicionales del servomotor y por eso es posible controlar su posición y su velocidad de giro.

Resumen Capítulo 03

En este capítulo realizamos un repaso de los componentes esenciales que necesitamos para comenzar a trabajar con Arduino. En primer lugar, elegimos la placa de desarrollo, para esta obra utilizaremos Arduino UNO. Describimos detalladamente sus pines y las secciones que podemos encontrar en su estructura, así como también llevamos a cabo una prueba de conexión básica con la PC mediante el cable USB. Más adelante analizamos el funcionamiento de Arduino UNO y conocimos sus terminales digitales y, para completar el análisis de Arduino UNO, la comparamos con Arduino Leonardo, otra de las placas más utilizadas. Continuamos con la descripción de los elementos adicionales que necesitamos, por ejemplo, el protoboard, los cables de puente, el condensador, el diodo y los servomotores, entre otros. Para estos componentes entregamos descripciones detalladas y ejemplos de uso, así como también efectuamos algunas tareas paso a paso para utilizarlos.

Arduino IDE

04



Es hora de conocer la forma en que nos comunicaremos con nuestra placa de desarrollo: el Arduino IDE. Gracias a esta interfaz de software, contamos con todo lo necesario para escribir los códigos y enviarlos a la placa para que sean ejecutados, y de esta forma dar vida a nuestros primeros proyectos. En este capítulo aprenderemos a instalar y a utilizar este Entorno de Desarrollo Integrado.

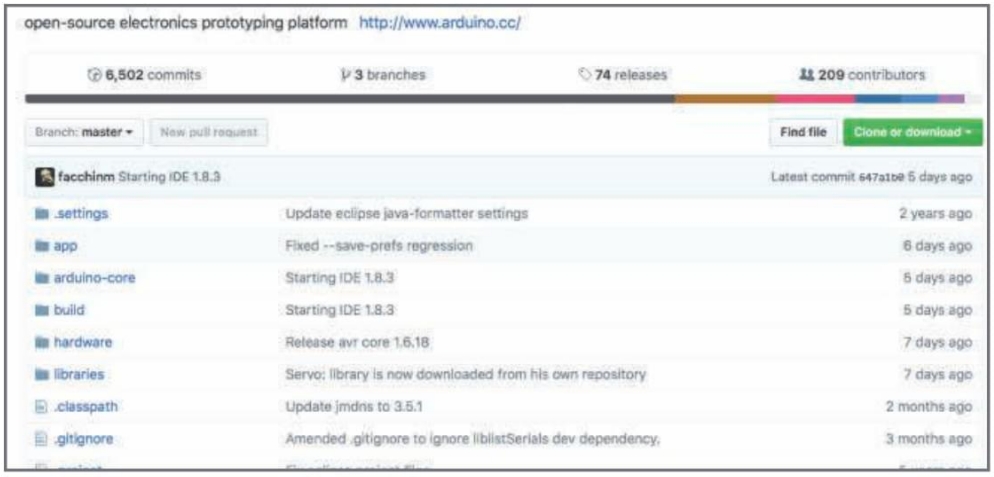
CARACTERÍSTICAS GENERALES

Como sabemos, un **IDE** (*Integrated Development Environment*) es un programa que se compone de una serie de herramientas que nos ayudan en tareas de programación. Existen IDEs que nos permiten trabajar con varios lenguajes de programación y plataformas, así como también algunos específicos, como Arduino IDE.

Un IDE se compone de un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI); en el caso de Arduino IDE, también nos ofrece las herramientas necesarias para que podamos cargar nuestros programas en la memoria flash de la tarjeta Arduino con la que estemos trabajando.

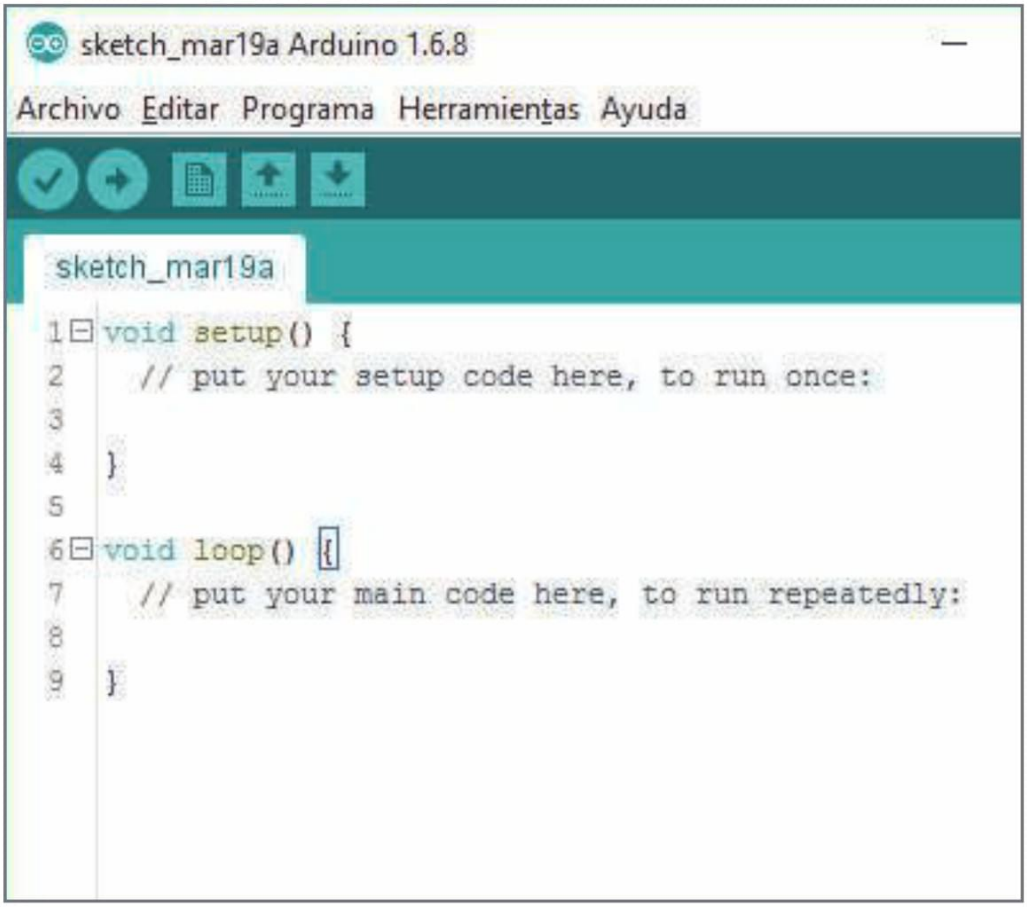
Entre las características más destacadas de Arduino IDE, encontramos las siguientes:

- **GRATIS Y LIBRE**: Arduino IDE se distribuye en forma gratuita, por lo que solo necesitamos acceder al sitio web oficial de la aplicación para descargar una copia instalable. También se trata de un software que se distribuye con una licencia libre, por lo que es posible acceder al código fuente del IDE y construir el instalador desde él o realizar las modificaciones que consideremos necesarias. Para los usuarios comunes, bastará con descargar el instalador adecuado para el sistema operativo y proceder con la instalación. El código fuente de Arduino IDE se encuentra disponible en la dirección **https://github.com/arduino/Arduino**.



En el sitio **https://github.com/arduino/Arduino** podemos descargar el código fuente del Arduino IDE. Solo debemos hacer clic en **Clone download**.

- **MULTIPLATAFORMA**: una de las ventajas de este IDE es que se trata de una aplicación multiplataforma, es decir, que puede ser instalado y utilizado en diferentes sistemas operativos, por ejemplo, Microsoft Windows, GNU/Linux o Mac OSX, entre otros. Para obtener el instalador adecuado, debemos visitar el sitio web oficial del IDE y, allí, elegir la opción que necesitemos.
- **INTERFAZ SENCILLA**: la interfaz de usuario de Arduino IDE es muy sencilla, aunque parezca una debilidad por las reducidas opciones que vemos al acceder por primera vez, pero, a medida que lo conozcamos más a fondo, veremos que nos proporciona todo lo que necesitamos y más aún. La disposición de las áreas de trabajo y los menús de opciones, junto a una consola de errores, se reúnen en la pantalla principal del IDE; en resumen, cada opción justo donde la necesitamos sin que la ventana principal ocupe toda la pantalla.



La interfaz del Arduino IDE es bastante sencilla; como vemos en esta imagen, la sección dedicada al código ocupa gran parte de la ventana.

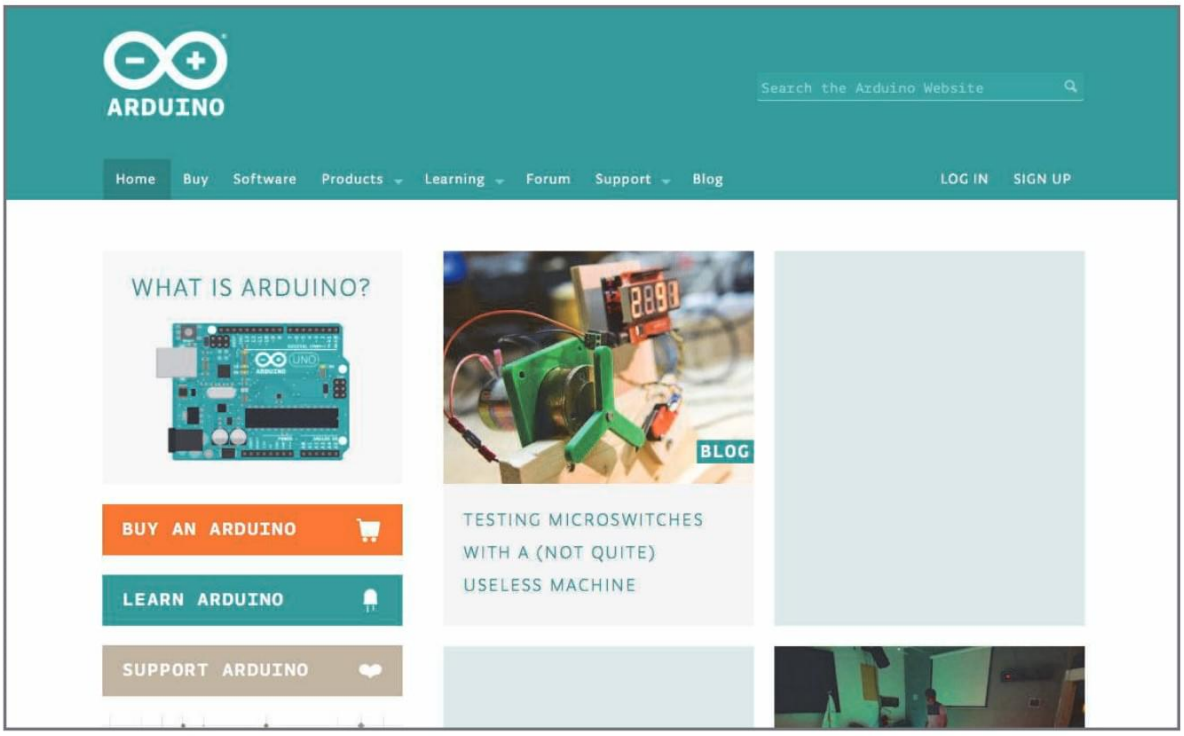
- **CARGAR PROGRAMAS EN ARDUINO**: gracias a las herramientas de Arduino IDE, podremos cargar los programas que escribamos directamente en la memoria flash de Arduino, en unos pocos pasos, sin necesidad de ejecutar complejos procedimientos. De esta forma, tendremos nuestra placa ejecutando un programa en un tiempo muy reducido.
- **CONFIGURACIÓN INICIAL**: la configuración inicial de Arduino IDE se realiza en pocos pasos y nos llevará un tiempo mínimo. Así tendremos las herramientas de programación listas para usar sin complicaciones. Más adelante, en este mismo capítulo revisaremos en detalle el proceso de configuración del IDE.
- **NUEVAS OPCIONES**: la última versión de este IDE nos propone algunas características novedosas, por ejemplo, detección automática de la placa conectada, información sobre memoria flash y SRAM ocupada por un sketch o proyecto, autoguardado al compilar y cargar un sketch, y carga de sketchs vía red para la placa Arduino Yún.

Dos Arduinos

Si profundizamos en los antecedentes históricos de Arduino, veremos que los cinco miembros fundadores crearon la empresa Arduino LLC, propietaria de la marca Arduino, que entregó diseños y software para la comunidad que lo quisiera utilizar. En simultáneo, existía Smart Projects Srl, empresa encargada de fabricar los productos Arduino. Más tarde Smart Projects Srl se renombró como Arduino Srl y registró la marca Arduino en Italia.

Como resultado de esto, existen dos sitios de Arduino: por un lado **www.arduino.cc** y, por otro, **www.arduino.org**. La primera dirección es la que corresponde al sitio creado por los cinco fundadores originales, en él se venden los productos y se entrega el IDE descargable, listo para programar para Arduino. La segunda dirección corresponde a Arduino Srl, y en ella también se ofrecen los productos Arduino y el IDE adecuado para desarrollar nuestros trabajo en Arduino con mayor celeridad y profesionalismo.

La importancia de esto radica en que tenemos a nuestra disposición dos IDEs que, aunque parecen similares, internamente presentan algunas diferencias. Para esta obra utilizaremos el IDE que se ofrece en **www.arduino.cc**.



Búsqueda en **www.arduino.cc el** IDE que nos ofrece el sitio.

Una de las consecuencias más importantes, para los usuarios finales, de esta división entre arduino.cc y arduino.org es que las placas arduino.org podrían no funcionar con el IDE de arduino.cc, y viceversa; aunque siempre se pueden realizar ciertas configuraciones para trabajar con la placa y el IDE que prefiramos.

De cualquier forma, las placas antiguas no deberían presentar mayores complicaciones, aunque es probable que las nuevas placas se vayan alejando de uno u otro IDE, y de esta forma cueste más trabajo desarrollar con un IDE que no la soporta completamente.

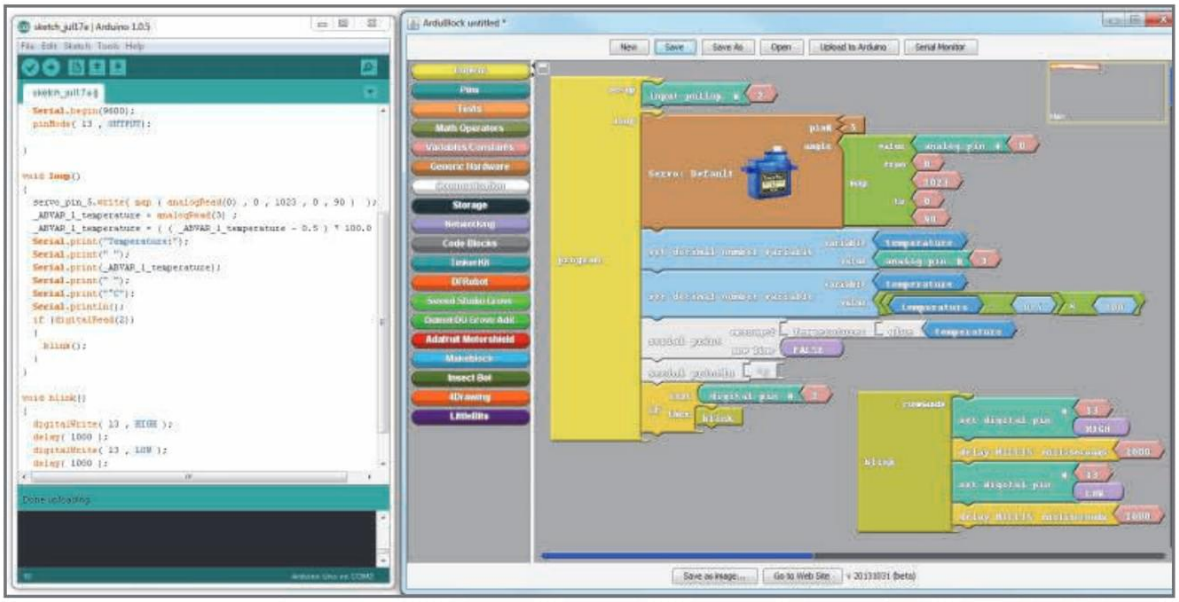
Cada uno de estos IDEs puede tener versiones diferentes de las librerías incluidas, y esto daría como resultado problemas a la hora de utilizar un sketch específico. Por otra parte, también podríamos encontrarnos con mensajes que advierten sobre el uso de una placa no certificada, ya que arduino.cc y arduino.org tienen su propio identificador de USB.

"Tanto arduino.cc como arduino.org son considerados oficiales, por lo tanto existen dos IDE que con el tiempo se van diferenciando."

IDEs alternativos

Además de los IDEs oficiales para trabajar con Arduino, existen algunas alternativas interesantes. Entre ellas, encontramos algunas opciones de programación visual que nos permiten trabajar con bloques para armar el programa sin necesidad de escribir código.

Entre las alternativas existentes encontramos **Ardublock** (**http://blog.ardublock.com**), sistema de **bq** (**http://bitbloq.bq.com**) y **S4A**, este último es una variación de Scratch adecuada para programar Arduino en forma sencilla. Nos ofrece una serie de bloques nuevos listos para trabajar con sensores y actuadores, que se encuentren conectados a una placa Arduino.

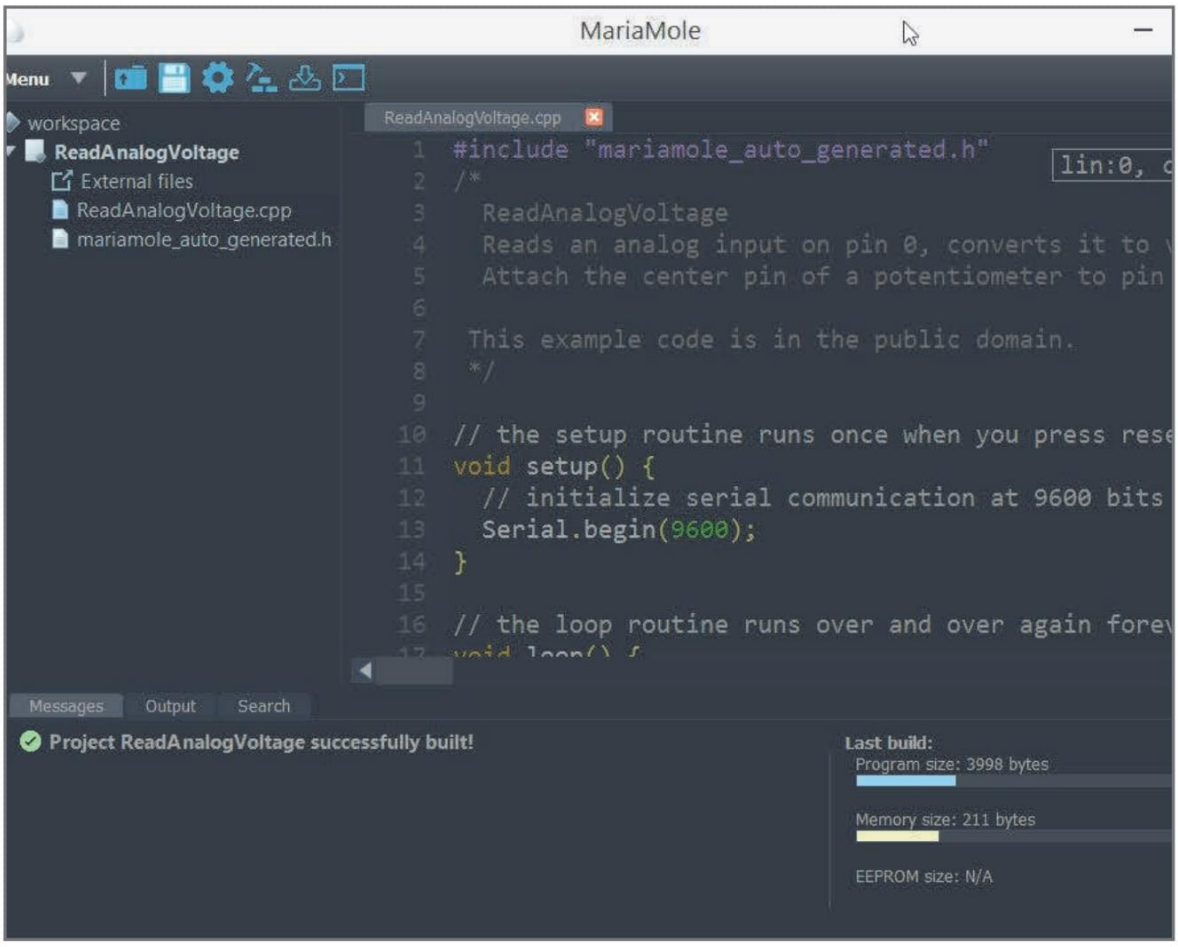


Ardublock es un sistema sencillo de programación visual, adecuado para quienes deseen iniciarse en la programación de Arduino en forma gráfica.

Otra de las opciones disponibles para desarrollar código para Arduino es utilizar un IDE en línea, de esta forma no será necesario instalar una aplicación en la computadora. **Arduino Create** es el IDE en la nube lanzado por **www.arduino.cc**; nos propone una interfaz de programación que es ejecutada directamente en un navegador web y la encontramos en la dirección **https://create.arduino.cc**.

Algunos IDEs alternativos instalables y opciones que nos permiten programar para Arduino son los siguientes:

- **Stino (https://github.com/Robot-Will/Stino)**: se trata de un plugin para el editor Sublime Text. Nos permite desarrollar para Arduino de manera fácil, ya que ofrece acceso muy rápido a toda la estructura del proyecto, posee autocompletado y otras características importantes.
- **Atmel Studio (www.atmel.com/microsite/atmel_studio6)**: se basa en Visual Studio, y es adecuado para trabajar con Arduino y con los distintos microprocesadores compatibles de Atmel. Solo es compatible con Microsoft Windows.
- **MariaMole (http://dalpix.com/mariamole)**: se trata de un editor de Arduino bastante ligero, por lo que puede ser ejecutado en computadoras con recursos limitados. Permite trabajar con opciones de ordenación por proyectos, ofrece variadas mejoras visuales y se ejecuta en diferentes sistemas operativos.



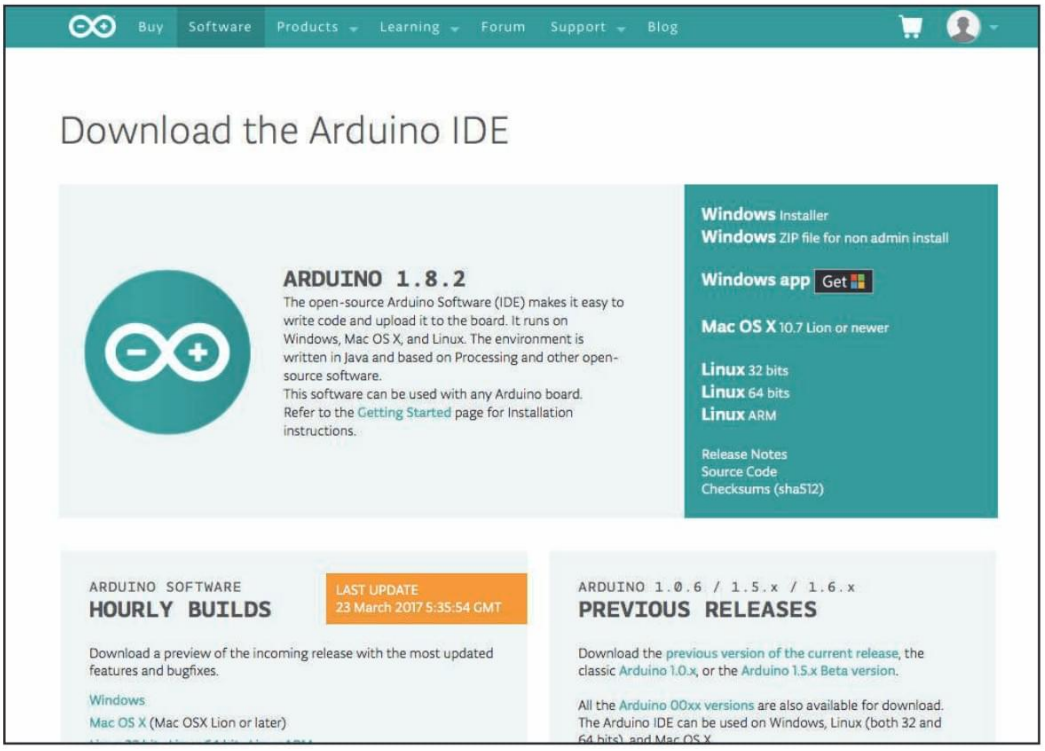
MariaMole es un excelente IDE alternativo para trabajar con Arduino; se destaca por su bajo consumo de recursos, y también por ofrecer una interfaz sencilla y limpia.

INSTALACIÓN DEL IDE

Antes de conocer y trabajar con el Arduino IDE, debemos realizar las tareas que nos permitirán instalarlo. Como sabemos, se trata de una herramienta multiplataforma, por lo que podremos trabajar en diversos sistemas operativos. En primer lugar, revisaremos la forma en que debemos instalar el IDE en un sistema Windows, a diferencia de ellos, Arduino IDE no solo es un entorno sencillo y claro para ser utilizado por principiantes con opciones avanzadas para profesionales, también, es una opción multiplataforma que no solo nos permite trabajar en Windows, sino, que también considera procedimientos para instalar sobre Mac OSX y GNU/Linux.

Paso a paso: Instalar Arduino IDE en Windows

01 Para comenzar ingrese en la dirección **www.arduino.cc**. Haga clic en el enlace denominado `Software`, que se encuentra en la barra superior de opciones. En la ventana que se presenta, baje hasta la sección Arduino IDE y haga clic sobre `Windows Installer`.

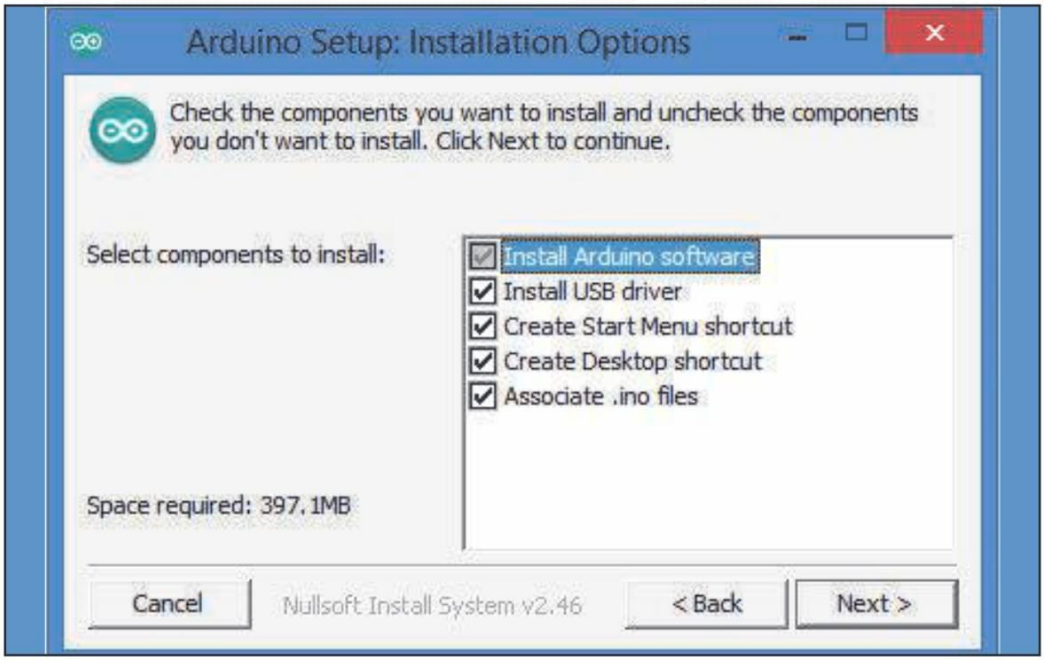


02

Una vez que la descarga haya finalizado, haga doble clic sobre el archivo adecuado y espere mientras se inicia el asistente de instalación. En la primera pantalla será necesario aceptar el acuerdo de licencia, para eso presione `I Agree`.

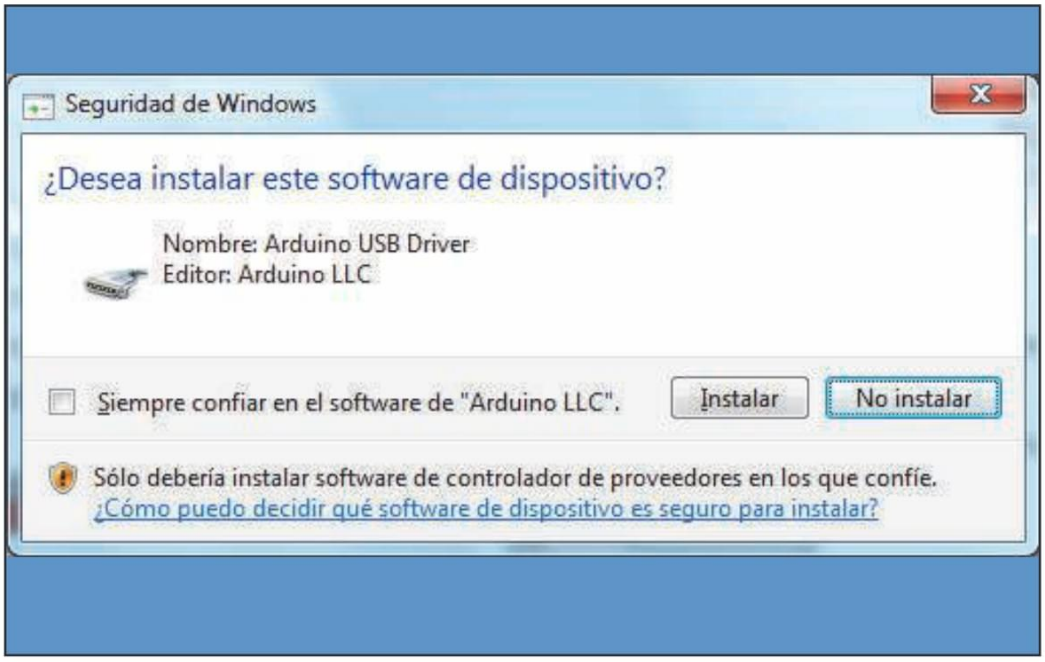
03

Seleccione los componentes que serán instalados; es recomendable marcar, al menos, las opciones `Install Arduino Software`, `Install USB driver`, y `Associate ino files`, luego presione el botón `Next`.



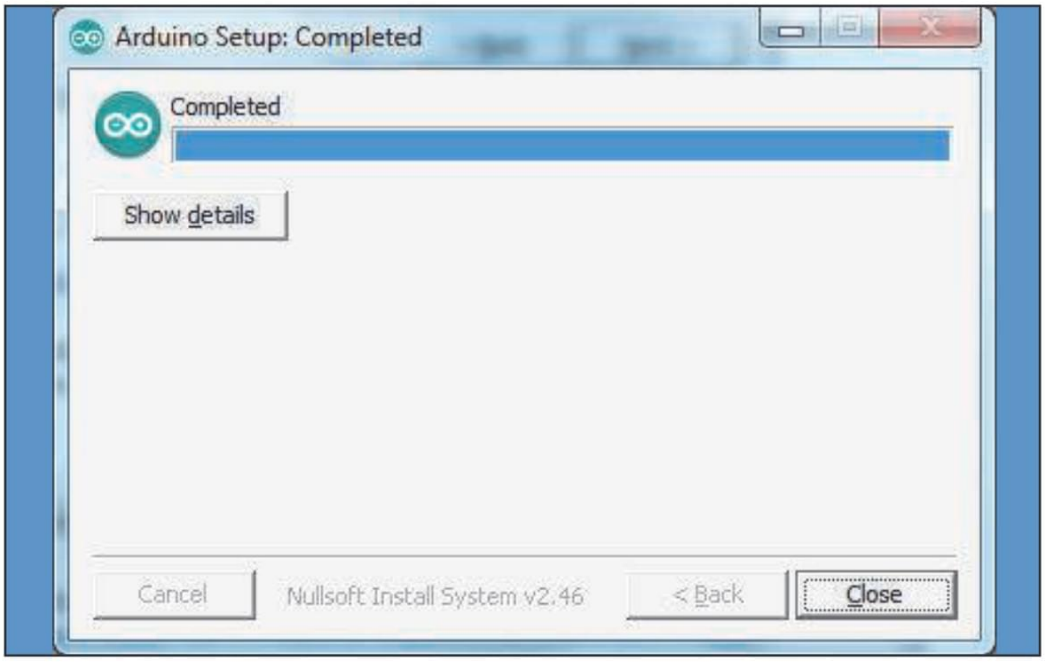
04

Es posible que aparezca una ventana que indica que se instalarán algunos drivers, acepte que se instalen; luego de esto el proceso continuará adelante.



05

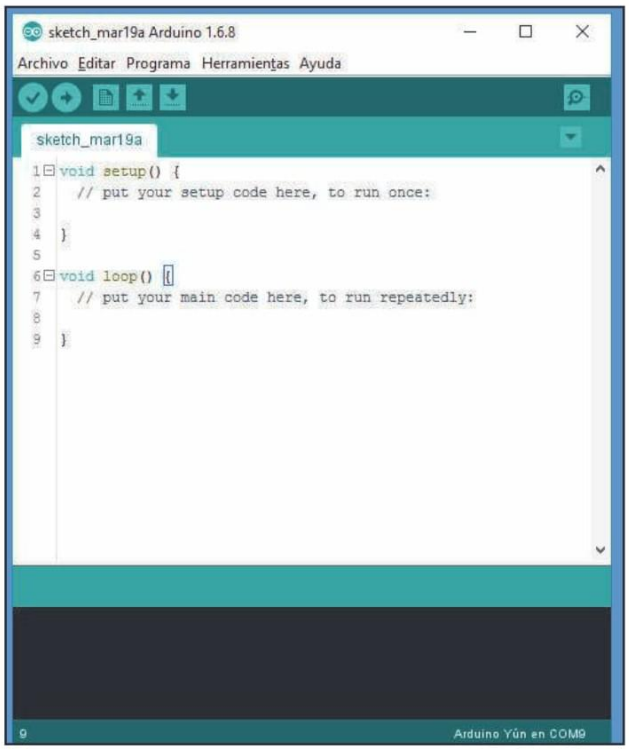
El proceso de instalación solo tardará un momento, podrá ver su avance gracias a la barra de la ventana. Cuando este procedimiento se complete, haga clic sobre el botón `Close`.



06

En este momento, el IDE de Arduino ya se encontrará instalado en la computadora. Es necesario saber que, en las nuevas versiones del IDE de Arduino, no se necesita instalar los drivers pues vienen integrados en el IDE. Inicie el software y verá su pantalla de carga.

07



Una vez cargado, la apariencia del IDE será similar a la que se ve en la imagen. Más adelante, se explicarán las partes principales que componen esta aplicación y cómo se deben utilizar.

Si deseamos utilizar el IDE de Arduino en otro sistema operativo, debemos acceder a **www.arduino.cc**, hacemos clic sobre **Software** y elegimos la descarga que deseemos. Por ejemplo, el instalador para Mac OSX, cuyo peso comprimido es de aproximadamente 158 MB.

Si utilizamos el navegador web Safari, el archivo será automáticamente descomprimido; como hacemos con todas las aplicaciones en Mac OSX es necesario copiar el programa en la carpeta de **Aplicaciones**.

Con el IDE de Arduino instalado, solo resta iniciarlo. Una vez que el programa se cargue, veremos la ventana principal y podremos comenzar a trabajar.

La interfaz principal de Arduino IDE para Mac OSX es similar a la que encontramos en la versión para sistemas Windows o GNU/Linux.

ENTORNO DE TRABAJO

El entorno de trabajo del IDE de Arduino es visualmente sencillo, se compone de algunas secciones bien diferenciadas donde encontraremos las herramientas y las opciones para desarrollar, un apartado donde escribir el código, y también espacios donde se muestran los mensajes y los errores que se presenten. En la próxima **Guía visual** se muestran las principales secciones del IDE de Arduino.

En la parte superior de la interfaz principal del IDE, veremos la barra de menús, en ella se encuentra el acceso a las herramientas y opciones que utilizaremos a través de este libro.

ARCHIVO

En este menú se encuentran las opciones relacionadas con la gestión de los archivos, desde aquí podremos guardar, abrir, acceder a los ejemplos y también a los proyectos recientes. Otras de las opciones disponibles se relacionan con la impresión y la configuración de páginas.

EDITAR

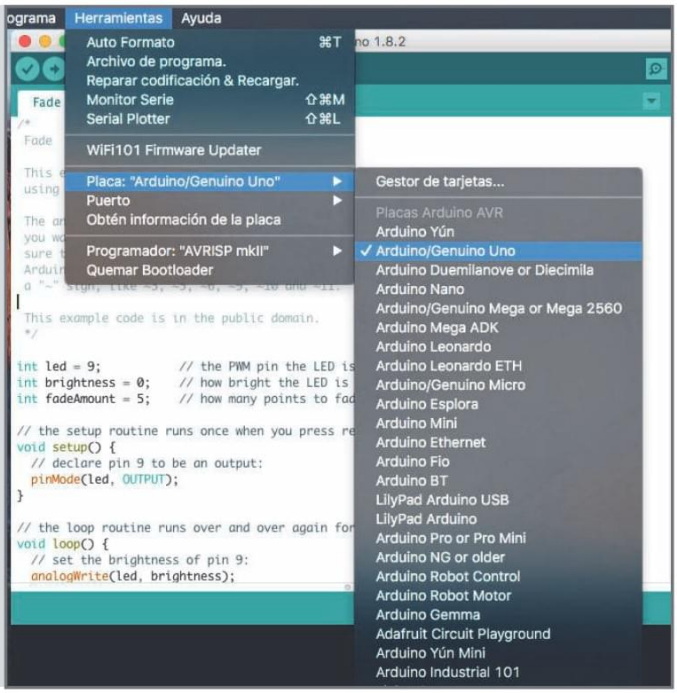
En este menú se encuentran opciones relacionadas con la gestión de los códigos con los que trabajamos. Es posible deshacer la última acción, copiar y pegar una porción de código, copiar directamente al foro o copiar como HTML, también podemos realizar búsquedas, comentar una porción del código o modificar la sangría.

PROGRAMA

En este menú se reúnen las opciones adecuadas para verificar nuestro código, subirlo a la placa Arduino, exportar binarios compilados, acceder a la carpeta del programa o incluir las librerías necesarias.

HERRAMIENTAS

Entre las herramientas que encontramos en este menú, se hallan el autoformato o el acceso al monitor serie, también podemos seleccionar la placa con la que trabajaremos. Para realizar esta última tarea, disponemos de una extensa serie de posibilidades, entre las que se encuentra Arduino UNO, la placa que usaremos en esta obra.

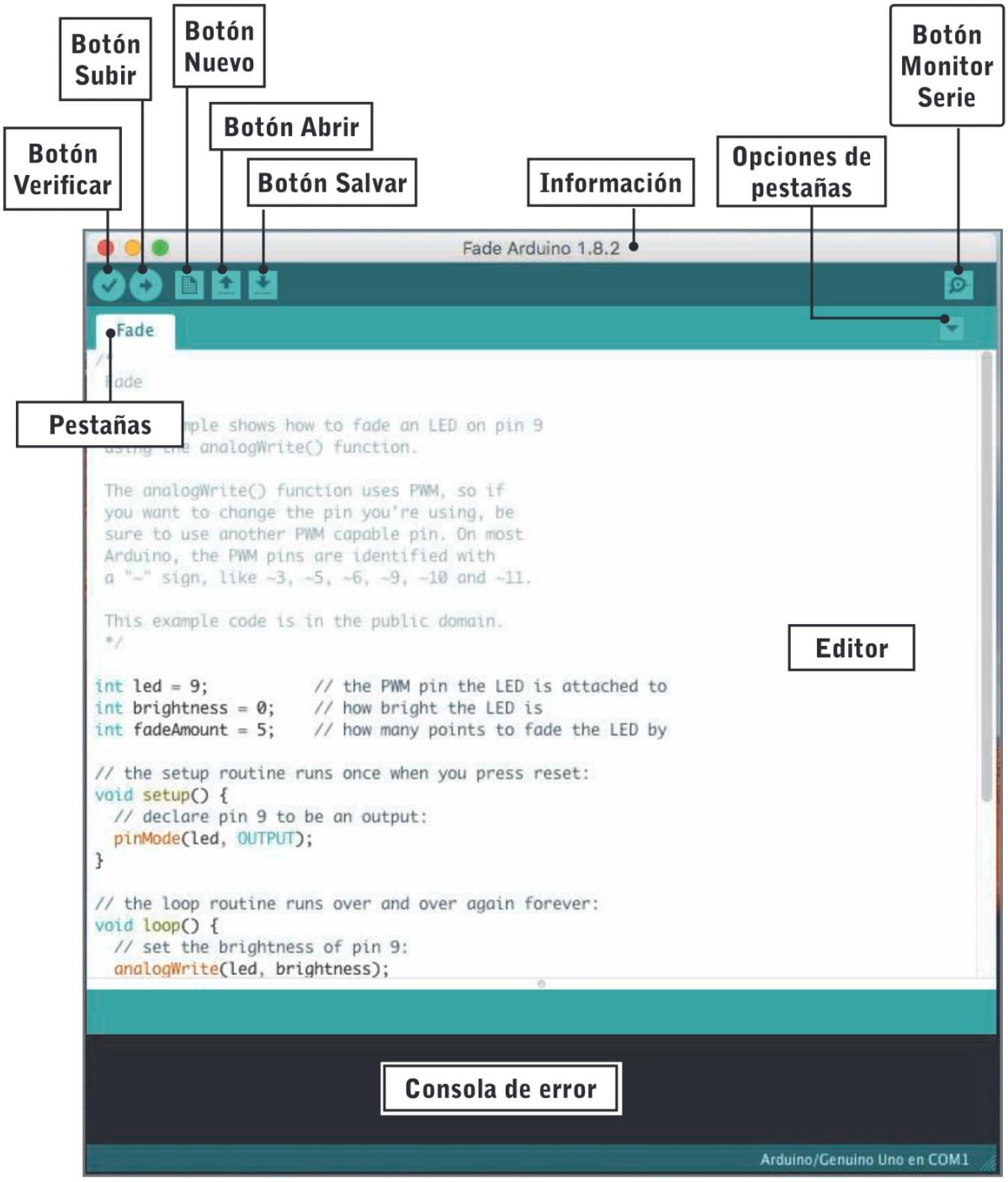


Podremos elegir la placa Arduino con la que trabajaremos en el IDE, en este caso, se trata de Arduino UNO.

AYUDA

El menú de ayuda nos entrega consejos y referencias de uso para dar los primeros pasos en la programación de Arduino. También posee un acceso a las preguntas frecuentes sobre el uso del IDE y un enlace directo a Arduino.cc.

Guía visual: Interfaz principal de Arduino IDE



Información En esta barra veremos el nombre del proyecto en el que estamos trabajando, así como también la versión del IDE que hemos instalado.

Botón Verificar Si hacemos clic en este botón, el IDE verificará el programa o sketch que hemos escrito; si se encuentran problemas, los veremos en la consola de error, de lo contrario, la misma consola nos entregará información relevante sobre el programa verificado.

Botón Subir Mediante este botón, podemos subir el programa ya verificado a la placa Arduino. Si presionamos lo presionamos sin tener la placa Arduino conectada a la computadora, se presentará un problema.

Botón Nuevo Este botón creará un nuevo proyecto en forma inmediata, veremos otra ventana donde tendremos acceso a todas las herramientas y opciones disponibles para trabajar.

Boton Abrir Al hacer clic sobre este botón, se desplegará un menú que nos permite abrir un proyecto almacenado con anticipación. También nos presenta una serie de sketchs de prueba listos para cargar y utilizar.

Boton Salvar Mediante este botón, podemos guardar el trabajo que hemos realizado hasta este momento.

Botón Monitor Serie Nos muestra los datos que son enviados por Arduino mediante el puerto serie; además, nos permite enviar a Arduino utilizando el puerto serie. Debemos tener en cuenta que existen alternativas al monitor serie, en algunas ocasiones puede ser necesario acudir a ellas pues la herramienta incluida en Arduino IDE es muy sencilla, aunque para usuarios principiantes será suficiente.

Pestañas En esta sección se presentan las pestañas abiertas, estas corresponden a los códigos en los que estamos trabajando. Para activar una u otra pestaña, es necesario hacer clic sobre su nombre.

Opciones de pestañas Si desplegamos este menú, veremos algunas opciones que nos permiten realizar tareas sobre las pestañas abiertas. Entre las opciones disponibles, se encuentran: Nueva pestaña, Renombrar, Pestaña anterior y Pestaña siguiente.

Editor En este apartado podemos escribir o editar el código que corresponde al programa que deseamos cargar. Se trata de la sección principal del IDE, pues la utilizaremos para escribir las acciones por medio del lenguaje de programación adecuado. Más tarde lo cargaremos en la memoria flash de Arduino.

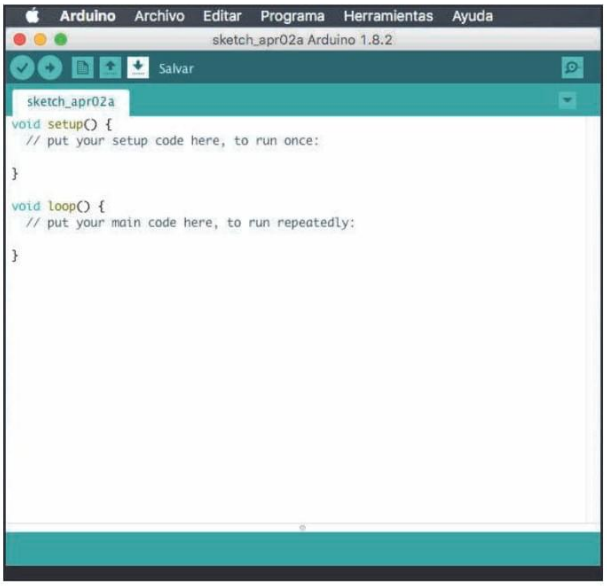
Consola de error En este pequeño apartado se mostrarán los posibles errores que puedan presentarse al verificar nuestro código, debemos poner atención en esta sección pues nos entregará la información que necesitamos para corregir los posibles problemas.

Configuración inicial

Con el IDE instalado, lo primero que debemos hacer es configurar el entorno de trabajo. Aunque no se trata de una tarea imprescindible, es una buena idea completar los pasos que mencionaremos pues nos permitirán facilitar la edición de los programas que desarrollemos, ya que luego de completar esta configuración inicial, podremos acceder a toda la información que puede otorgarnos el IDE. Para realizar esta configuración, seguiremos los pasos que mencionamos a continuación.

Paso a paso: Configuración inicial del IDE

01

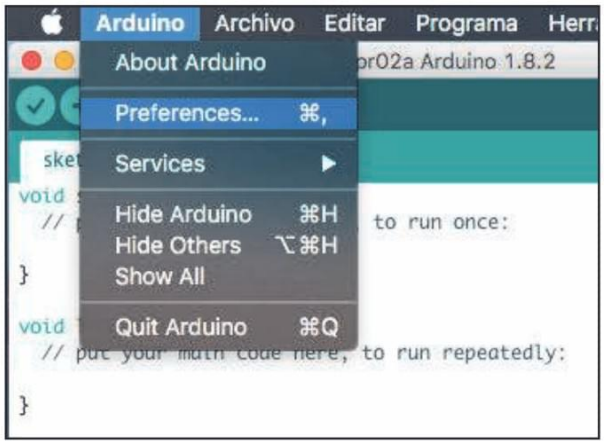


Como primera instancia, inicie el IDE de Arduino que instaló siguiendo las instrucciones expuestas en una sección anterior. Cuando aparezca la interfaz principal del IDE, estará listo para comenzar el proceso de configuración inicial.

Configuración adicional

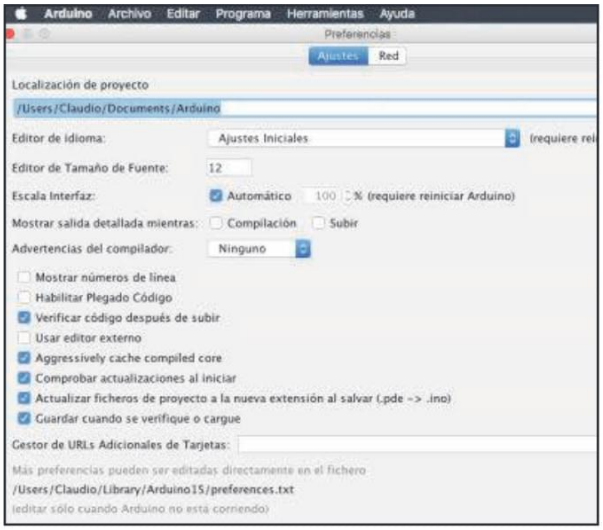
Aunque la configuración inicial del IDE que presentamos en el Paso a paso debería bastar para cualquier usuario principiante, tengamos en cuenta que existen opciones de configuración avanzadas que están reservadas para usuarios con mayor experiencia. Para acceder a estas opciones avanzadas, debemos abrir el archivo preferences.txt, que se ubica en la carpeta AppData/Local/Arduino15. Hay que tener mucho cuidado al editar las opciones que están contenidas en este archivo y, tal como se advierte en el IDE, realizar las modificaciones solo cuando el programa no se encuentra ejecutado.

02



Si ha iniciado el IDE en un sistema Windows, tendrá que hacer clic en el menú Archivo, luego elegir la opción Preferencias. Por otra parte, si se encuentra en un sistema Mac OSX, tendrá que hacer un clic en el menú Arduino, y, posteriormente, otro clic sobre Preferencias.

03



En este punto, verá la ventana de preferencias del IDE, la cual se divide en dos pestañas: Ajustes y Red. Para este proceso de configuración, utilizará algunas opciones que se encuentran en la pestaña Ajustes.

04

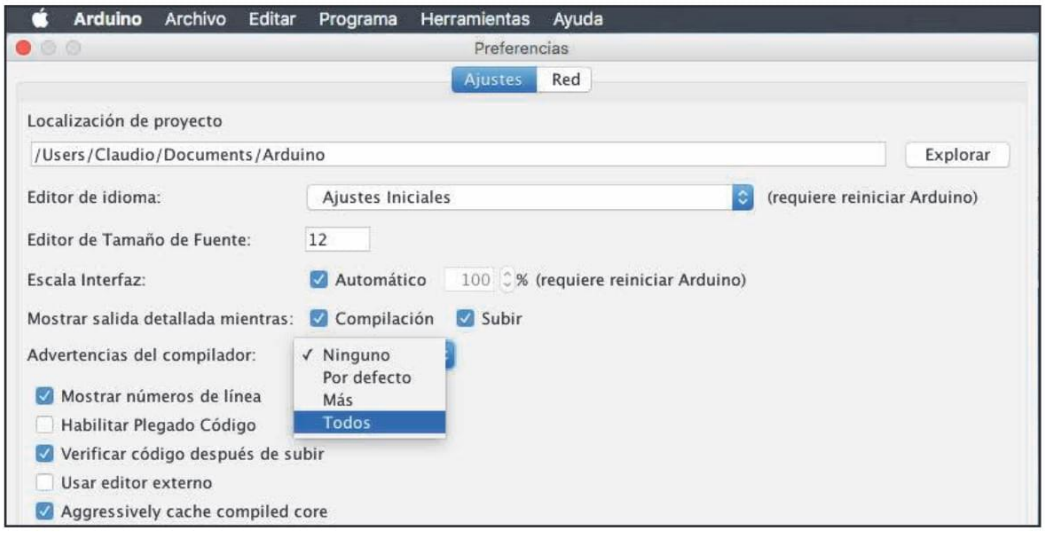
Lo primero que hará será buscar la casilla que corresponde a la opción Mostrar números de línea. Esta opción le permitirá ver los números que corresponden a cada línea de los programas que realice. La importancia de esto radica en que será más sencillo ubicar una línea específica, sobre todo cuando trabaje con códigos extensos.

05

Para continuar, busque la opción denominada `Mostrar salida detallada para`. Junto a esta opción se presentan dos alternativas: `Compilación` y `Subir`. Marque ambas casillas, de esta forma tendrá acceso a información detallada tanto cuando realice la compilación del programa como cuando lo suba a Arduino; esta salida puede incluir datos importantes para ubicar algún problema o dificultad que pudiera presentarse en el código.

06

Ahora vaya a la opción denominada `Advertencias del compilador`, despliegue el menú que se encuentra a su lado y elija la opción `Todos`. La opción predeterminada es `Ninguno`, que puede ser útil cuando tenga experiencia en el manejo del IDE, pero si recién comienza en la programación con Arduino, bien vale la pena tener a su disposición toda la información que pueda ayudarlo.

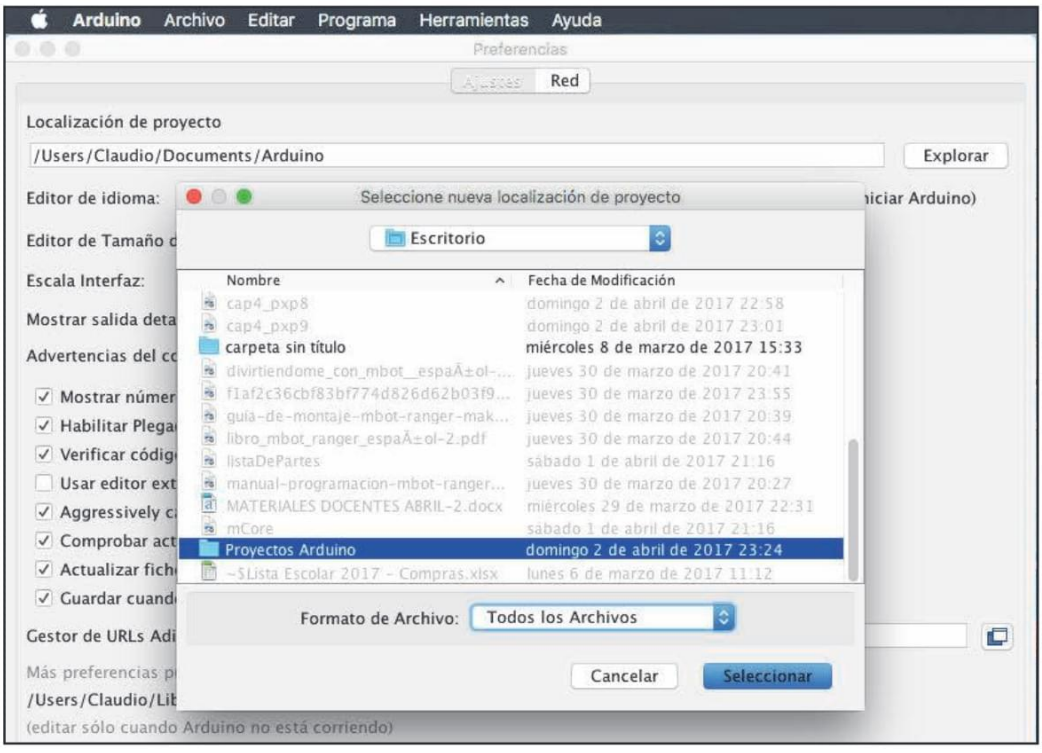


07

La siguiente opción que marcará es el plegado de código. Esto le facilitará el trabajo cuando maneje códigos extensos, pues le permitirá agrupar y plegar ciertos segmentos para tener el programa más organizado, que sea más entendible y sin que deba navegar por mucho tiempo para llegar a la línea de código que necesita.

08

Para finalizar la configuración inicial del IDE, elija la ruta que utilizará para almacenar sus proyectos. En forma predeterminada se encontrará con una ubicación como la siguiente: `Users/Claudio/Documents/Arduino`, pero puede elegir la ruta que más le guste, por ejemplo, `Escritorio/Proyectos Arduino`.



Una vez que hayamos aplicado todas las opciones que mencionamos en el **Paso a paso** anterior, será necesario hacer clic sobre el archivo **Aceptar**, de esta forma las opciones de configuración que marcamos se almacenarán y estarán disponibles para todos los proyectos que iniciemos de aquí en adelante.

Siempre podemos volver a la sección de preferencias y desactivar aquellas opciones que nos complican la tarea de escribir código o que no nos convencen del todo.

Librerías

Una **librería** es un código escrito por terceros que puede ser utilizado en nuestro programa o sketch. En realidad, se trata de trozos de código que han sido escritos para que sea más fácil realizar la interconexión de elementos, tales como sensores, pantallas o módulos electrónicos.

Por ejemplo, si quisiéramos utilizar un sensor capacitivo touch, deberíamos escribir un código como el siguiente:

```
  *sOut&= ~sBit;
   *rReg&= ~rBit;
   *rOut&= ~rBit;
   *rReg |= rBit;
   *rReg&= ~rBit;
   *sOut |= sBit;
  interrupts();

  while ( !(*rIn&rBit) && (total <CS_Timeout_Millis) ) {
total++;
  }
  if (total >CS_Timeout_Millis)
  return -2;

  noInterrupts();

  *rOut |= rBit;
  *rReg |= rBit;
  *rReg&= ~rBit;
```

```
*rOut&= ~rBit;
*sOut&= ~sBit;
interrupts();

while ((*rIn&rBit) && (total <CS_Timeout_Millis)) {
 total++;
}
if (total >= CS_Timeout_Millis)
return -2;
else
return 1;
}
```

Ahora bien, podemos reemplazar esto por un código más sencillo si hacemos uso de la librería denominada **CapacitiveSensor**, entonces, solo necesitamos el siguiente código:

```
senseReading = myCapPad.capacitiveSensor(30);
```

En este caso, vemos que la orden **myCapPad.capacitiveSensor()** realiza el trabajo pues la variable **senseReading** contiene el valor que recibe el sensor capacitivo touch. De esta forma, podemos entender que las librerías son capaces de convertir tareas complicadas en accesibles y fáciles, así optimizaremos el tiempo sin necesidad de programar algo que está listo para agregar a nuestros proyectos en forma de librería.

Lo interesante es que el IDE de Arduino incorpora algunas librerías, gracias a ellas es posible mejorar la creación de los programas o sketchs.

Para simplificar esta idea, mencionaremos que una librería no es más que una colección de funciones que podemos incluir de una

"Las librerías son conjuntos de funciones que nos facilitan la tarea de programar o acceder a tareas específicas."

forma bastante sencilla y rápida en el sketch, y que es capaz de proporcionarnos funciones específicas. Por ejemplo, si incluimos la librería de cristal líquido, podremos usar fácilmente una pantalla LCD.

En Arduino existen tres tipos de librerías: base, estándar y contribuciones.

Librería base

La **librería base** de Arduino forma parte del IDE, su propósito es ocultar la complejidad que relaciona el trabajo con el microprocesador; gracias a esta librería podemos realizar tareas en forma sencilla sin necesidad de programar cientos de línea de código.

Esta librería base es la que diferencia a Arduino si lo comparamos con el trabajo que debe realizarse para programar microprocesadores tradicionales. Una de las ventajas de la librería base o core es que permite efectuar tareas, como la lectura de datos de una de las entradas o la escritura de datos en una de las salidas, en forma muy sencilla y simple de ejecutar.

Un ejemplo claro es la tarea de leer el valor de un pin digital, para lograrlo solo necesitamos utilizar la función **digitalRead**.

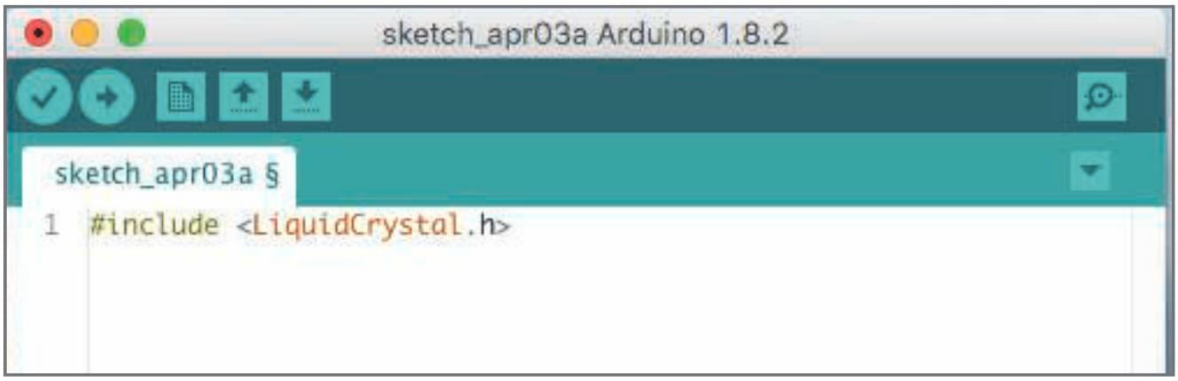
Librería estándar

Las **librerías estándares** son aquellas que se incluyen en el IDE de Arduino, porque los desarrolladores consideraron que se trataba de funciones necesarias para muchos proyectos, por esta razón se agregan en forma automática con la instalación del IDE, pero no son parte de la librería base o core.

Estas librerías no serán incluidas en forma automática en un sketch, al contrario, tendremos que cargarlas una a una, así nos aseguramos de utilizar solo aquellos recursos que se necesiten. Incluir una librería en un sketch es una tarea sencilla, solo debemos utilizar la orden **#include** seguida del nombre de la librería, por ejemplo:

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

Este código incluirá la librería **LiquidCrystal**, encargada de proporcionar las funciones necesarias para trabajar con pantallas LCD.



■ Para incluir una librería utilizamos **`#include`**, debemos incluir el nombre de la librería entre corchetes menor/mayor (< y >), y la línea no terminará con punto y coma (**`;`**) como debemos hacerlo en las demás líneas de código.

Como vemos, las librerías estándares pueden sernos de mucha utilidad en los proyectos en los que trabajemos, ya que cada una incorpora una serie de funciones relacionadas con un objetivo específico. En la siguiente tabla, analizaremos las diferentes librerías estándares que se incluyen en Arduino.

Contribuciones

Por último, tenemos las librerías conocidas como **contributed**, se trata de librerías creadas por usuarios para resolver problemas o ser utilizadas en situaciones específicas, pero que no son parte de las librerías estándares que conocimos en la sección anterior.

Estas librerías pueden ser agregadas a la colección de librerías estándares si demuestran ser útiles, y son requeridas para muchos proyectos.

Podemos encontrar estas librerías en diversos sitios de internet, aunque también existen algunas disponibles en el sitio web de Arduino.

Generalmente las librerías se componen de los siguientes elementos, comprimidos en un archivo .ZIP: un archivo .cpp (código de C++), un archivo .h o encabezado de C, un archivo Keywords.txt, un archivo readme con información adicional sobre la librería para el desarrollador y un directorio con un sketch de ejemplo.

En el próximo **Paso a paso** analizaremos en detalle el proceso de instalación de una nueva librería en Arduino.

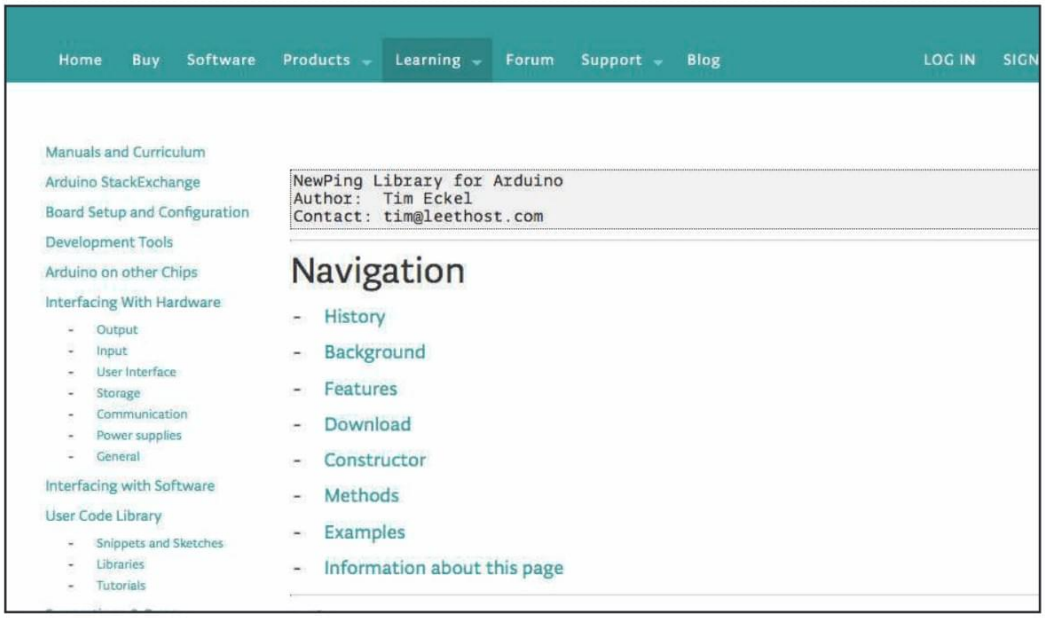
LIBRERÍAS ESTÁNDARES

LIBRERÍA	DESCRIPCIÓN
ArduinoTestSuite	Nos entrega los métodos estándares y las funciones que se pueden utilizar para probar los sketchs antes de cargarlos en la placa Arduino. Esto es importante pues nos permite asegurarnos de que el sketch funcione según lo que hemos previsto y, de esta forma, nos ahorraremos problemas al probarlos directamente en la placa.
EEPROM	Esta librería nos permite acceder a la EEPROM de Arduino mediante las funciones read y write.
SD	Esta librería entrega funciones básicas para que la placa Arduino interaccione con las tarjetas SD. Es necesario utilizarla con precaución, pues consume mucha memoria del programa.
Ethernet	La librería Ethernet permite configurar Arduino como servidor para recibir conexiones de clientes, o como cliente, para conectarse a servidores.
Firmata	Permite utilizar los protocolos de comunicación serie para relacionar la PC con Arduino. Por ejemplo, podremos controlar servos, motores o pantallas desde una PC a través de Arduino.
LiquidCrystal	Nos permite interactuar con pantallas LCD, por ejemplo, para mostrar datos del GPS, mensajes de situación del sistema u otra información útil para el usuario.
Servo	Nos permite controlar hasta 12 motores servos con Arduino estándar, y 48 con Arduino Mega.
Stepper	Nos permite establecer la velocidad de la rotación del motor, el número de pasos que queremos dar y la dirección de estos pasos.
Wire	Esta interfaz se utiliza para comunicar a baja velocidad con una amplia gama de dispositivos y componentes, por ejemplo, relojes de tiempo real. Esta librería nos permite interactuar como maestro o esclavo.
SoftwareSerial	Esta librería es la solución cuando necesitamos más puertas seriales para nuestros proyectos.

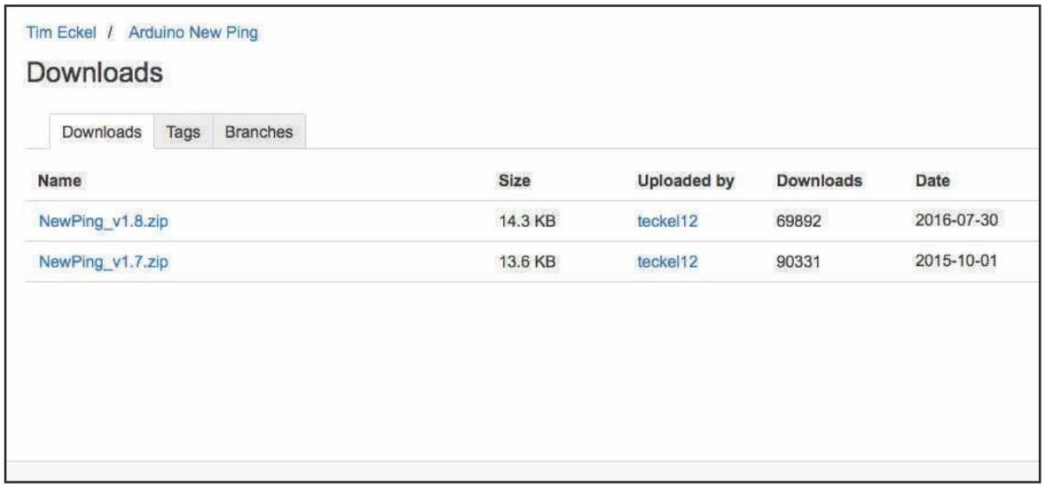
EJEMPLOS DE FUNCIONES INCORPORADAS
ATS_begin (inicia el proceso de test), ATS_end (termina el proceso de test), ATS_Test_DigitalPin (testea un pin de entrada digital dado), ATS_Test_PWM (testea la salida PWM), ATS_Test_AnalogInput (testea la entrada analógica), ATS_Test_EEPROM (testea la EEPROM).
Read (lee el valor de un byte almacenado en una posición de la EEPROM), Write (escribe un valor en una posición de la EEPROM).
Begin (inicializa la librería y la tarjeta SD), exists (verifica la existencia de un fichero o directorio en la tarjeta), mkdir (crea un directorio en la tarjeta), rmdir (elimina un directorio en la tarjeta), remove (elimina un fichero de la tarjeta).
Begin (inicializa la librería y configura los parámetros de red), localIP (devuelve la dirección IP local), dnsServerIP (devuelve la dirección DNS del servidor), server (crea un servidor), begin (comienza a escuchar posibles peticiones de conexión).
Begin (inicializa la librería Firmata), printVersion (envía versión del protocolo a la PC), setFirmwareVersion (establece la versión del firmware).
Begin (establece las dimensiones en filas y columnas de la pantalla LCD), LiquidCrystal (inicializa la librería y determina los pines usados para comunicar con la pantalla LCD), print (visualiza datos en la pantalla LCD).
Attach (asigna el servo a un pin), Attached (verifica que el servo está conectado al pin), Detach (desconecta el servo del pin), Read (lee el ángulo del servo).
Stepper (inicializa la librería Stepper y establece el número de pasos por revolución), setSpeed (establece la velocidad de rotación del motor en revoluciones por minuto), step (gira el motor el número de pasos indicado).
Begin (inicializa la librería Wire y conecta el Arduino al bus I2C como maestro o esclavo), requestFrom (pide datos del esclavo al maestro), beginTransmission (prepara la transmisión de datos), Send (envía datos del esclavo al maestro o pone en cola bytes para la transmisión de maestro a esclavo).
Begin (activa la puerta y establece la velocidad de transmisión en baudios), Available (comienza a usar la puerta), sListening (devuelve la puerta active en este momento), listen (escucha a esa puerta y la activa).

Paso a paso: Instalar y utilizar una librería

01 En primer lugar, ubique en internet la librería que necesita, para este ejemplo utilizará la librería NewPing, que lo ayudará en el manejo de varios sensores ultrasónicos. Esta librería en particular se encuentra en la dirección **http://playground.arduino.cc/Code/NewPing**. Haga clic sobre Download, luego presione sobre el enlace DownloadNewPinglibrary y descargue la librería a la computadora.

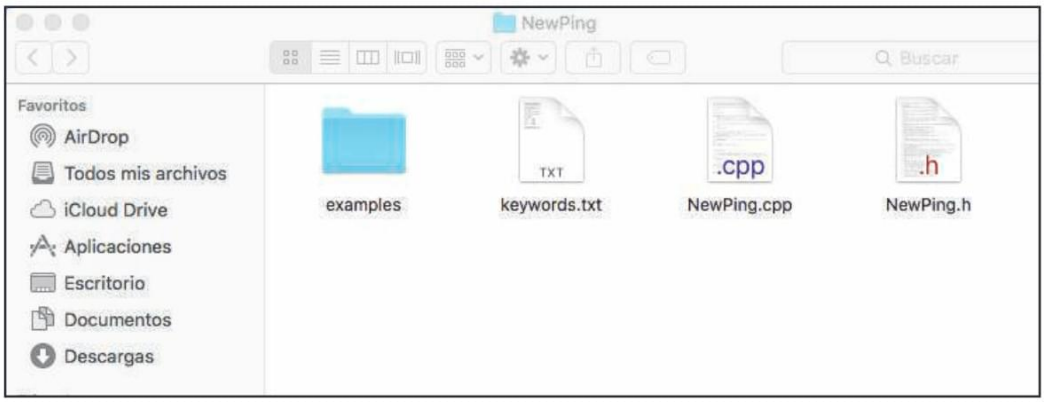


02 Para algunas librerías, encontrará diferentes opciones de descarga, en este ejemplo se ve que se encuentran disponibles las versiones 1.7 y 1.8 de NewPing Library, seleccione la versión más actualizada.



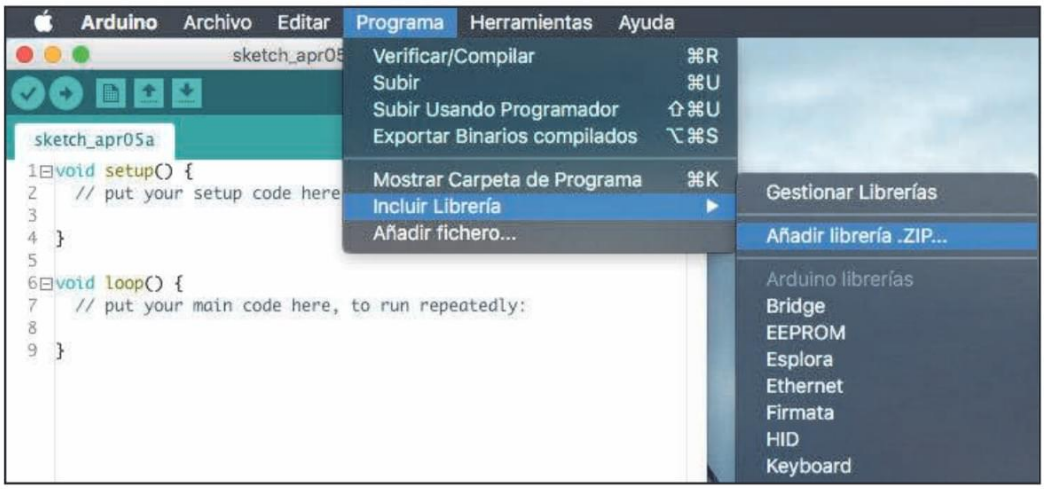
03

Una vez que la descarga de la librería se haya completado, verifique el contenido de la carpeta. En este caso, se ven todos los archivos que componen la librería NewPing.



04

Inicie el IDE de Arduino y despliegue las opciones que se encuentran en el menú `Programa`, seleccione `Incluir librería` y luego pulse clic sobre `Añadir librería .ZIP`. Antes de realizar este paso, tenga en cuenta el lugar donde descargó la librería.



Gestor de librerías

Las últimas versiones del IDE de Arduino incorporan un práctico gestor de librerías, accesible desde el menú **Programa/Incluir librerías/Gestionar librerías**. Gracias a esta utilidad, es posible ver las librerías que se encuentran instaladas, buscar entre un listado de librerías disponibles, y también instalarlas y actualizarlas mediante unos pocos clics del mouse.

05

Utilice la ventana que se presenta para navegar hasta la ubicación de la librería comprimida, luego acepte. El proceso solo tardará algunos segundos; cuando se complete, verá un mensaje que indica que la librería fue añadida sin problemas.

06

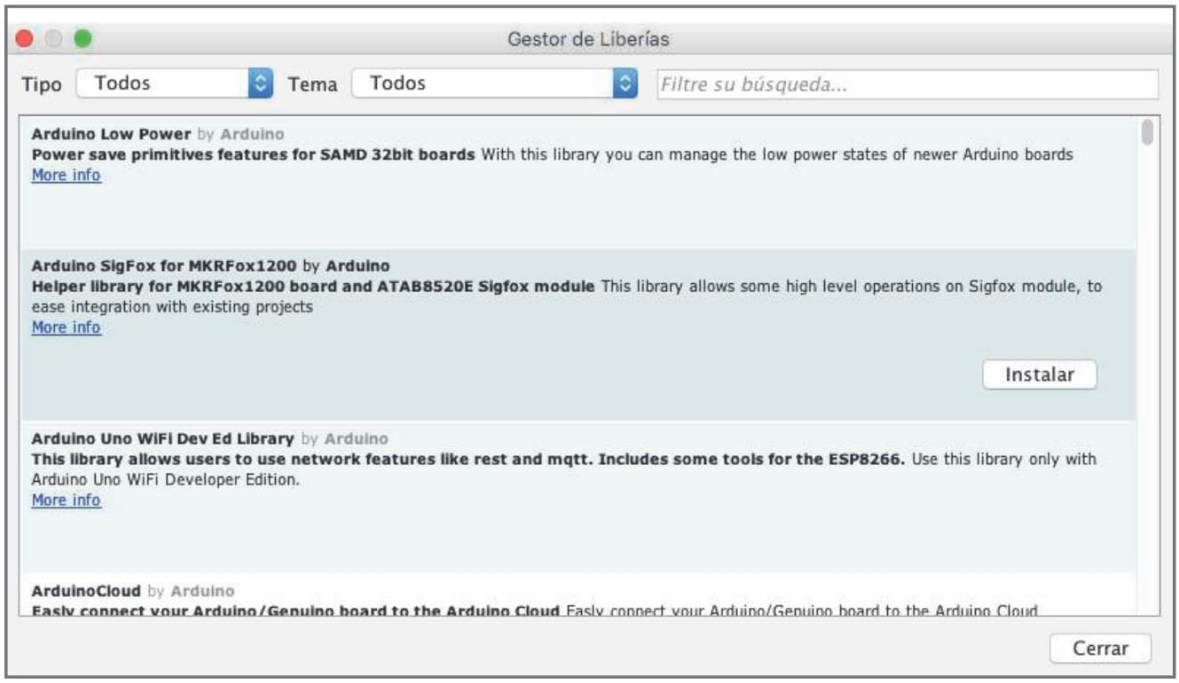
Para incluir la librería que se añadió recién, es necesario utilizar el código `#include`, seguido del nombre de la librería, por ejemplo, `#include<NewPing.h>`. También puede agregar esta línea de código en forma automática si despliega el menú `Programa`, haga clic en `Incluir librería` y, posteriormente, seleccione la librería deseada.

Si el método anterior no funciona, intentaremos agregar la librería en forma manual, para ello es necesario descargarla y luego descomprimir. Si estamos en un sistema Windows, copiamos los elementos descomprimidos en una carpeta que corresponda a la librería, la ruta será similar a la siguiente: **Mis Documentos\Arduino\libraries.**

En cambio, si nos encontramos en un sistema Mac OSX, tendremos que copiar la carpeta que contiene los elementos descomprimidos en la ruta **Documents/Arduino/libraries**. La ruta final debería quedar de la siguiente forma y con los siguientes archivos:

"Para instalar una librería en forma manual la descomprimimos y copiamos a la carpeta libraries."

Mis Documentos\Arduino\libraries\NewPing\NewPing.cpp
Mis Documentos\Arduino\libraries\NewPing\NewPing.h
Mis Documentos\Arduino\libraries\NewPing\examples



El Gestor de librerías es otra forma de instalar librerías en Arduino; se encuentra disponible en el menú **Programa/Incluir librería/Gestionar librerías**.

EJEMPLOS DE CÓDIGO

Uno de los puntos fuertes del IDE de Arduino es la incorporación de una gran cantidad de códigos o sketchs listos para utilizar. Una vez que iniciamos el IDE, podemos cargar cualquiera de estos programas en forma directa y probar su funcionamiento en la placa Arduino, luego es posible realizar algunas modificaciones para obtener variantes en las acciones que el programa realiza.

Con el IDE ejecutado haremos clic en **Archivo** y desplegaremos el submenú **Ejemplos**; de inmediato, veremos que se muestra una gran cantidad de ejemplos de código, para acceder a cada uno de ellos los seleccionamos y se presentarán en una nueva ventana del proyecto.

Los ejemplos se encuentran organizados en las siguientes categorías:

- **EJEMPLOS CONSTRUIDOS**: en este apartado encontraremos listados de ejemplos catalogados según su uso, por ejemplo, básicos, USB, comunicación, control, display, entre otros. En cada categoría se incluyen variados ejemplos de código.
- **EJEMPLOS PARA CUALQUIER TARJETA**: en esta categoría se reúne una serie de ejemplos que son válidos para cualquier tarjeta.
- **EJEMPLOS PARA ARDUINO/GENUINO UNO**: en esta sección se reúnen aquellos ejemplos adecuados para la tarjeta que hemos elegido para trabajar en el IDE. Podemos cambiar esta elección desplegando el menú **Herramientas/Placa**.
- **EJEMPLOS DE LIBRERÍAS PERSONALIZADAS**: finalmente, en este apartado encontramos los ejemplos que se incluyen en las librerías que hemos instalado utilizando los procedimientos que describimos en una sección anterior. En este caso deberíamos ver los ejemplos que incluye la librería NewPing.

Cargar un programa o sketch

La carga de un programa en la tarjeta Arduino es un proceso sencillo al que nos enfrentaremos con frecuencia, pues cada vez que deseamos probar alguna modificación en el código que escribamos será necesario cargarlo a la placa. Esta carga se puede realizar a través del mismo Arduino sin tener necesidad de utilizar otro tipo de elemento externo.

Para aprender a realizar esta tarea y también verificar la forma de uso de los ejemplos que incorpora el IDE de Arduino, cargaremos un ejemplo básico y lo ejecutaremos en la placa.

Como sabemos, al aprender un nuevo lenguaje de programación es típico encontrarnos con el clásico "Hola mundo", cuando se trata de la programación para Arduino también existe un programa básico, pero se denomina **Blink**.

Para realizar esta actividad, necesitamos contar con la placa Arduino, en este caso utilizaremos Arduino UNO, el protoboard, un led, una resistencia adecuada, algunos cables de puente y también el cable USB para conectar la placa a la PC donde tengamos instalado el IDE.

Lo primero que debemos hacer es conectar Arduino UNO a la PC mediante el cable USB, verificaremos que la luz de ON nos indique que la conexión es correcta.

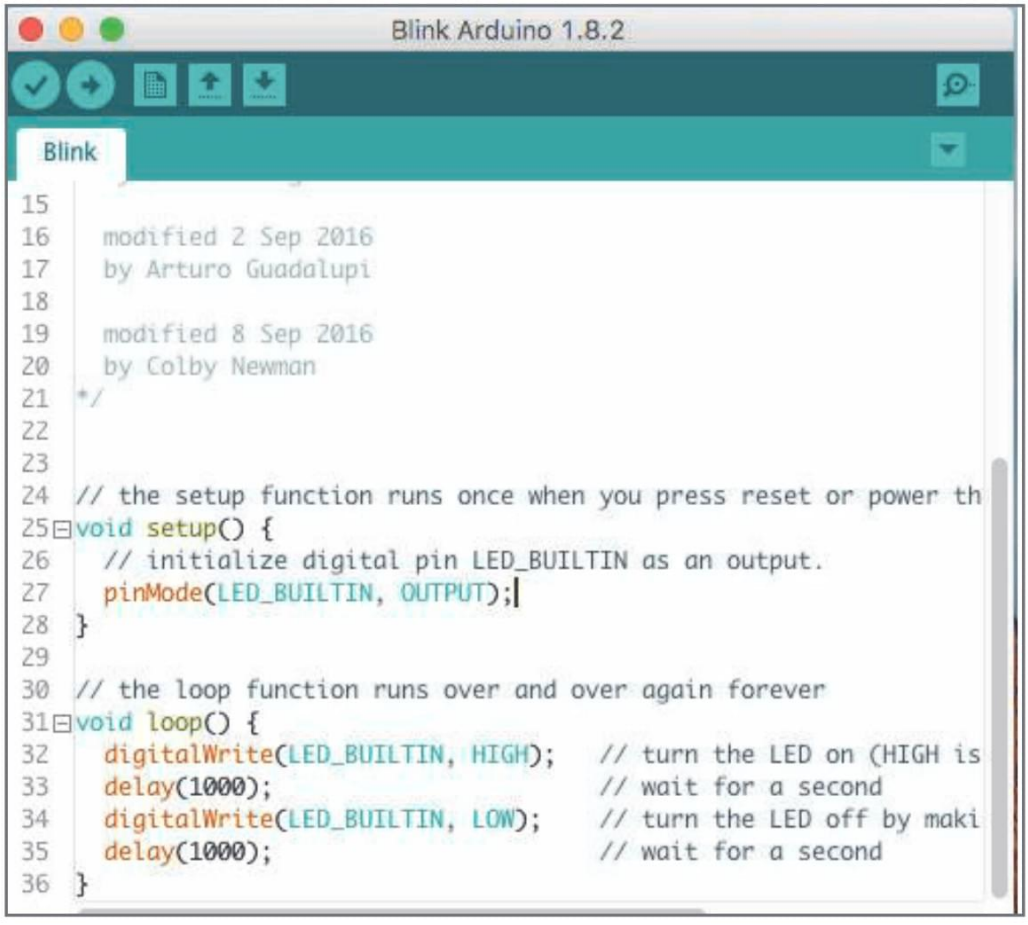
Luego utilizamos las opciones que se encuentran en el menú **Herramientas**, para seleccionar la placa y el puerto adecuado (la selección del puerto debe realizarse con la placa conectada a la PC).

Es necesario seleccionar el puerto USB para que la comunicación con la placa Arduino sea exitosa. Esto debe hacerse luego de conectar la placa a la computadora.

Con la placa y el puerto debidamente seleccionados, elegiremos el código de ejemplo para enviar a Arduino UNO. Esto lo realizamos en el menú **Archivo**, en el apartado **Ejemplos**, para este caso se trata de ejemplo **Blink**, que se encuentra en la categoría **Basics**. Una vez que lo seleccionamos, se abrirá una nueva ventana de proyecto que contiene el código adecuado. Gran parte del código que aparecerá en la ventana corresponde a comentarios de los desarrolladores; para ver el código que será ejecutado, debemos desplazarnos hacia abajo.

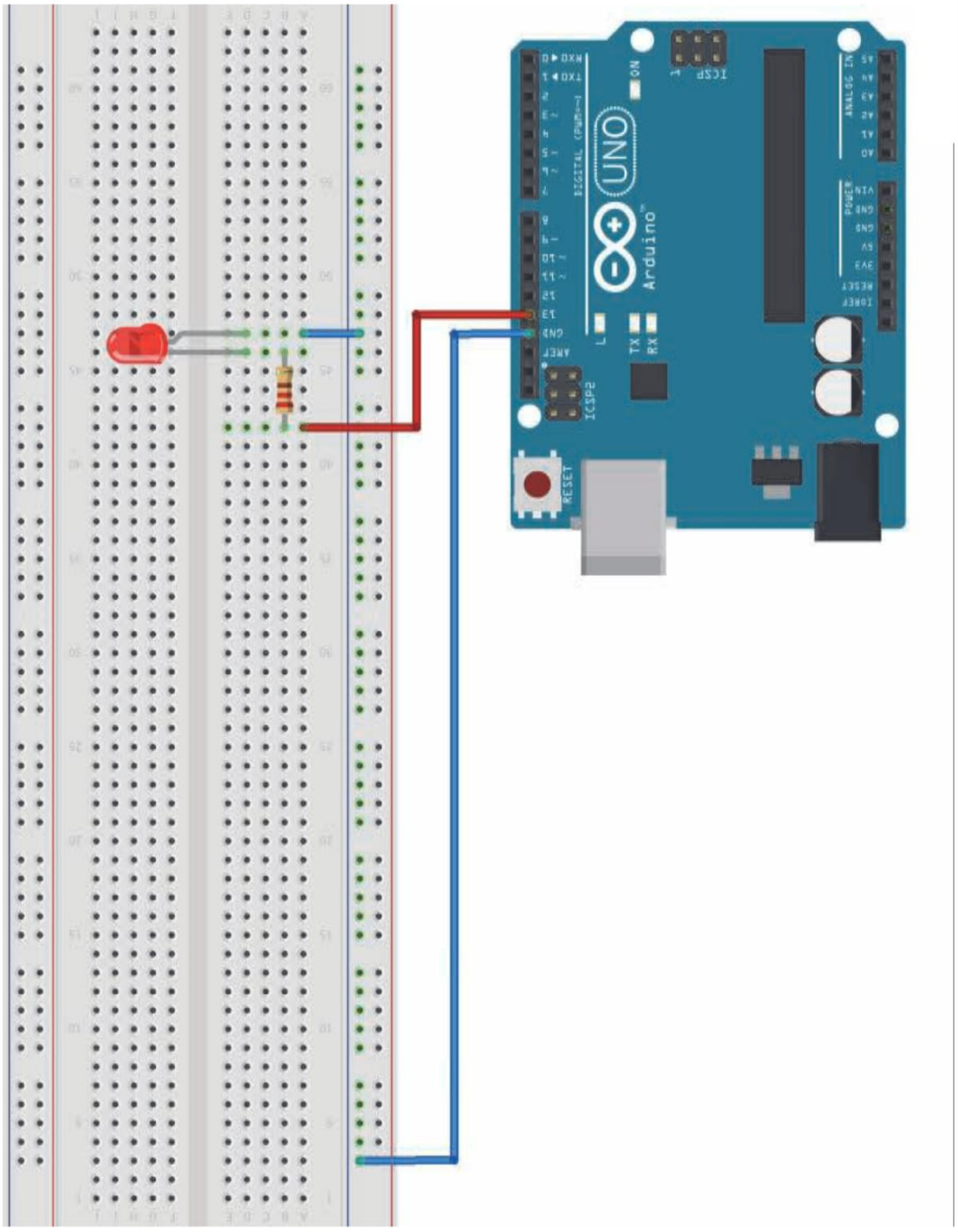
A continuación copiamos el código que corresponde al ejemplo Blink, sin los comentarios:

```
voidsetup() {
pinMode(LED_BUILTIN, OUTPUT);
}

voidloop() {
digitalWrite(LED_BUILTIN, HIGH);
delay(1000);
digitalWrite(LED_BUILTIN, LOW);
delay(1000);
}
```



En esta imagen vemos el código de Blink con los comentarios agregados por los desarrolladores.

+ Conectar para cargar el proyecto Blink



Aquí vemos el esquema de conexión que utilizaremos para cargar el proyecto Blink. Debemos tener cuidado de conectar cada elemento de la forma que se muestra.

"Para compilar y ejecutar el código que hemos escrito solo es necesario hacer clic en la opción Subir."

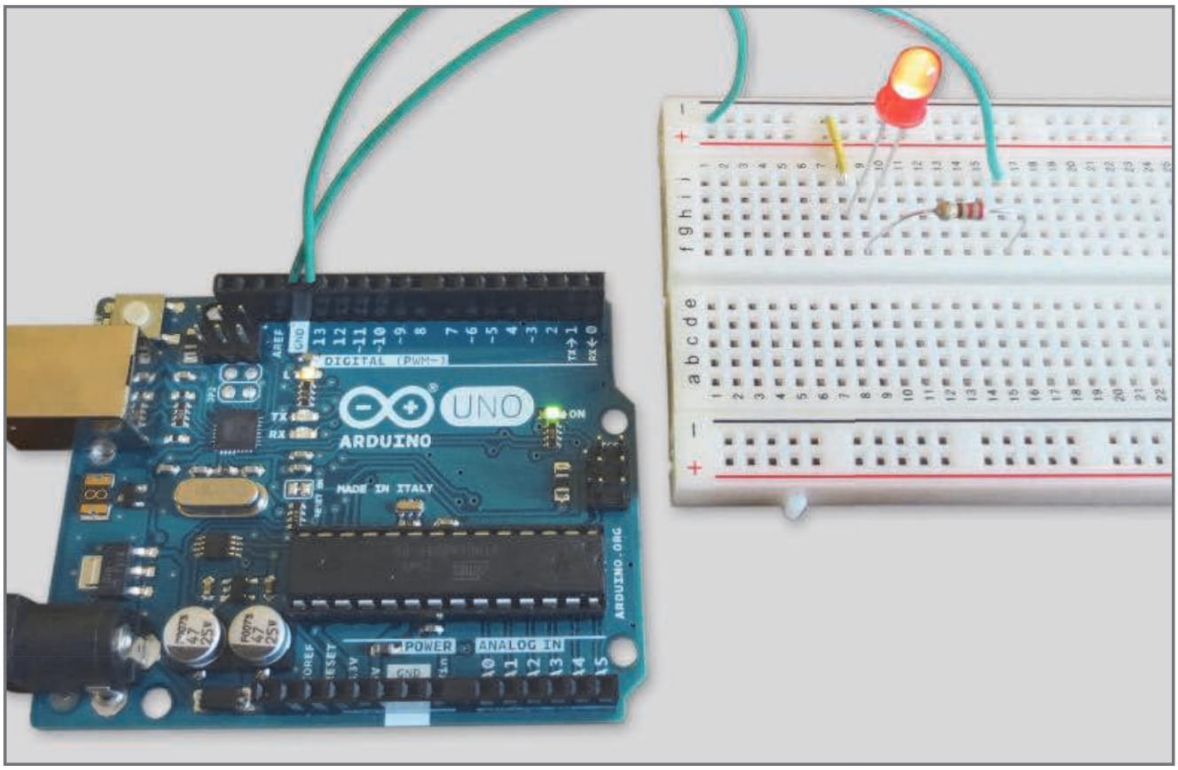
En este punto tenemos la placa conectada a la PC y también el código que deseamos ejecutar, por lo tanto, debemos enviarlo a Arduino UNO. Antes de eso, prepararemos las conexiones que necesitamos en el protoboard, el esquema de conexión requerido se muestra en el cuadro que se encuentra en la página anterior.

Con las conexiones realizadas, hacemos clic en **Programa/Subir**, de inmediato el código será compilado y, si no se encuentra ningún error, se grabará en la placa; en unos segundos será ejecutado, y veremos el led parpadeando. Cuando el programa haya sido cargado, observaremos un mensaje que indica que el procedimiento fue realizado.

Ahora que hemos cargado nuestro primer programa en Arduino UNO y logramos que se encienda el LED conectado, realizaremos algunas modificaciones en el código. Por ejemplo, si deseamos que el led se mantenga más tiempo encendido que apagado, cambiaremos los parámetros que se encuentran en **delay**, de la siguiente forma:

```
voidsetup() {
pinMode(LED_BUILTIN, OUTPUT);
}

voidloop() {
digitalWrite(LED_BUILTIN, HIGH);
delay(9000);
digitalWrite(LED_BUILTIN, LOW);
delay(1000);
}
```

Luego de esto, repetimos el procedimiento para subir el programa a la placa y esperamos que se ejecute el código. La modificación que efectuamos ordenará al led mantenerse encendido por 9 segundos y apagado por 1 segundo.



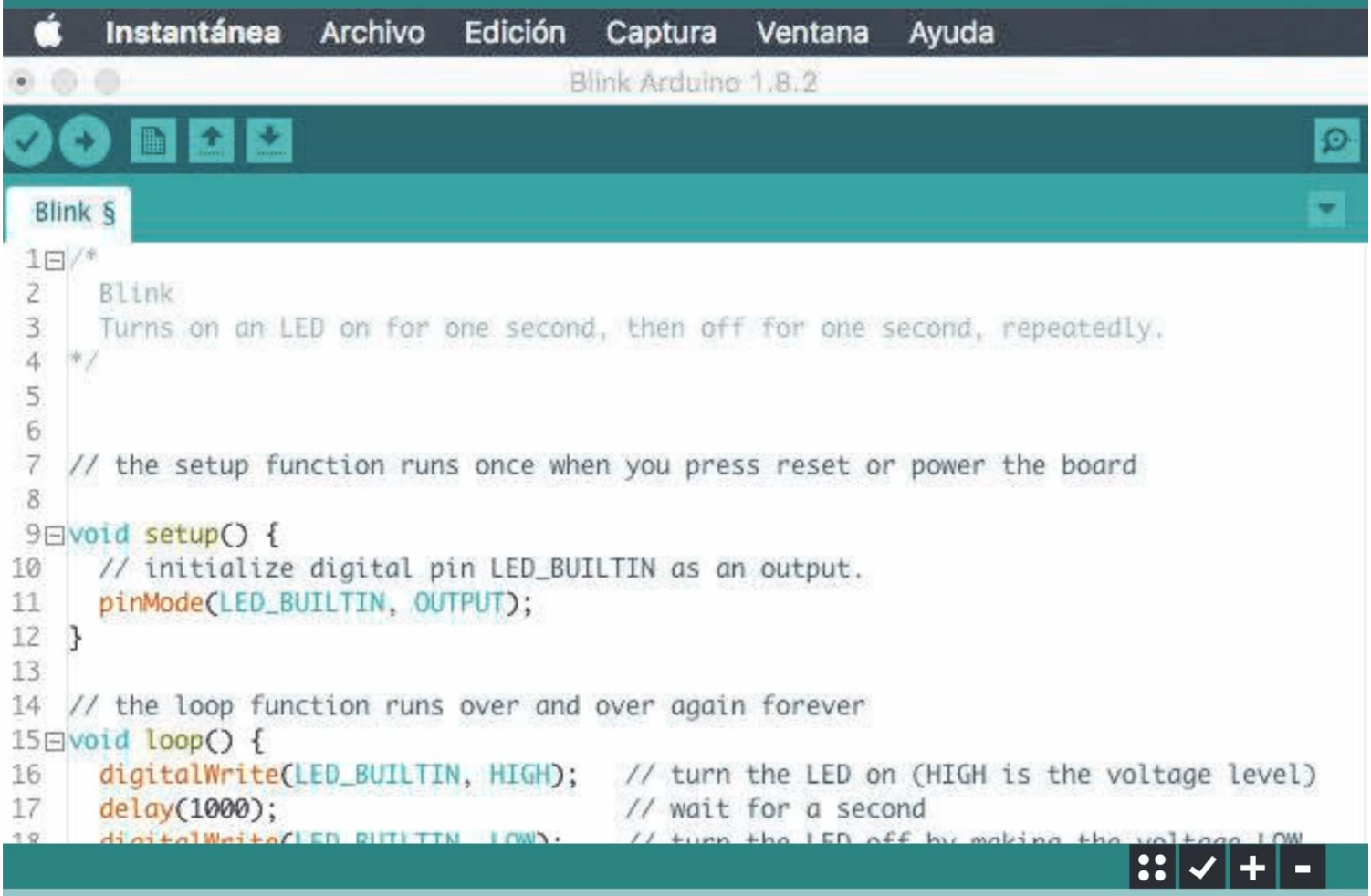
El programa que cargamos en la tarjeta se mantendrá almacenado, aunque desconectemos y volvamos a conectar el cable USB a la PC. Lo podemos eliminar presionando el botón **Reset**, ubicado en la esquina de la placa, o subiendo un nuevo sketch.

Resumen Capítulo 04

En este capítulo conocimos las características del IDE de Arduino, analizamos su funcionamiento y aprendimos a configurarlo para obtener la mayor información posible mientras trabajamos en la programación de nuestros sketchs. Más adelante revisamos qué son y para qué sirven las librerías de Arduino, conocimos los tipos de librerías existentes, y aprendimos a instalar y agregar una nueva librería a un proyecto. Para terminar, revisamos los ejemplos de código que incluye el IDE de Arduino y vimos la forma en que podemos grabar un sketch en la placa y, de esa forma, ejecutarlo.

Programar Arduino

05

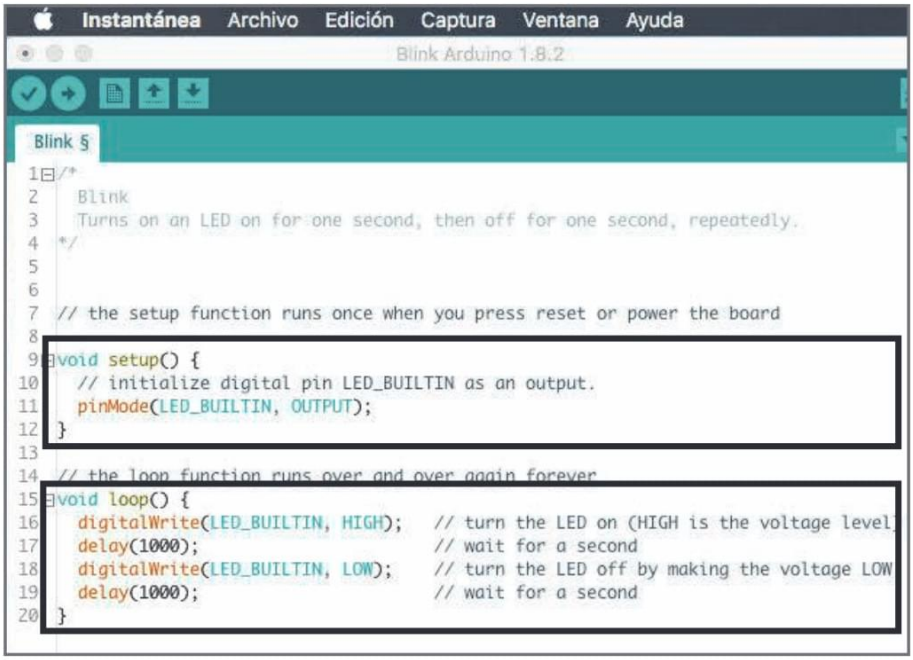


Ya hemos conocido el IDE de Arduino, revisamos la forma de instalarlo y también cómo podemos configurar su interfaz para obtener la mayor información posible. En este capítulo, conoceremos la sintaxis adecuada para crear nuestros sketchs.

ESTRUCTURA BÁSICA DE UN SKETCH

Ya sabemos que un programa para Arduino se denomina **sketch** y posee la extensión .ino. Un proyecto puede componerse por varios archivos, pero todos deben estar alojados en la misma carpeta donde está el archivo principal.

La estructura básica de un sketch de Arduino es bastante sencilla y, generalmente, encontraremos dos partes obligatorias, las que se encargarán de encerrar las declaraciones, estamentos y también instrucciones: **setup()** y **loop()**.



En esta imagen se indican las partes principales de un sketch de Arduino, la sección `setup()` y también la sección `loop()`.

Ejecución de instrucciones

Debemos tener en cuenta que las instrucciones que escribimos en la sección **voidsetup()** se ejecutarán una vez, cuando se encienda la placa Arduino. Por otra parte, las instrucciones que escribamos en la sección **voidloop()** se ejecutarán después de las instrucciones contenidas en **voidsetup()**, pero lo harán infinitas veces, hasta que la placa se apague o se resetee.

Por un lado, en **setup()** se recoge la configuración adecuada, por otro lado, en **loop()** se encuentran las instrucciones que se ejecutan en forma cíclica.

Además de estas dos secciones básicas, podemos encontrar otros apartados en un sketch:

COMENTARIOS

En un sketch podemos encontrar líneas de comentarios, se trata de indicaciones o información sobre las tareas relacionadas con una línea de código o con una parte del programa. Si se trata de un comentario de varias líneas, deberá delimitarse por /* al comienzo y por */ al final. En cambio, si es un comentario de una sola línea, debemos comenzarlo con //. Por lo general, encontraremos un segmento de comentarios al inicio del sketch o también al inicio de cada bloque de código.

VARIABLES

Las variables son adecuadas para guardar información, pueden estar referidas a diferentes ámbitos y relacionarse con distintos tipos de datos. Al inicio de un sketch de Arduino, se declararán las variables globales, las veremos en detalle más adelante en este mismo capítulo.

FUNCIONES

Si tenemos bloques de código que utilizaremos a menudo, es posible crear una función para facilitar nuestro trabajo. En este sentido, una función es un bloque de código que encontramos fuera de setup() y loop(), que posee un nombre característico, con uno o más parámetros de entrada, y puede devolver o no un valor. Por ejemplo:

```
tipo nombreFuncion(parametro1, parametro2, ...)
{
  sentencia1;
  sentencia2;
  ...
}
```

Aunque la función no necesite parámetros de entrada, se deben incluir los paréntesis de apertura y de cierre.

Case sensitive

Al programar en Arduino, debemos tener en cuenta que es *case-sensitive*, es decir, que es diferente escribir una letra en mayúscula que en minúscula, por ejemplo **arduino** y **Arduino** serían tratados en forma distinta, o las funciones **detalle** y **DETALLE** también son consideradas diferentes.

No se trata de algo trivial, por ejemplo, si escribimos **Serial.begin(7700);** el compilador no nos mostrará errores, pero, si escribimos **serial.begin(7700);** el código no se compilará pues detectará un error en la sintaxis.

Tabulaciones

El uso de tabulaciones no es necesario para que la compilación del sketch se realice en forma correcta, aunque se trata de una manera de escribir código ordenado, claro y cómodo tanto para el programador como para quien debe leer o analizar el código en forma posterior. De esta manera, las tabulaciones nos facilitan la tarea de leer código escrito y nos permite mantener una estructura a la hora de escribirlo.

Teniendo esto en cuenta, el siguiente código tabulado:

```
voidsetup() {
pinMode(LED_BUILTIN, OUTPUT);
}

voidloop() {
digitalWrite(LED_BUILTIN, HIGH);
delay(1000);
digitalWrite(LED_BUILTIN, LOW);
delay(1000);
}
```

tiene el mismo efecto que el siguiente código sin tabulaciones:

```
voidsetup() {
pinMode(LED_BUILTIN, OUTPUT);
```

```
}
voidloop() {
digitalWrite(LED_BUILTIN, HIGH);
delay(1000);
digitalWrite(LED_BUILTIN, LOW);
delay(1000);
}
```

Puntos y comas

En un sketch de Arduino, las instrucciones deben terminar con un punto y coma, si no agregamos este signo al final de cada línea nos encontraremos con problemas a la hora de realizar la compilación. Esto sucede porque el compilador busca este signo para detectar el final de cada instrucción.

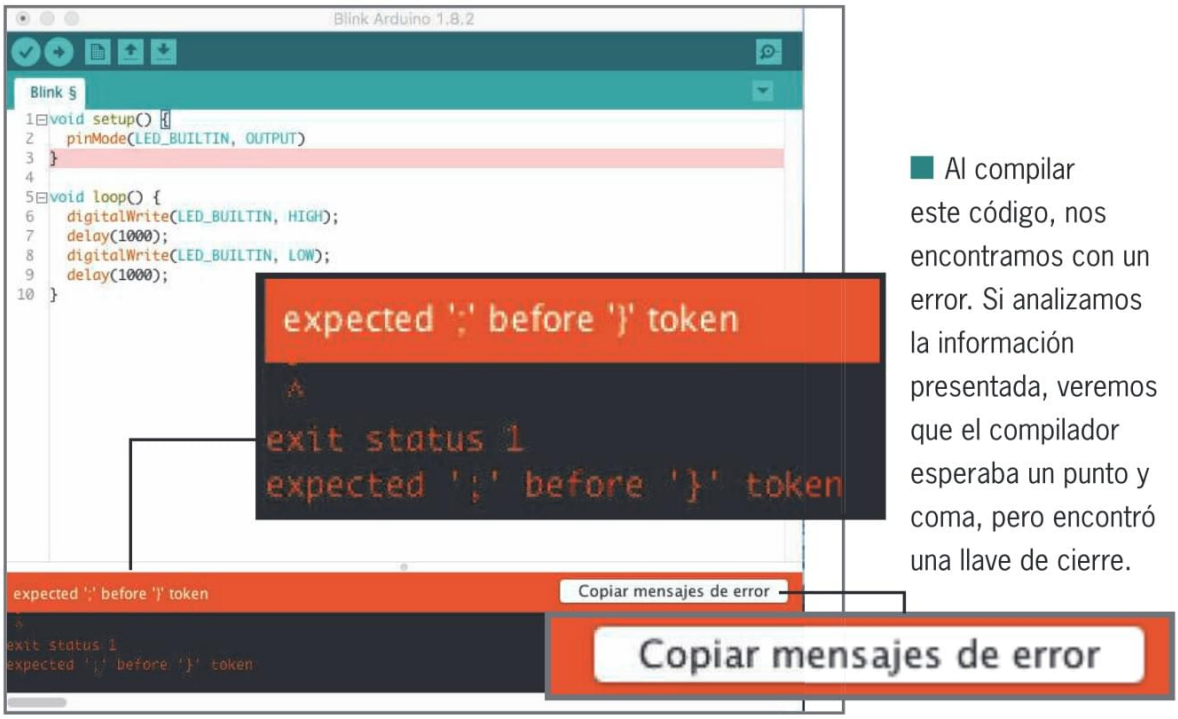
Así, la instrucción no presentará problemas:

```
voidloop() {
digitalWrite(LED_BUILTIN, HIGH);
delay(1000);
digitalWrite(LED_BUILTIN, LOW);
delay(1000);
}
```

Pero, por otra parte, encontraremos errores si intentamos compilar lo siguiente:

```
voidloop() {
digitalWrite(LED_BUILTIN, HIGH)
delay(1000)
digitalWrite(LED_BUILTIN, LOW)
delay(1000)
}
```

"Para marcar el final de cada instrucción debemos utilizar un punto y coma (;) "



Al compilar este código, nos encontramos con un error. Si analizamos la información presentada, veremos que el compilador esperaba un punto y coma, pero encontró una llave de cierre.

FUNCIONES

Cuando desarrollamos código, una **función** es un bloque de instrucciones que se encuentran identificadas por un nombre y que pueden ejecutarse cada vez que lo necesitemos (cuando sea llamada).

Para declarar una función, es necesario saber el tipo de datos que esta devolverá, por ejemplo un valor entero; luego, debemos especificar el nombre de la función y, posteriormente, las instrucciones que ejecutará, como en el siguiente código:

Necesidad de las funciones

Si nos apegamos a la estructura de un sketch, podríamos decir que las funciones no son necesarias pues se trata de un conjunto de código que puede ser insertado en el lugar del sketch donde haga falta. Pero, entonces, ¿cuál es la utilidad de las funciones? En verdad se trata de una forma de generar código limpio y ordenado, además de reutilizar algunos segmentos sin necesidad de reescribirlos todas las veces que los necesitemos.

```
intvalor() {
int val;
   val = analogRead(pot);
   val /= 4;
return val;
}
```

Siguiendo el ejemplo que aprendemos, cada vez que nos enfrentamos a un nuevo lenguaje de programación podríamos escribir una función que muestre por el puerto serie **Hola Mundo**; para ello escribimos el siguiente código:

```
voidHolaMundo(){
   Serial.println("Hola Mundo");
}
```

En esta función encontramos lo siguiente:

- **void**: la utilizamos cuando la función escrita no produce ningún resultado, en este caso solo imprime un mensaje.
- **HolaMundo**: se trata del nombre de la función que estamos escribiendo.
- **( )**: entre los paréntesis ubicamos los parámetros; para este caso no existen parámetros, por esa razón los paréntesis se escriben vacíos.

Un programa completo, utilizando la función que acabamos de crear, es el siguiente:

```
voidHolaMundo(){
   Serial.println("Hola Mundo");
}

voidsetup()
{
   Serial.begin(9600);
}
```

```
voidloop()
{
   HolaMundo();
   delay(4000);

}
```

Parámetros

Hasta este momento conocimos qué son y para qué nos sirven las funciones, pero ejemplificamos su uso escribiendo una función bastante sencilla que no incluye el uso de parámetros.

Los **parámetros** pasan información a la función cuando necesitamos que esta nos devuelva un resultado. Por ejemplo, imaginemos que necesitamos una función que nos entregue el resultado de una operación matemática, como la función no sabe el valor de los números que participarán en la operación, debemos pasárselos como parámetros. Veamos un ejemplo:

```
void restar(int c, int d){
   int resta = c + d;
   Serial.println(String(resta));
}
```

En este código vemos los parámetros **c** y **d**, ambos enteros, que son pasados entre los paréntesis: **(int c, int d)**, y se encuentran justo después del nombre de la función.

Manipulación de tiempo

Entre las funciones de manipulación de tiempo que podemos utilizar en la programación de Arduino, tenemos las siguientes: **millis función()**, que se utiliza para devolver el número de milisegundos en el tiempo; función de **micros()**, que devuelve el número de microsegundos del tiempo.

VARIABLES

En palabras sencillas, una variable nos sirve para almacenar datos, que pueden ser consultados y utilizados desde diversos segmentos del sketch. Para utilizar un dato específico, accedemos a él mediante el nombre de la variable (estas pueden cambiar continuamente, lo cual, no pasa lo mismo con las constantes cuyo valor nunca cambia).

Existen diferentes tipos de variables, dependiendo de los datos que almacenamos; entre ellos, encontramos los siguientes:

- **char**: almacena caracteres, ocupa un byte.
- **byte**: almacena un número entre 0 y 255.
- **int**: almacena un número entre -32,768 y 32,767, ocupa 2 bytes.
- **unsignedint**: almacena un número entre 0 y 65,535, ocupa 2 bytes.
- **long**: ocupa 4 bytes, almacena un número entre -2,147,483,648 y 2,147,483,647.
- **float**: almacena números decimales que van entre -3.4028235E+38 y +3.4028235E+38.
- **double**: almacena números decimales, pero dispone de 8 bytes.
- **string**: almacena cadenas de texto.

Al declarar una variable, es importante elegir la que menos espacio utilice pues hay que considerar que se restará ese espacio de la memoria de nuestra placa. Para declarar una variable, debemos indicar un nombre y el tipo de variable; además, es posible asignarle un valor en forma inmediata, por ejemplo:

```
charCaracterDos='b';
byte Numero = 192;
intMiEntero;
unsignedintnumPos = 2212;
```

Las variables deben tomar nombres mas descriptivos para que podamos hacer el código mas legible. Como vemos, hemos creado cuatro variables, aunque solo en tres de ellas asignamos un valor, dejamos la variable de tipo **int** sin inicializar. Veamos un ejemplo de código donde se utilizan variables de tipo **int**:

```
int f = 7;
int g = 9;
int h = f + g;

voidsetup()
{
   Serial.begin(9600);
   Serial.print("f + g equals ");
   Serial.println(String(h));
}

voidloop()
{
}
```

En este código hemos creado las variables **f**, **g** y **h**, en las dos primeras almacenamos los datos **7** y **9**, respectivamente, mientras que en la variable **h** almacenamos el resultado de sumar las variables **f** y **g**.

Ámbitos

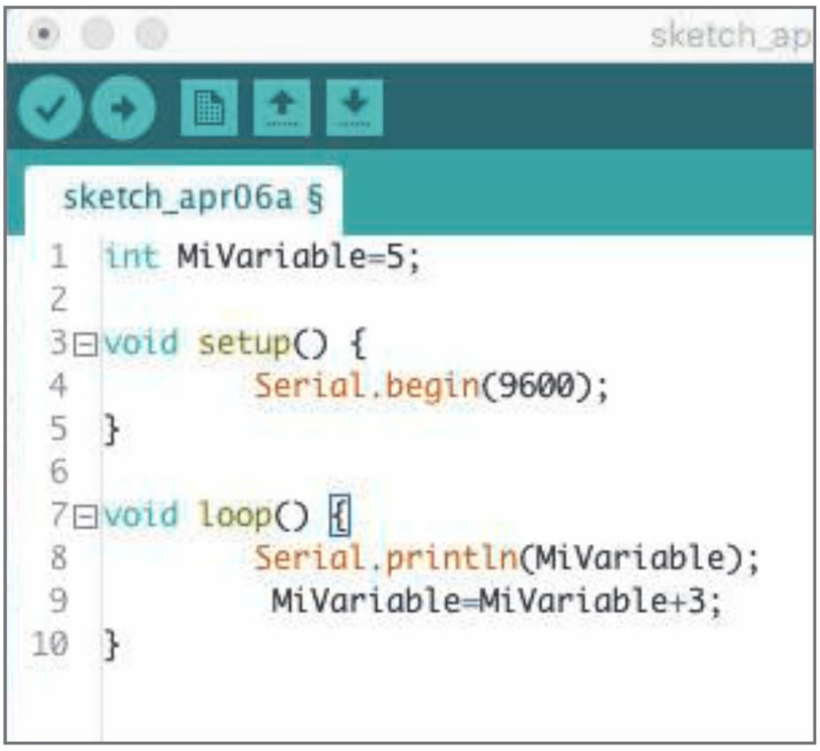
Las variables pueden ser globales o locales, la diferencia entre ellas radica en el alcance que poseen, es decir, desde qué lugares del sketch pueden ser utilizadas. Una variable es global cuando podemos acceder a ella desde cualquier lugar del sketch, es decir, está disponible tanto para **setup()** como para **loop()**. Veamos un ejemplo:

Constantes

Podemos entender a las constantes como variables de solo lectura, es decir, no es posible modificar el valor que les fue asignado; si intentamos hacerlo, el compilador nos mostrará un error. La definición de una constante se hace de forma similar a una variable, pero debemos preceder su declaración con la palabra **const**. Por ejemplo, **constint numero1=200**. En este caso, se trata de una constante de nombre **numero1**, con el valor **200**.

```
int a= 2;

voidsetup()
{
   Serial.begin(9600);
   Serial.println(String(a));

}

voidloop()
{
   a = a + 2;
   Serial.println(String(a));
   delay(3000);
}
```

En este caso hemos declarado la variable fuera de **setup()** y **loop()**, por lo tanto, su ámbito es global y la podemos utilizar en cualquier parte del sketch.



En este sencillo código hemos creado una variable global, por lo tanto, la podemos utilizar desde **setup()** y también desde **loop()**. En este ejemplo la hemos nombrado **MiVariable** y le asignamos el valor **5**.

Por otra parte, una variable es local cuando se declara dentro de un ámbito específico (dentro de corchetes), de esta forma no podrá ser usada en un ámbito distinto. Veamos un ejemplo basado en el código anterior:

```
voidsetup()
{
int a= 2;
   Serial.begin(9600);
   Serial.println(String(a));

}

voidloop()
{
   a = a + 2;
   Serial.println(String(a));
   delay(3000);
}
```

En este caso, hemos declarado la función dentro de **setup()**, por lo tanto, podemos utilizarla solo dentro de ese ámbito. Al llamarla desde **loop()** obtendremos un error de compilación. Una posible solución sería la siguiente, aunque no es un código correcto pues debemos tener en cuenta que la variable **int** dentro de **loop()** se creará y se destruirá con cada iteración, pero no presentará errores de compilación:

Consejos para declarar variables

Como vimos, la declaración de variables no es una tarea compleja, pero, de igual forma, es necesario tener en cuenta algunas recomendaciones importantes. En primer lugar, necesitamos examinar muy bien el ámbito que deseamos asignarle a la nueva variable; dependiendo de esto, la escribiremos en un lugar o en otro. Por otra parte, también es relevante darle un nombre descriptivo, de esa forma será más fácil reconocerla en códigos extensos, por ejemplo, es mejor una variable denominada **SensorDeDistancia** que una llamada **VariableDos**.

```
voidsetup()
{
int a= 2;
   Serial.begin(9600);
   Serial.println(String(a));

}

voidloop()
{
int a= 2;
   a = a + 2;
   Serial.println(String(a));
   delay(3000);
}
```

DATOS Y OPERADORES

Dos conceptos relevantes que debemos tener en cuenta a la hora de programar para Arduino son los **datos** y los **operadores**. Los datos son importantes pues se trata de la materia prima con la que trabajaremos y, por otra parte, los operadores nos permitirán generar ciertas acciones sobre los datos, de esta forma obtendremos resultados.

En primer lugar, analizaremos los diferentes tipos de datos con los que podemos trabajar, esto se encuentra íntimamente relacionado con las variables que conocimos en una sección anterior, pues, para declarar una variable, definimos el tipo de dato que almacenará. Los tipos de datos más comunes utilizados en un ambiente Arduino se muestran en la tabla que mostramos a continuación.

"Tal como sucede con cualquier lenguaje de programación necesitamos diferentes tipos de datos y operadores para realizar acciones sobre ellos."

DATOS PARA PROGRAMAR EN ARDUINO IDE

TIPO DE DATO	TAMAÑO	DESCRIPCIÓN
boolean	**8 bits**	Lógico simple, verdadero/falso.
byte	**8 bits**	Número sin signo entre 0 y 255.
char	**8 bits**	Número con signo entre -128 y 127.
unsignedchar	**8 bits**	Número sin signo entre 0 y 255.
word	**16 bits**	Número sin signo entre 0 y 65535.
int	**16 bits**	Número con signo entre -32768 y 32767. Se trata del tipo de dato más usado para propósitos generales en Arduino.
unsignedlong	**32 bits**	Número sin signo entre 0 y 4294967295. Se utiliza para guardar el resultado de la función millis().
long	**32 bits**	Número con signo entre -2,147,483,648 y 2,147,483,647.
float	**32 bits**	Número con signo entre 3.4028235E38 y 3.4028235E38.

Tipos de datos comúnmente utilizados para programar en Arduino IDE.

El tamaño necesario para cada tipo de dato es un aspecto relevante a la hora de escribir código. Debemos recordar que, por ejemplo, el procesador ATmega328P es nativo de 8 bits, por lo que posee soporte integrado para números de punto flotante. Para trabajar con tamaños de memoria mayores a 8 bits, el procesador gestionará secuencias de código que le permitan segmentar el trabajo, para almacenar el resultado donde corresponda.

De esta forma, cuando estemos trabajando con un procesador nativo de 8 bits, lo mejor será elegir tipos de datos de 8 bits.

Ahora que revisamos los tipos de datos que tenemos a nuestra disposición, es tiempo de conocer los operadores que nos permitirán

trabajar con ellos. En la siguiente tabla se muestran, a grandes rasgos, los tipos de operadores disponibles. Más adelante analizaremos cada uno en detalle:

TIPOS DE OPERADORES

TIPO DE OPERADOR	EJEMPLOS DE OPERADORES	DESCRIPCIÓN
Aritméticos	**Asignación**	Se trata de la forma de entregar o asignar un valor.
	+, -, *, /	Operaciones aritméticas básicas: suma, resta, multiplicación y división.
	Módulo	Obtiene el resto de la división de un número por otro.
Compuestos	**++**	Los operadores compuestos permiten abreviar el código, por ejemplo, incrementar en uno el valor anterior.
	+= , -= , *= , /=	Estos operadores compuestos equivalen a x = x + y, x = x – y, x = x * y, y x = x / y, respectivamente.
Comparación	**==, !=, <, >, <=, >=**	Este tipo de operadores nos permiten comparar dos datos, por ejemplo, menor que, mayor que, diferente de, etcétera.
Booleanos	**AND (&&), OR (||) y NOT (!)**	También se conocen como operadores lógicos. Nos permiten conectar dos datos mediante el uso de nexos lógicos, comparando dos expresiones para devolver VERDADERO o FALSO.

Tipos de operadores ofrecidos por el core de Arduino, y algunos ejemplos específicos junto a su descripción.

Operadores aritméticos

Los operadores aritméticos son aquellos que más fácilmente asociamos con operaciones matemáticas; en este grupo se encuentran las cuatro operaciones fundamentales y también la asignación y el módulo.

- **+** operador suma (x=y+20)
- **–** operador resta (x=y-10)
- ***** operador multiplicación (x=y*2)
- **/** operador división (x=y/3)
- **=** operador de asignación (int numero1=25)

El operador **%** o módulo se encarga de devolver el resto cuando un número entero es dividido por otro. Por ejemplo en **x = 7 % 5** obtendremos como resultado **2**.

Las operaciones, como la suma o la división, tendrán en cuenta el tipo de dato que hemos asignado. Por ejemplo, si ejecutamos la operación 9/4, pero hemos indicado que los datos son **int**, obtendremos como resultado 2, no 2,25 pues el tipo de dato **int** no reconoce decimales. Si necesitamos obtener resultados con decimales, será necesario elegir el tipo de dato **float**.

Operadores compuestos

Una operación compuesta integra una operación aritmética junto a una variable asignada. Entre estos operadores, encontramos el incremento (**++**) y el decremento (**--**), que se utilizan para aumentar o disminuir el valor almacenado en una variable, respectivamente.
Podemos usarlos de la siguiente forma:

- **x++** para incrementar x una unidad y devolver el antiguo valor de x.
- **x--** para decrementar x una unidad y devolver el antiguo valor de x.
- **++x** para incrementar x una unidad y devolver el nuevo valor de x.
- **--x** para decrementar x una unidad y devolver el nuevo valor de x.

Otros operadores compuestos no son más que una simplificación de operaciones complejas y sirven para realizar una operación matemática en una variable con otra variable o constante. Por ejemplo, si escribimos **x += y**, equivale a la expresión **x = x + y**, o si escribimos **x -= y**, es igual a la expresión **x = x – y**.

Operadores de comparación

Utilizaremos los operadores de comparación en forma frecuente cuando trabajemos con estructuras condicionales, pues nos permiten averiguar si una condición es verdadera. A continuación, presentamos los operadores de comparación:

- **>** mayor que
- **<** menor que
- **>=** mayor o igual que
- **<=** menor o igual que
- **==** igual que
- **!=** diferente

El resultado de estos operadores puede ser **TRUE** o **FALSE**, su importancia radica en que, aplicados en estructuras condicionales, nos permitirán determinar el camino que debe seguir nuestro sketch, dependiendo de si la operación indicada resulta verdadera o falsa. Por ejemplo, podríamos verificar el nivel de tensión en un pin de la tarjeta Arduino y, según la lectura, activar o apagar un led.

Algunos ejemplos de operaciones de comparación son los siguientes:

```
60 < 120;
30 > 28;
45<= 32;
20 == 21;
```

Las dos primeras comparaciones entregarán como resultado **TRUE**, mientras que las demás serán **FALSE**.

Igualdad y asignación

Aunque parezcan similares, los operadores **==** y **=** son diferentes. El primero, el signo igual escrito dos veces (**==**), corresponde a la operación de igualdad que nos permite comparar dos valores, devolviendo **TRUE** si son iguales y **FALSE** si son distintos. Por otra parte, el signo igual escrito una sola vez (**=**) se refiere a la operación que nos permite asignar un valor a una variable o constante.

Operadores lógicos

Cuando necesitamos realizar un análisis sobre más de alguna condición para que se produzca un resultado, es necesario utilizar los operadores lógicos. Los operadores lógicos básicos son los siguientes:

- **AND (&&)**: permite validar dos o más condiciones, estas deben cumplirse todas para que se ejecute el código dentro de las llaves. Por ejemplo:

```
if (variable1 > variable1 && variable3 > variable4) {
digitalWrite(pinLedRojo, HIGH);
}
```

  Si el valor de la variable1 es mayor que el almacenado en la **variable2**, y el valor de la **variable3** es mayor que el de la **variable4**, se ejecutarán las instrucciones entre corchetes.

- **OR(||)**: permite que, al cumplirse cualquiera de las condiciones, se ejecute el código entre las llaves. Por ejemplo:

```
if (variable1 > variable1 || variable3 > variable4) {
digitalWrite(pinLedVerde, HIGH);
}
```

  Si el valor de la **variable1** es mayor que el almacenado en la **variable2**, o el valor de la **variable3** es mayor que el de la **variable4**, se ejecutarán las instrucciones entre corchetes.

- **NOT (!)**: este operador ejecuta las sentencias que están entre llaves cuando la condición es evaluada como falsa.

ESTRUCTURAS DE CONTROL

Las estructuras de control son elementos que nos permiten tomar decisiones basándonos en los datos que se encuentran disponibles en determinado momento. Para ello, elegiremos como base los test lógicos que se realizan gracias a los operadores que ya hemos conocido.

if

Se trata de una de las estructuras de control más básicas que podemos utilizar mientras programamos para Arduino. Su sintaxis es la siguiente:

```
  if (test) {
     // acciones a realizar si el test devuelve ver-
dadero o true
  }
```

Por ejemplo, podríamos desarrollar un sketch que se encargara de comparar los valores almacenados en dos variables (valor, intensidad de ruido, etcétera); si el valor que corresponde a la **variable1** es inferior al que se encuentra en la **variable2**, se ejecutará el código contenido entre las llaves del bloque **if**, de lo contrario se ejecutará la instrucción que continúa luego del bloque **if**.

if else

Si necesitamos especificar las instrucciones que deben ser ejecutadas cuando el resultado del test resulte negativo, podemos complementar la estructura de control con **else**:

```
  if (test) {
  // acciones a realizar si el test devuelve verdade-
ro o true
  }
  else {
  // acciones a realizar si el test devuelve false
  }
```

if else elseif

Pero eso no es todo, cuando trabajamos con este tipo de estructuras de control, es posible detallar aún más las instrucciones que se ejecutarán, para ello integraremos **elseif**. La función de esta modificación es permitirnos detallar el control que ocurre en el bloque **if**. Gracias a **elseif**, introduciremos un test lógico adicional dentro de **if**, teniendo la oportunidad de ejecutar un bloque de acciones alternativo, por ejemplo:

```
if (variable1 < variable2) {
digitalWrite(PinLedRojo, HIGH);
}
elseif (variable1 == variable2) {
digitalWrite(PinLedVerde, HIGH);
}
else {
digitalWrite(PinLedRojo,LOW);
}
```

En este ejemplo, encenderemos el LED rojo en caso de que la **variable1** sea menor a la **variable2**, pero encenderemos el LED verde si ambas variables son iguales. Si ninguno de estos test resulta verdadero, encenderemos el LED rojo.

switch case

Es posible programar diferentes bloques **if** utilizando **elseif**, cuando necesitamos que el sketch decida entre varias opciones. Pero en realidad no lograremos un código fácil de leer, para estos casos es recomendable el uso de la estructura **switch case**. Esta estructura nos permite elegir entre varias opciones, su sintaxis básica es la siguiente:

```
switch (variable){
   case valor1:
   // instrucciones para variable == valor1
   break;
case valor2:
   // instrucciones para variable == valor2
   break;
case valor3:
   // instrucciones para variable == valor1
   break;
.....

default:
   // instrucciones para otros casos
   break;
}
```

La instrucción **break** es opcional, pero al agregarla no se seguirán analizando los demás casos hasta el final si uno resulta verdadero. Por otra parte, la instrucción **default** también es opcional, pero su importancia radica en que nos permite indicar aquellas acciones que serán ejecutadas cuando ninguno de los casos anteriores resulte verdadero. A continuación, vemos un ejemplo más concreto del uso de esta estructura de control:

```
int v;
voidsetup(){
Serial.begin(9600);
}
voidloop() {
if (Serial.available()>0){
  x=Serial.parseInt();
switch (x) {
  case 20:
Serial.println("v es 20");
  break;
  case 50:
Serial.println("v es 50 ");
  break;
  default:
Serial.println("v no es 20 ni 50");
            }
  }
}
```

En este código que describimos podemos ver como declaramos la variable **v** sin un valor inicial, luego de esto, iniciamos la comunicación serial. Leemos un número entero en el buffer serial y lo guardamos en la variable **v**, posteriormente, utilizamos los casos de la estructura de control para imprimir un mensaje si el valor almacenado en **v** es **20**, otro mensaje si la variable almacena el valor **50** y un mensaje diferente si el valor no es 50 ni 20.

"Podemos declarar una variable sin un valor inicial y luego asignarle el valor adecuado"

BUCLES

Los **bucles** o **loops** son capaces de realizar una tarea en forma repetitiva hasta que se considere completa, estas estructuras son **for**, **while** y **do while**.

El bucle **for** será útil cuando se necesite ejecutar un bloque de código una cantidad determinada de veces. Por lo general, integra un contador incremental que aumentará hasta que alcance un valor que prefijamos para que el bucle termine. La primera línea de este bucle es la instrucción **for**; esta posee tres partes: inicialización, test y el incremento o decremento de la variable de control; posee el siguiente aspecto:

```
for (int i = 0; i < 100; i++)
```

En él vemos que **i** se inicializa en **0**, luego, al final de cada bucle **i** se incrementará en **1** (ya sabemos que **i++** es una simplificación que corresponde a **i = i + 1**). Con esto aclarado, podemos observar este bucle en forma completa:

```
voidsetup()
{
Serial.begin(9600);
}
voidloop {
for (int i = 0; i < 100; i++){
Serial.println(i);
   }
}
```

Como vemos, las instrucciones que se encuentran en el interior del bucle **for** se ejecutarán hasta que se alcance el valor **100**. Cuando esto suceda, el bucle finalizará. Otro bucle que debemos conocer es **while**, se trata de una opción que se encarga de realizar un test sobre la expresión que indicamos y ejecuta el bloque de código que encerramos entre llaves, hasta que la expresión evaluada sea falsa. Su sintaxis es la siguiente:

```
while (expresion) {
   // sentencias para ejecutar
}
```

Teniendo esta sintaxis en cuenta, podemos ver la forma en que se realiza un bucle **while**, utilizando el ejemplo que entregamos para el bucle **for**:

```
voidsetup() {
Serial.begin(9600);
}

voidloop(){
int i = 0;
while (i < 100){
Serial.println(i);
          i++;
    }
}
```

El bucle **do while** se diferencia porque testea la expresión después de ejecutar el bucle, de esta forma, nos podemos asegurar de que las instrucciones contenidas son ejecutadas, al menos, una vez. Su sintaxis es la siguiente:

```
do {
    // bloque de código
   } while (expresión);
```

Al trabajar con códigos complejos, es necesario tener en cuenta algunas sentencias especiales, que podemos utilizar cuando trabajamos con varias funciones y deseamos que la ejecución del sketch siga algún camino en particular.

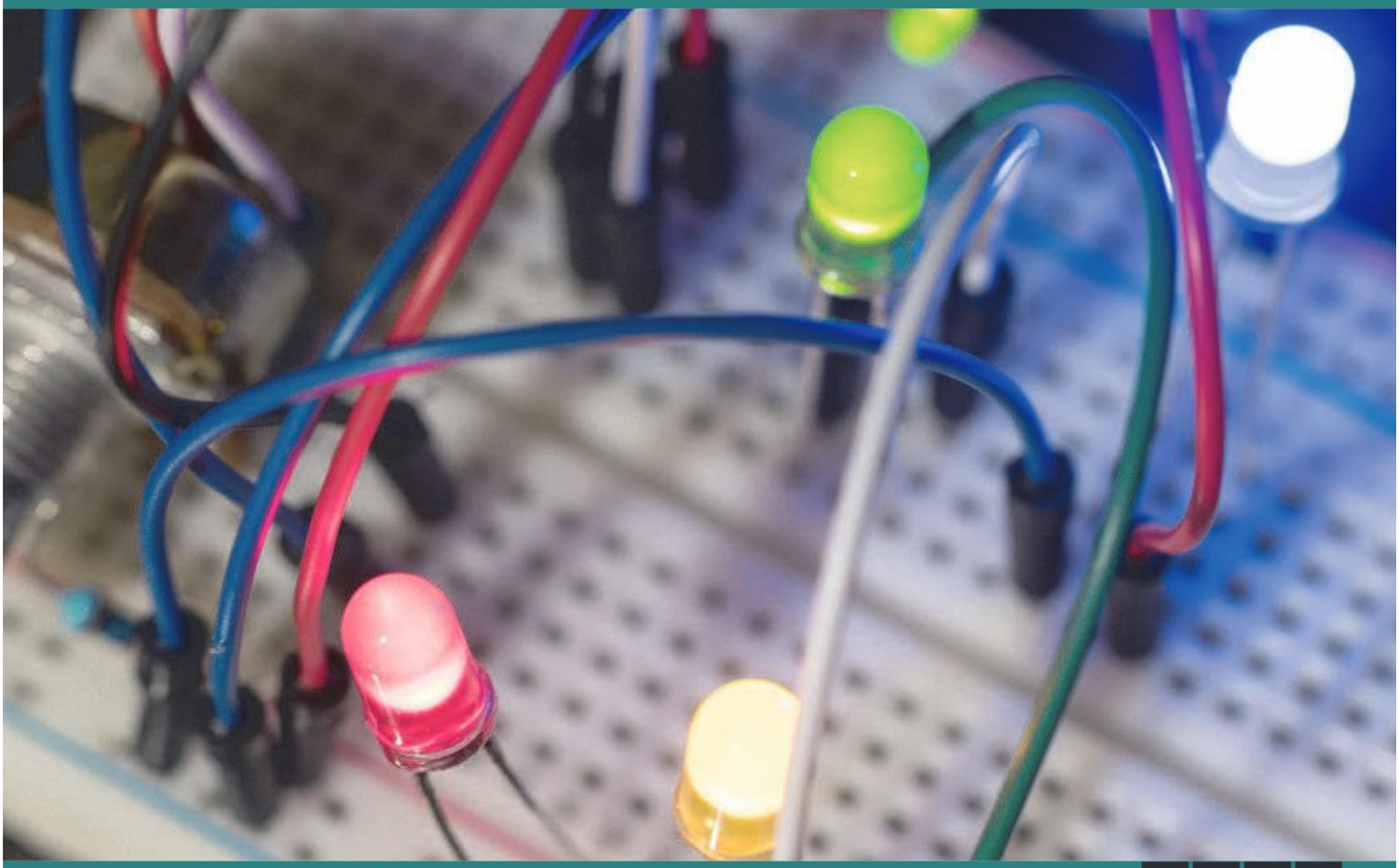
- **Break**: esta sentencia permite romper la iteración del bucle para salir de él sin que sea necesario que se cumplan las condiciones marcadas para salir.
- **Goto**: se trata de una sentencia que nos permite marcar el lugar del código al que realizaremos un salto. La posición desde la que saltamos se almacena en la pila del programa para que sea posible regresar.
- **Return**: es una sentencia que se utiliza para volver de un salto realizado con **goto**.

Resumen Capítulo 05

En este capítulo hemos realizado un recorrido básico por aquellos aspectos que es necesario conocer antes de programar nuestros sketchs en Arduino. Conocimos la estructura básica de un sketch, las funciones de las variables y también su uso. Analizamos los tipos de datos y los operadores con los que es posible trabajar, y revisamos algunos ejemplos de las estructuras de control que nos permitirán dotar, a los programas, de la capacidad de decidir entre varios caminos o de ejecutar una acción en repetidas ocasiones.

Trabajar con Leds

06



Es hora de poner manos a la obra, en este capítulo realizaremos nuestros primeros proyectos sencillos, para ello utilizamos nuestra placa Arduino junto a un conjunto de LEDs y algunos componentes adicionales.

LEDS Y ARDUINO

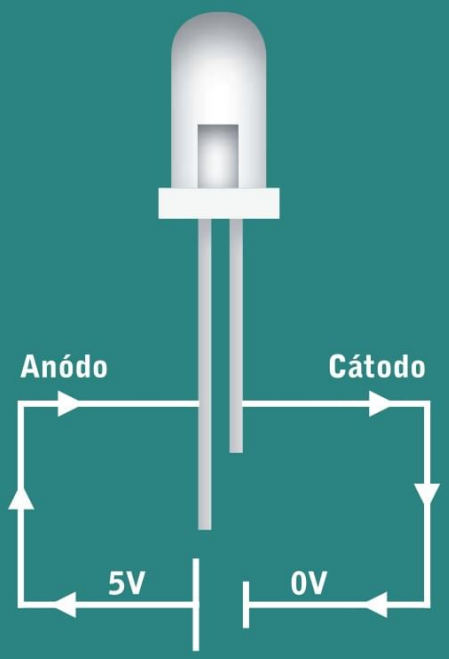
Ahora que hemos dado un paseo por algunas bases teóricas y también hemos revisado los procedimientos adecuados para comenzar a trabajar con Arduino, es hora de poner manos a la obra.

Existen muchos proyectos sencillos que es posible realizar, pero, en este capítulo, nos dedicaremos al manejo de los LEDs. Se trata de pequeños elementos que nos proporcionan un indicador claro de que están funcionando en forma correcta: emiten luz. Así podremos saber exactamente qué cambios logramos cuando realizamos una alteración o una modificación en los códigos que utilizamos.

Para comenzar, revisaremos en detalle las características de un LED. En pocas palabras, podemos decir que un **LED** es un diodo, es decir, un componente que deja pasar la electricidad en un solo sentido.

Si nos acercamos a un LED, veremos que presenta dos patillas, y una es más larga que la otra. La patilla más larga corresponde al **ánodo** o **polo positivo** mientras que la más corta es el **cátodo** o **polo negativo**. Como sabemos, la corriente entrará por el polo positivo y saldrá por el polo negativo.

+ LED



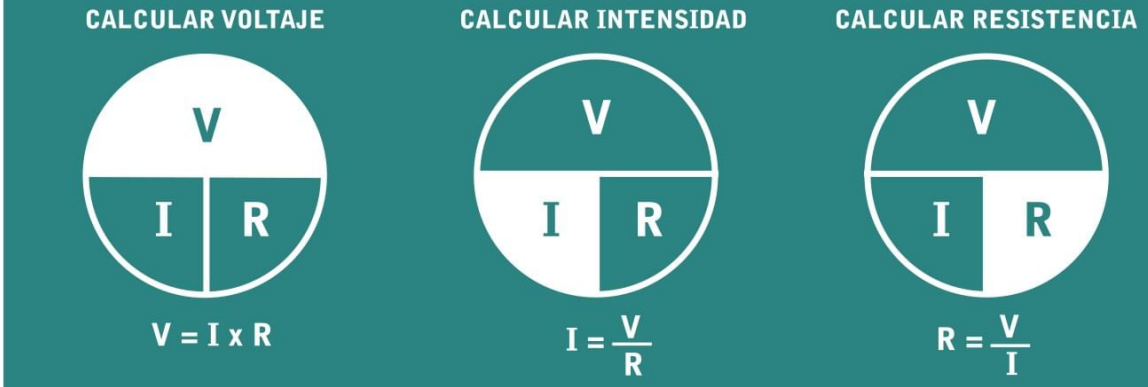
En este diagrama, se aprecian las partes que componen un LED, y la relación entre el ánodo y el cátodo.

Una vez que pongamos esto en claro, no tendremos problemas para conectar LEDs a los circuitos que diseñemos, aunque nos queda otro punto importante: las resistencias. En capítulos anteriores hemos mencionado la importancia de utilizar resistencias junto a los LEDs, pero ¿qué resistencia debemos utilizar en un caso específico? Para esta pregunta, no existe una respuesta directa pues es necesario hacer uso de la ley de Ohm.

Ley de Ohm

Esta ley se encarga de relacionar el voltaje, la corriente y la resistencia, y nos entega las fórmulas adecuadas para calcular cualquiera de estas magnitudes, aunque necesitaremos conocer dos de las magnitudes para calcular la tercera.

+ Ley de Ohm



Resumen de las intensidades que es posible calcular mediante la ley de Ohm.

Necesidad de una resistencia

Conectar una resistencia junto a un LED no es una tarea sin importancia, ya que se trata de una actividad completamente recomendable, pues, si no lo hacemos, podríamos dañar el LED o disminuir su vida útil. Es necesario considerar que, para un LED común, el voltaje de operación puede estar entre 1,8 V y 2,2 V; el voltaje del LED estará en la base de la elección de la resistencia, por esta razón es necesario que miremos sus características técnicas.

Por ejemplo, si tenemos un circuito sencillo en el que conectaremos un LED a la placa Arduino, necesitaremos calcular el valor de la resistencia para decidir cuál utilizar. Para ello aplicamos la ley de Ohm y, como necesitamos dos de las magnitudes, tenemos lo siguiente:

- **VOLTAJE**: los pines de la placa suministran 5 V.
- **INTENSIDAD**: podemos obtenerlo en la hoja de especificaciones o en la web oficial de Arduino; para la placa Arduino UNO se trata de 20 mA.

Teniendo estos datos, realizamos la siguiente reflexión: si la intensidad corresponde a 20 mA (miliamperios), o lo que es igual, 0,02 amperios; y el voltaje que caerá en la resistencia es de 5 V (un LED consume entre 1,8 V y 2,2 V), por una sencilla operación matemática deducimos que la resistencia debe consumir 3,2 V como mínimo.

Luego de esto aplicamos la fórmula de la ley de Ohm y escribimos los valores que tenemos: **R = V / I**

$$R = \frac{V}{I} = \frac{3{,}2\ V}{0{,}02\ A} = 160\ \Omega$$

Esto corresponde al valor mínimo; por supuesto, utilizar una resistencia de 220 Ω en este caso podría ser una buena idea. Con estos conceptos claros, pongamos manos a la obra.

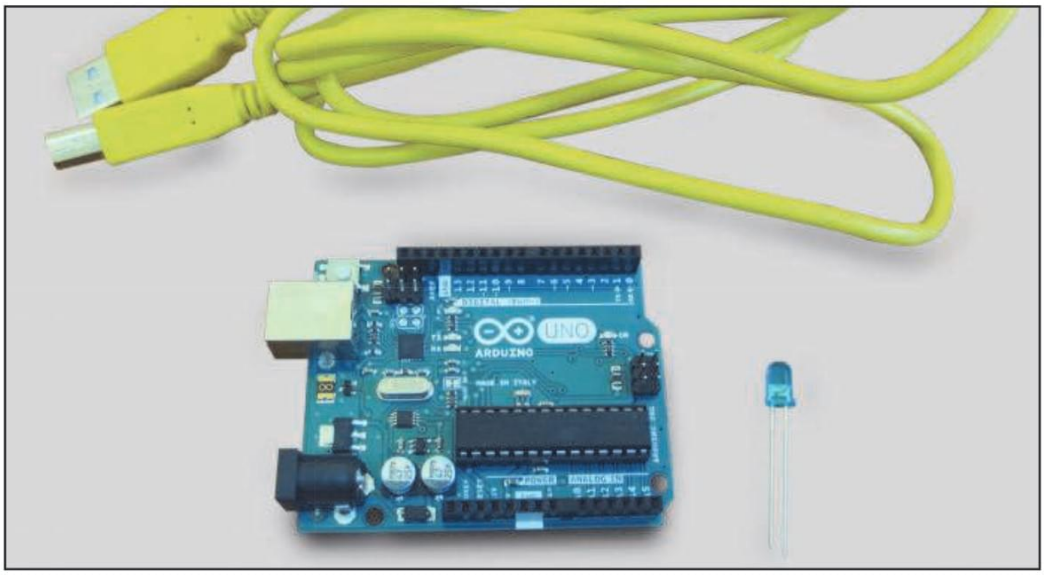
EL PROYECTO BÁSICO: BLINK

Ya conocimos la forma de cargar un ejemplo básico en nuestra placa Arduino UNO; para ello hicimos uso de Blink, uno de los códigos que se encuentran disponibles en el IDE. En esta ocasión trabajaremos con este código ya cargado en la placa, y lo profundizaremos y modificaremos para obtener diferentes resultados.

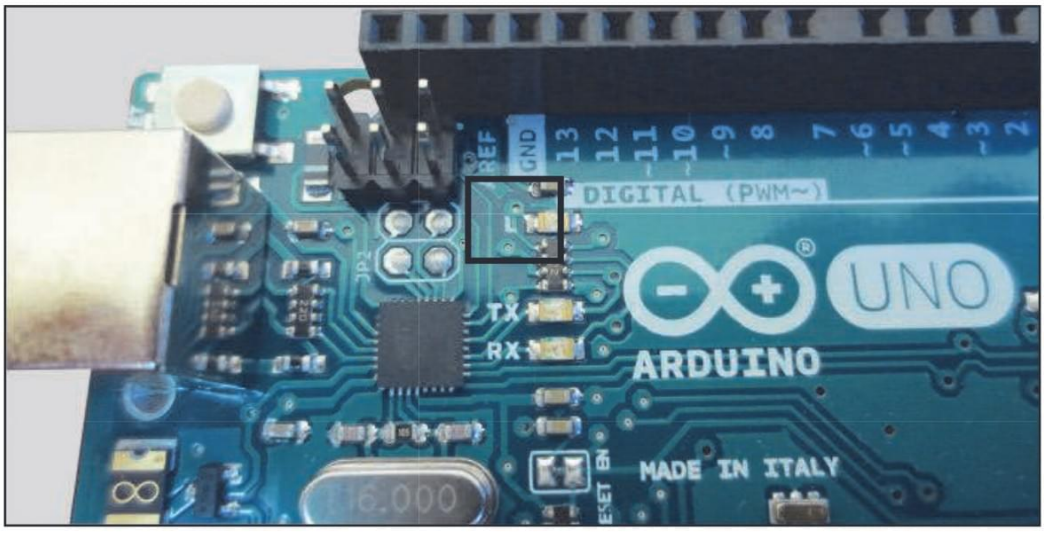
En primer lugar, efectuaremos la conexión de un LED a nuestra placa. En este punto es importante mencionar que, aunque una forma completa de conectar un LED es utilizando un protoboard, cables de puente y resistencia adecuada, también lo podemos conectar directamente a la placa. Para saber cómo hacerlo, sigamos las instrucciones del siguiente **Paso a paso**.

Paso a paso: Conectar un LED directamente

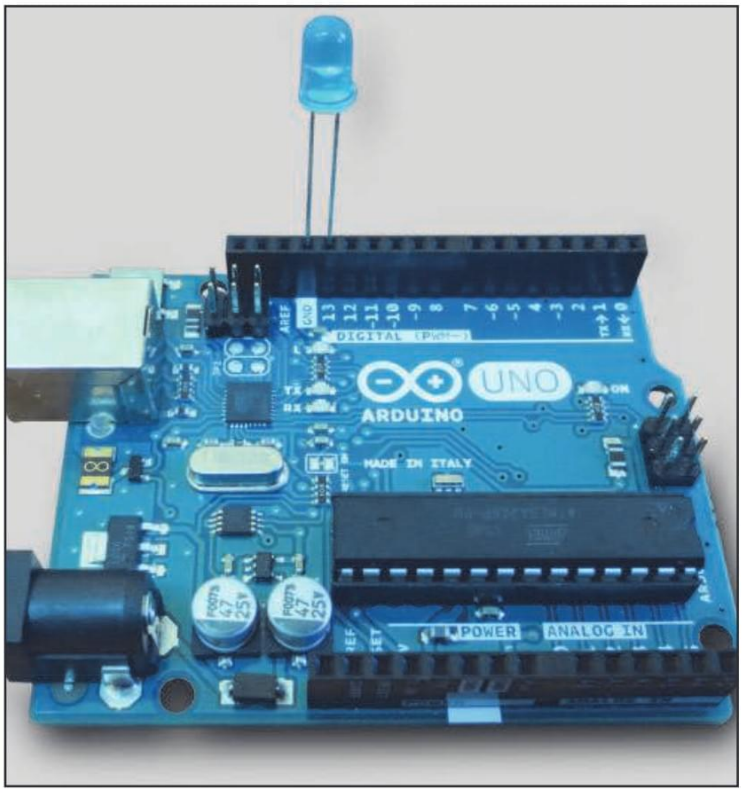
01 Como ya se completaron los pasos necesarios para conocer la placa Arduino UNO, se instaló el IDE y se conoció la forma correcta de cargar un sketch en la placa, en esta ocasión se revisará cómo se conecta un LED en forma directa. Necesitará la placa Arduino UNO, un LED y el cable USB para conectarla a la PC.



02 Antes, se verá la forma de conectar un LED mediante el protoboard, esta vez se hará en forma directa. En este punto es necesario mencionar que la placa ya dispone de un LED incorporado, por lo que es posible testear código aun si no se conecta un LED externo. Este LED se encuentra indicado con la letra L.



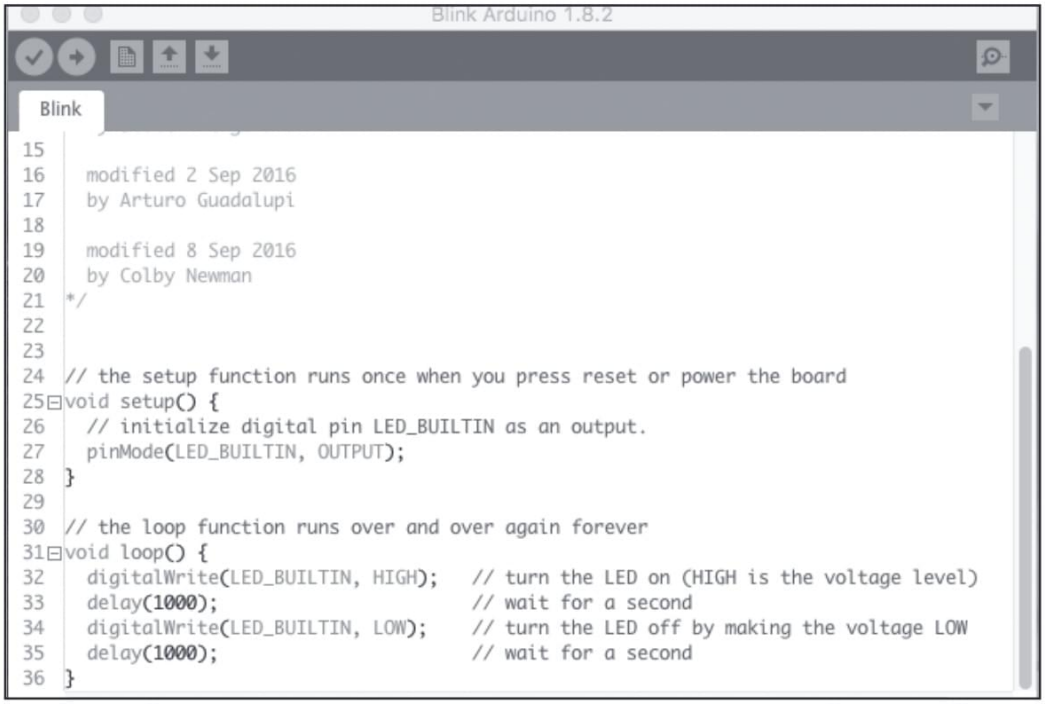
03



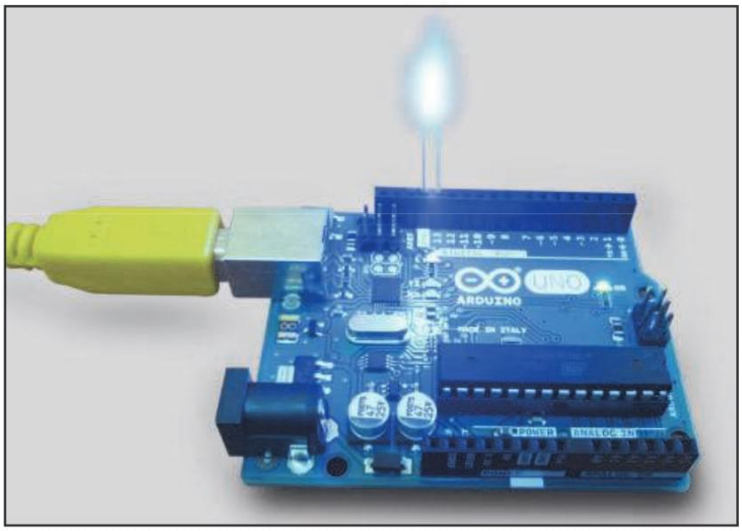
Para realizar la conexión de un LED en forma directa, conecte la pata más larga o positivo al ping digital 13 y la pata más corta al ping GND. Tenga en cuenta que la placa ya incorpora una resistencia en el ping 13, por lo tanto, no se dañará el LED conectado.

04

Ejecute el IDE de Arduino y cargue el ejemplo Blink, se trata del mismo código que se presentó en un capítulo anterior, por ahora no realice modificaciones en el código.



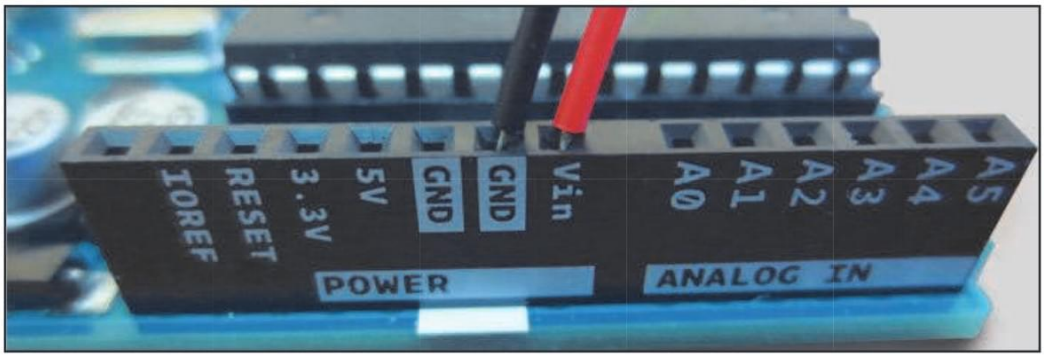
05



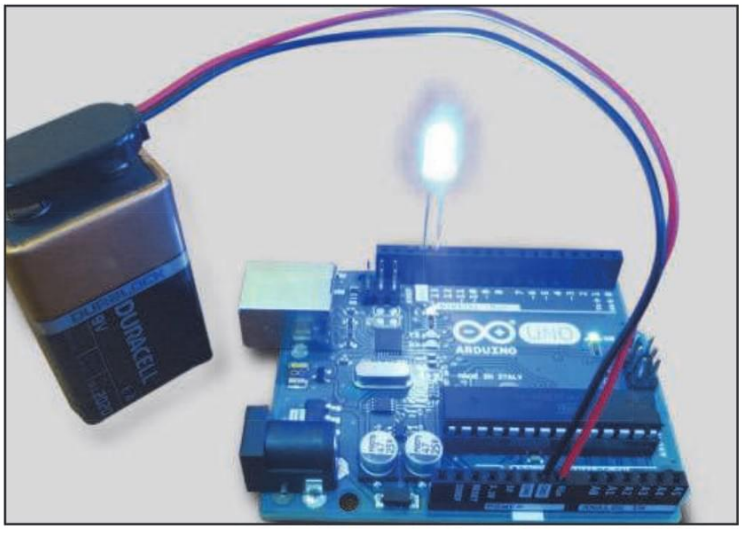
Conecte la placa a la computadora mediante el cable USB y cargue el sketch en la placa Arduino UNO. De inmediato verá que el LED parpadea, según las indicaciones del sketch Blink.

06

Ahora desconecte el cable USB, y la placa quedará sin energía, pero el sketch cargado seguirá en Arduino. En este punto es posible utilizar la energía de una fuente externa, por ejemplo, una batería de 9 V. Para efectuar la conexión, utilice los pines GND y Vin.



07



Con la batería conectada a la placa Arduino, se seguirá ejecutando el sketch que guardó, de esta forma el LED conectado directamente se encenderá y apagará según el código, un segundo encendido y un segundo apagado.

Podemos simplificar este código para que, haciendo uso del mínimo código que nos permita encender el LED, escribamos el siguiente sketch:

```
voidsetup(){
pinMode(13, OUTPUT);
}
voidloop(){
digitalWrite(13,HIGH);
}
```

"Si utilizamos un pin distinto al 13, será necesario conectar una resistencia de protección, de esta forma no dañaremos el led conectado."

Como vemos, se trata de un código muy sencillo, pero que nos servirá como punto de partida para entender lo que se encuentra en la base del encendido del LED.

Como sabemos, en **setup()** es posible establecer el uso que le daremos al microcontrolador o iniciar las variables que necesitamos en el sketch, por otra parte, en **loop()** podemos establecer los procesos que se ejecutarán en forma repetitiva.

De esta forma, en este código de ejemplo usamos **setup()** para establecer que el pin 13 entregue un voltaje cuando lo decidamos. Si seguimos analizando el código, vemos que utilizamos **loop()** para establecer el pin 13 en **HIGH**, así podemos obtener 5 V en él.

Para terminar, usamos **digitalWrite()** para iniciar el pin y establecer el estado **HIGH**, que mantendrá encendido el LED conectado. Si deseamos mantener el LED apagado, solo necesitamos efectuar una pequeña corrección en el código:

```
voidsetup(){
pinMode(13, OUTPUT);
}
voidloop(){
digitalWrite(13,LOW);
}
```

Ahora bien, para acercarnos al sketch Blink debemos conocer la función **delay()**, adecuada para lograr un retraso en la ejecución de un comando. Para usar esta función, indicamos entre paréntesis los milisegundos que durará el retraso, por ejemplo **delay(1000)** para un retraso de mil milisegundos o, lo que es igual, un segundo. Ahora veamos esta modificación en nuestro código:

```
voidsetup(){
pinMode(13, OUTPUT);
}
voidloop(){
digitalWrite(13,HIGH);
delay(500);
digitalWrite(13,LOW);
delay(500);
}
```

Como vemos, hemos utilizado **delay(500)**, que en la práctica se trata de medio segundo en el que el LED estará encendido y medio segundo en el que permanecerá apagado.

Ahora intentemos otras modificaciones, reduciendo el valor que indicamos en **delay()**, probemos, por ejemplo, con 400, 300, 200 y 100, respectivamente. En cada caso, carguemos el sketch modificado en la placa y observemos el funcionamiento del LED.

Al cargar este código, obtendremos un LED que parpadea rápidamente pues permanece una décima de segundo encendido y una décima de segundo apagado; para lograrlo utilizamos **delay(100)**.

CONTROLAR UN LED

Ya sabemos cómo encender un LED conectado a Arduino y, también, cómo controlar su parpadeo, para ello hicimos uso de **delay()** y modificamos el valor que le pasamos como parámetro.

Pero es tiempo de avanzar un poco más; para lograrlo integraremos la posibilidad de decidir en tiempo real cuándo se enciende o se apaga el LED.

Para este ejemplo seguiremos utilizando el LED que conectamos directamente a la placa Arduino UNO, mediante el pin 13.

Pero, antes de continuar con las instrucciones necesarias, debemos detenernos para conocer el **Monitor serial**. Se trata de una herramienta ofrecida por el IDE de Arduino, que nos permite enviar y ver los datos que se manejan a través del puerto serie. Es una forma sencilla y rápida de establecer una comunicación serial con la placa Arduino, por esta razón debemos utilizarlo para controlar el LED.

Para acceder al Monitor serial dentro del IDE de Arduino, es necesario que la tarjeta se encuentre conectada mediante el cable USB, de lo contrario, veremos un error que indica que la tarjeta no está disponible.

Para iniciar este tipo de comunicación, utilizaremos el siguiente código:

```
voidsetup(){
Serial.begin(9600);
}
voidloop(){
Serial.println('3');
delay(1000);
}
```

Analicemos las sentencias que aparecen y su utilidad:

- **Serial.begin(9600)**: esta sentencia se encarga de inicializar la comunicación serial. El número 9600 nos indica la cantidad de baudios (número y símbolos por segundo) que manejará el puerto serie. Utilizaremos esta sentencia siempre que necesitemos comunicarnos con Arduino mediante el puerto serie.
- **Serial.println('1')**: esta sentencia le indica, al microcontrolador, que debe imprimir un carácter a través del puerto serie. La ventaja de esta sentencia es que agrega un salto de línea cada vez que envía un dato; esto es muy útil cuando estemos utilizando el Monitor serial.

Si subimos este sencillo sketch a la placa Arduino, veremos que el LED TXT parpadeará cada vez que se envíe un dato a través del puerto serie, y, si desplegamos el Monitor serial, verificaremos que, al tiempo que se envía un dato, este se imprime en la ventana junto a un salto de línea. Ya estamos comunicándonos con la placa Arduino a través del puerto serie.

¿Programa o sketch?

Los códigos que desarrollamos para Arduino son denominados **sketch** y no programas, pero ¿cuál es la razón para utilizar una designación y no otra? IDE de Arduino se origina en Processing, un lenguaje en el que los códigos son considerados bocetos o sketchs. De esta forma Arduino ha heredado el IDE de Processing, también su forma de guardar el código y el nombre que utilizamos para los programas, es decir, sketch.

Ahora bien, partiendo de este código que nos permite efectuar la comunicación serial, realizaremos algunas modificaciones para declarar que usaremos el pin 13 como salida. Además, necesitaremos una de las estructuras selectivas que aprendimos en el **capítulo 5**, ya que debemos verificar la tecla que se presiona desde el teclado.

Para comenzar, declararemos el uso del pin 13, esto lo hacemos mediante:

```
pinMode(13, OUTPUT);
```

Luego, mediante un **if else** verificaremos el valor de **input**, en este ejemplo encenderemos el LED si el valor es **2**, de lo contrario lo apagaremos:

```
if (input=='2'){
      digitalWrite(13, HIGH);
    }
    else
    {
      digitalWrite(13, LOW);
    }
```

En este código utilizamos **if** para verificar el valor de **input**, si se trata de **2** ejecutaremos el código que está entre los corchetes: **digitalWrite(13, HIGH)**, es decir, encenderemos el LED. Por otra parte, usamos **else** para ejecutar el código **digitalWrite(13, LOW)**, cuando el valor de **input** sea distinto a **2**.

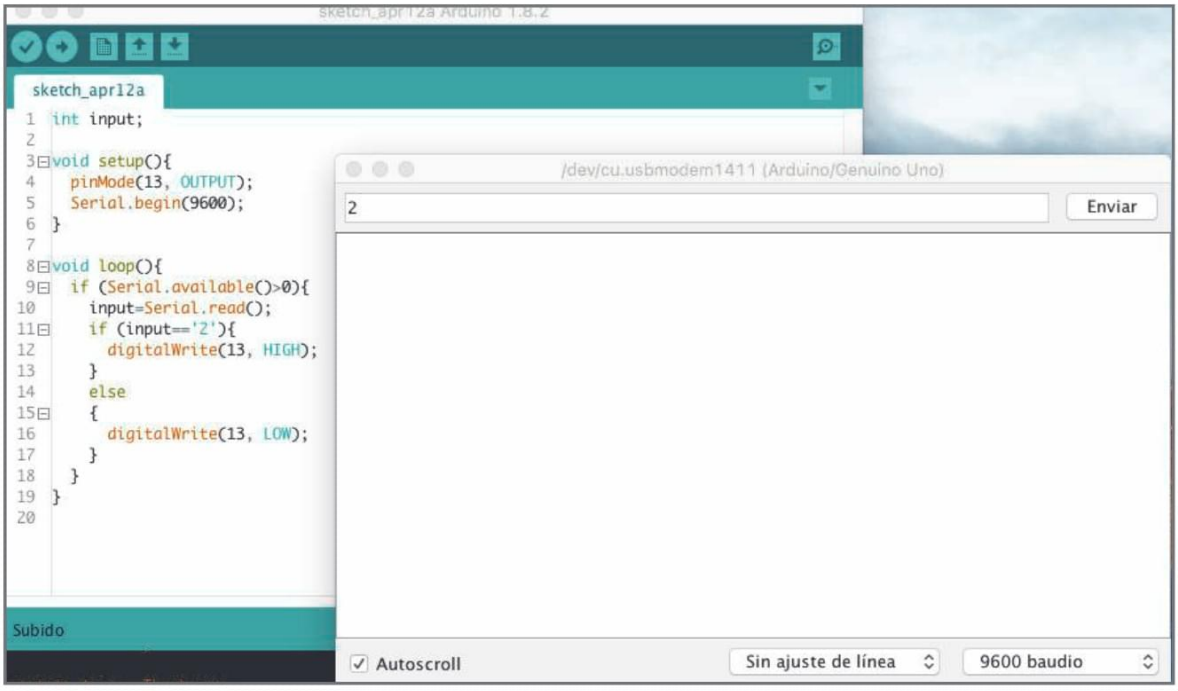
Debemos incluir la declaración de uso del pin 13 en **setup()**, mientras que la estructura **if else** irá en **loop()**:

```
int input;

voidsetup(){
  pinMode(13, OUTPUT);
  Serial.begin(9600);
}

voidloop(){
```

```
    if (Serial.available()>0){
      input=Serial.read();
      if (input=='2'){
        digitalWrite(13, HIGH);
      }
      else
      {
        digitalWrite(13, LOW);
      }
    }
  }
```

Manejamos el LED conectado directamente a la placa, utilizando el pin 13 y GND. Luego conectamos la placa a la computadora mediante el cable USB y subimos el sketch que acabamos de comentar.

Desplegamos el Monitor serie y escribimos **2**, luego presionamos **ENTER** y veremos que el LED se encenderá. Si realizamos el mismo procedimiento, pero presionando otra tecla, el LED se apagará.



Utilizaremos el Monitor serie para establecer una comunicación serial con Arduino; en este ejemplo encendemos el LED si presionamos **2** y luego **ENTER**; con otra tecla lo apagamos.

INCORPORAR ITERACIONES

Para complejizar un poco más el trabajo con LEDs, en Arduino podemos hacer uso de las iteraciones.

Una forma sencilla de conectar y encender varios LEDs sería repetir, tantas veces como necesitemos, las instrucciones que nos permiten encender un LED; de esta forma lograríamos como resultado la posibilidad de encenderlos todos. En primer lugar, será necesario declarar todos los pines requeridos en **setup()**, de la siguiente forma:

```
voidsetup()
   {
pinMode( 13, OUTPUT);
   pinMode( 12, OUTPUT);
   pinMode( 11, OUTPUT);
pinMode( 6, OUTPUT);
   }
```

Pero utilizar un ciclo **for** en esta declaración nos ahorrará código y hará que nuestro sketch sea más eficiente. Esto es posible pues sabemos que la instrucción básica debe ser repetida un número determinado de veces, por lo tanto, una instrucción **for** combinada con una variable que cambiará dependiendo de las veces que se ejecuten las instrucciones, será una alternativa perfecta para declarar los pines en **setup()**. Analicemos el siguiente código:

Pines

Debemos tener en cuenta que los pines digitales en Arduino tienen dos estados: **On** y **Off**. Generalmente utilizamos **HIGH** y **LOW** pues nos permiten leer el código en forma más sencilla. Los pines digitales entregan una potencia de salida de 5 V (HIGH), por lo que hay que tener cuidado al realizar conexiones a los pines digitales; en primer lugar, debemos estar seguros de que el elemento conectado es capaz de manejar 5 V.

```
voidsetup()
     {
int a = 0 ;
for ( a = 6 ; a < 14 ; a++)
pinMode( a , OUTPUT) ;
     }
```

Aquí inicializamos la variable **a** como **int**, es decir, como un entero; luego utilizamos el ciclo **for** para trabajar con esta variable, teniendo en cuenta tres parámetros:

- **variable**: en este caso se trata de la **variable a**, que se ha inicializado con un valor **0** (**a=0**), pero al entrar al ciclo **for** le asignamos un valor **6**: **i=6**.
- **comparación**: se trata de la condición que se debe satisfacer para que se ejecuten las condiciones dentro del ciclo **for**; en este caso se ejecutarán mientras la **variable a** sea menor que **14**: **a<14**.
- **cambio en la variable**: se refiere al incremento o decremento que sufrirá la variable con cada iteración del ciclo **for**; en este caso la variable **a** aumentará en **1** cada vez: **a++**.

Luego especificamos las instrucciones que se ejecutarán mientras la condición evaluada continúe siendo verdadera:

```
pinMode( a , OUTPUT);
```

Esta instrucción nos permite inicializar los pines que corresponden a cada valor que se almacenará en la variable **a**. En realidad, el ciclo **for** estará ejecutando las siguientes instrucciones, una por cada iteración:

```
pinMode( 6 , OUTPUT);
pinMode( 7 , OUTPUT);
pinMode( 8 , OUTPUT);
pinMode( 9 , OUTPUT);
pinMode( 10 , OUTPUT);
pinMode( 11 , OUTPUT);
pinMode( 12 , OUTPUT);
pinMode( 13 , OUTPUT);
```

Ya tenemos declarados los pines necesarios mediante un ciclo **for**, pero ¿podemos hacer lo mismo con **loop()**? Como trabajaremos con varios LEDs, utilizar un ciclo **for** en **loop()** puede ser una buena idea. Para implementarlo, recordemos el código que utilizamos para encender un solo LED en forma intermitente:

```
voidloop(){
digitalWrite(13,HIGH);
delay(500);
digitalWrite(13,LOW);
delay(500);
}
```

Gracias a este código, el LED conectado permanecerá encendido por medio segundo y apagado por otro medio segundo. Ahora bien, si quisiéramos encender los LEDs que hemos inicializado, podríamos copiar este código para cada LED que conectaremos, pero nuevamente obtendríamos una solución poco eficiente. Veamos cómo utilizar un **for** para resolverlo:

```
for ( b = 6 ; b < 14 ; b++)
   {
   digitalWrite( b , HIGH) ;
delay (500) ;
   digitalWrite( b , LOW);
   delay (500) ;
   }
```

En este caso utilizamos los siguientes parámetros:

- **variable**: usamos la variable **b**, a la que asignamos el valor **6**: **b=6**.
- **comparación**: indicamos que las instrucciones dentro del ciclo se ejecuten mientras la variable **b** sea menor que **14**: **b<14**.
- **cambio en la variable**: en este caso, la variable **b** aumentará en **1** con cada iteración: **b++**.

Dentro del ciclo, encontramos las instrucciones que serán ejecutadas en cada iteración:

```
// Proporciona electricidad al pin almacenado en b
digitalWrite( b , HIGH) ;
// Mantiene encendido el LED por medio segundo
delay (500) ;
// Quita la electricidad al pin almacenado en b
digitalWrite( b , LOW);
// Mantiene apagado el LED por medio segundo
delay (500) ;
```

Estas instrucciones se repiten para cada uno de los pines que declaramos en **setup()**, por lo tanto, el resultado será que todos los LEDs conectados se encenderán y se apagarán. Nuestro código completo queda de la siguiente forma:

```
voidsetup()
     {
int a = 0 ;
for ( a = 6 ; a < 14 ; a++)
pinMode( a , OUTPUT) ;
     }

voidloop()
  {
int b = 0 ;
for ( b = 6 ; b < 14 ; b++)
          {
digitalWrite( b , HIGH) ;
delay (500) ;
digitalWrite( b , LOW);
delay (500) ;
          }
  }
```

Ahora solo nos queda escribirlo en el IDE de Arduino y proceder a su compilación, así podremos verificar que no existan errores de sintaxis.

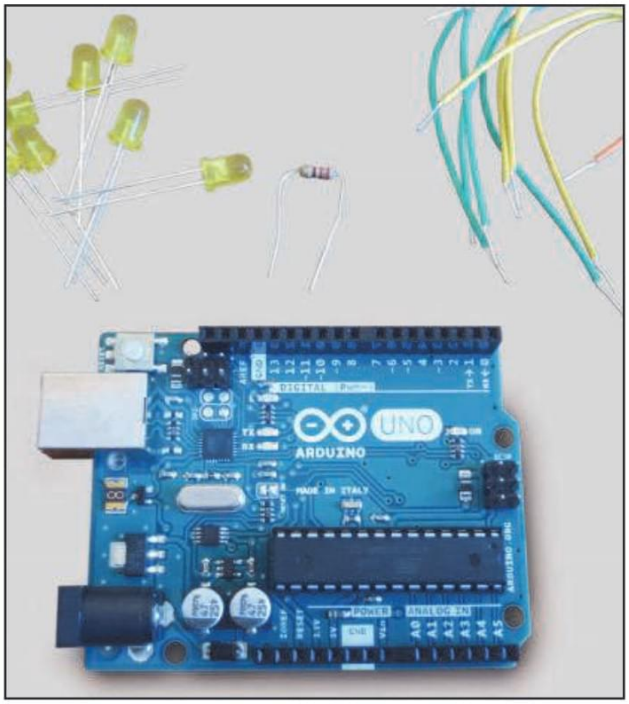
```
sketch_apr12b
1  void setup()
2      {
3          int a = 0 ;
4          for ( a = 6 ; a < 14 ; a++)
5              pinMode( a , OUTPUT) ;
6      }
7
8  void loop()
9    {
10         int b = 0 ;
11         for ( b = 6 ; b < 14 ; b++)
12           {
13             digitalWrite( b , HIGH) ;
14             delay (500) ;
15             digitalWrite( b , LOW);
16             delay (500) ;
17           }
18   }
19
```

Luego de escribir el código, hacemos clic en **Verificar/Compilar**, que se encuentra en el menú **Programa**. Como vemos en esta imagen, la compilación ha sido correcta.

Ahora earmaremos el circuito, para ello necesitaremos la placa Arduino, un protoboard, cables de puente, ocho LEDs y una resistencia. Debemos seguir las instrucciones del siguiente **Paso a paso**.

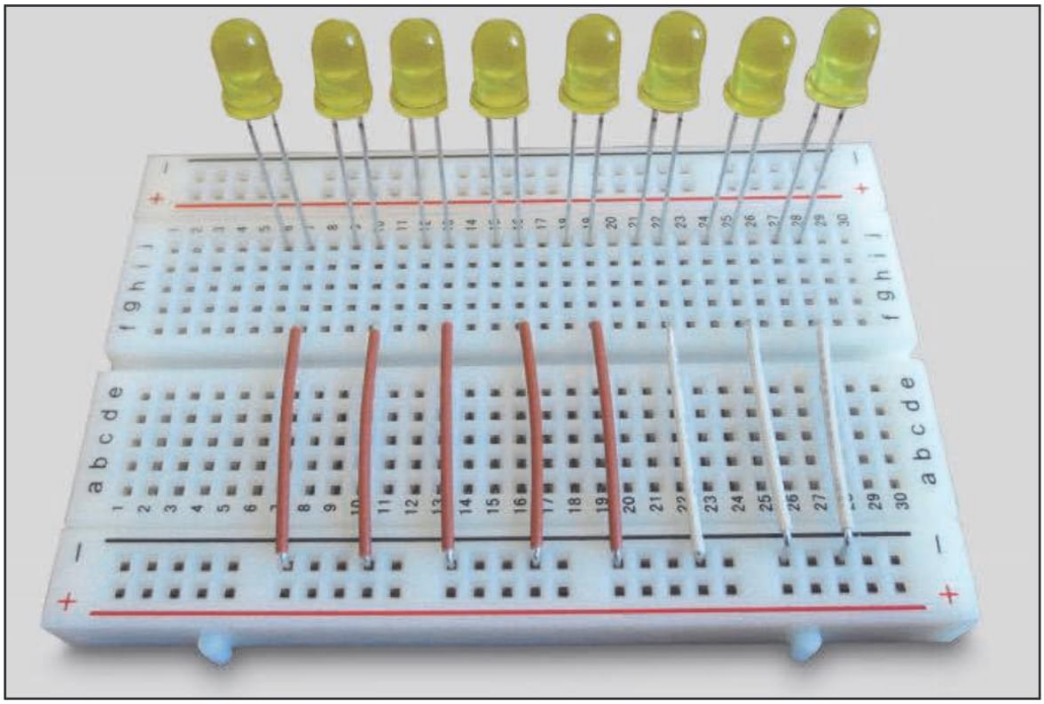
Paso a paso: Conectar varios LEDs

01

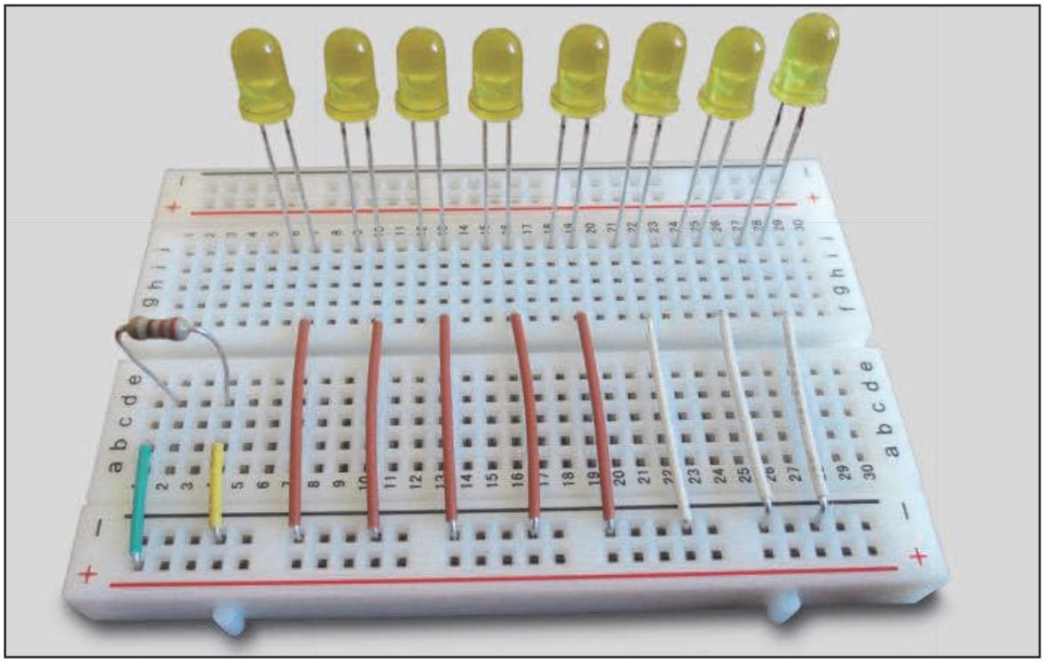


Para comenzar, reúna los materiales que necesite para llevar a cabo las instrucciones entregadas en este Paso a paso: protoboard, cables de puente, LEDs, placa Arduino y cable USB.

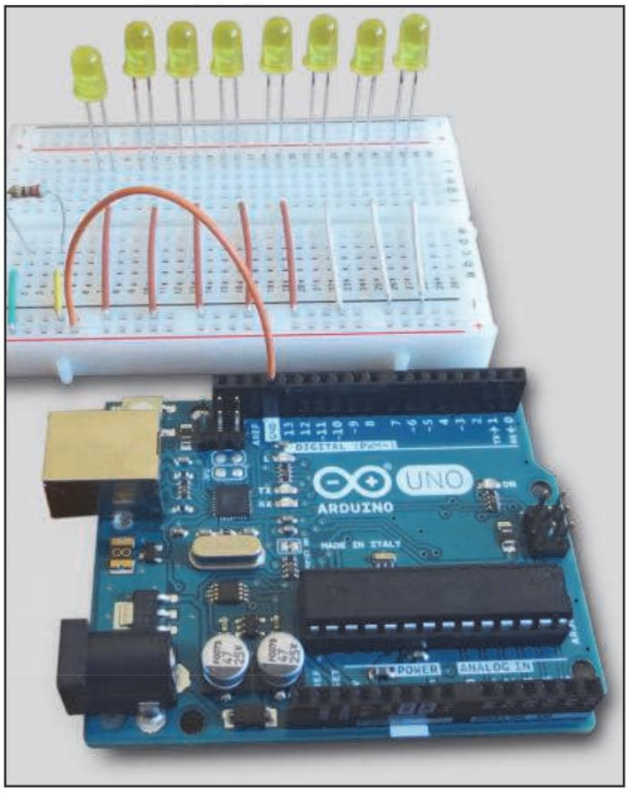
02 Conecte los LEDs, al protoboard, luego utilice cables de puente para unir el extremo negativo de cada LED con la línea negativa del protoboard.



03 Ahora agregue una resistencia. A continuación, una cada uno de sus extremos con las líneas negativa y positiva que se encuentra en uno de los extremos del protoboard.



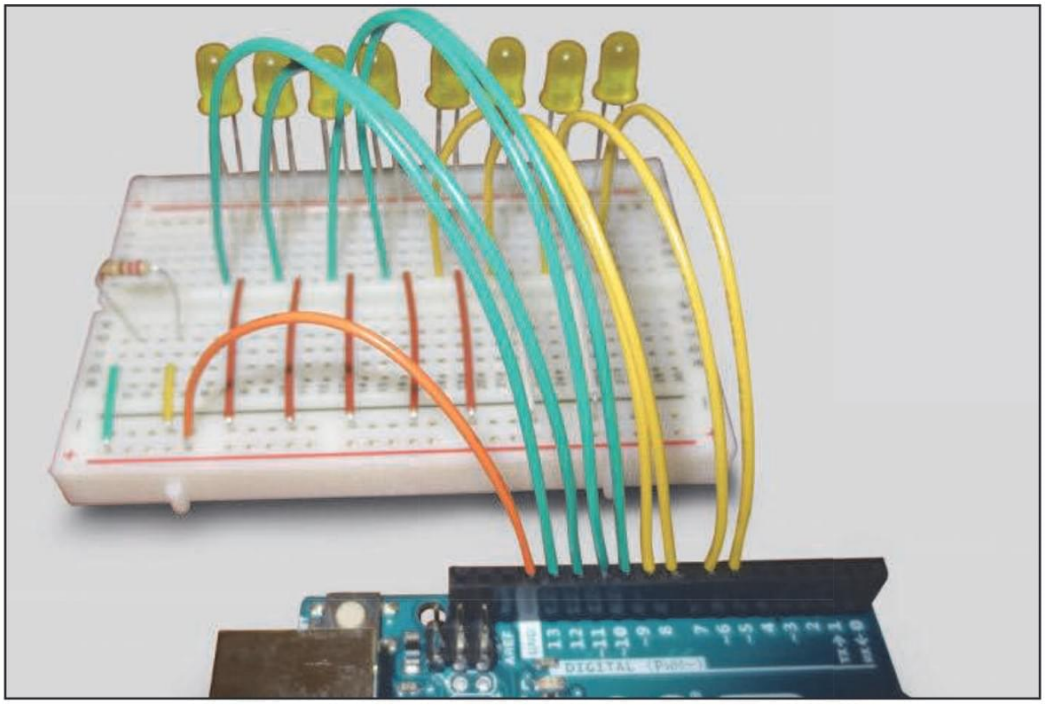
04



Es tiempo de unir el protoboard a la placa Arduino UNO. Para esto, use un cable de puente que irá desde el pin GND hasta la línea positiva del protoboard.

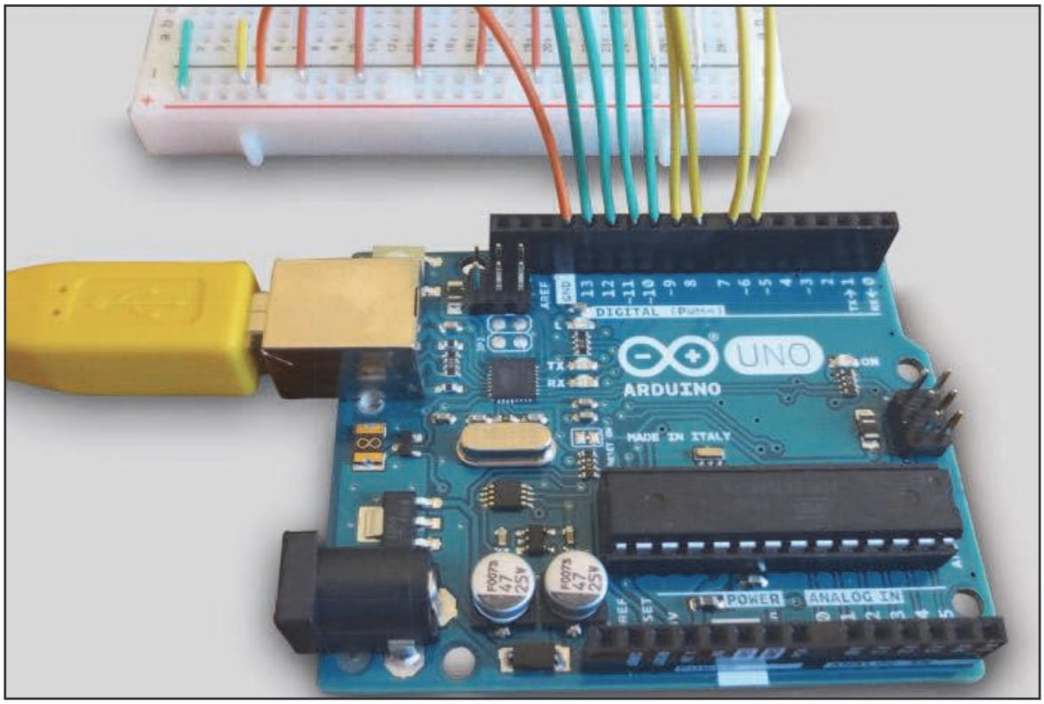
05

Para continuar, utilice los cables de puente necesarios para conectar el extremo positivo de cada LED con los pines 6 al 13 de la placa Arduino.

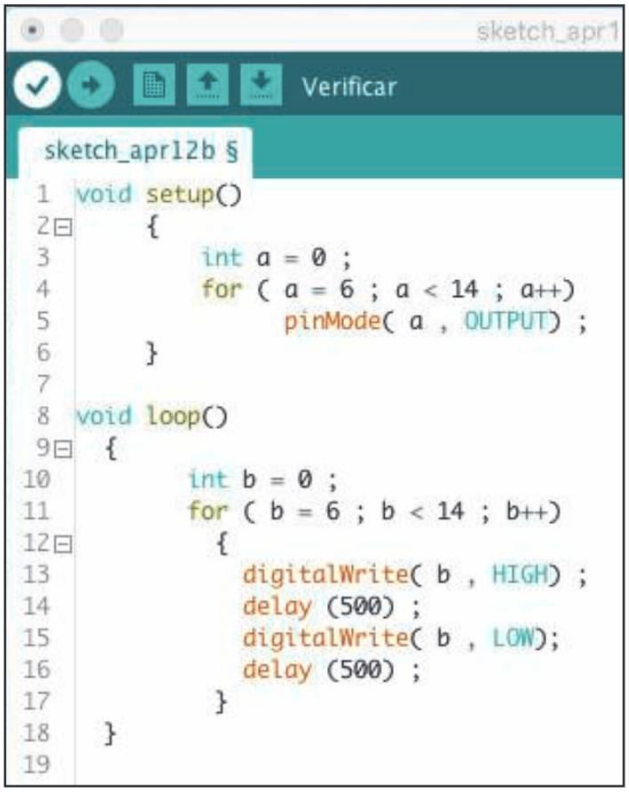


06

Con todas las conexiones ya realizadas, conecte la placa Arduino a la computadora, para ello utilice el cable USB.



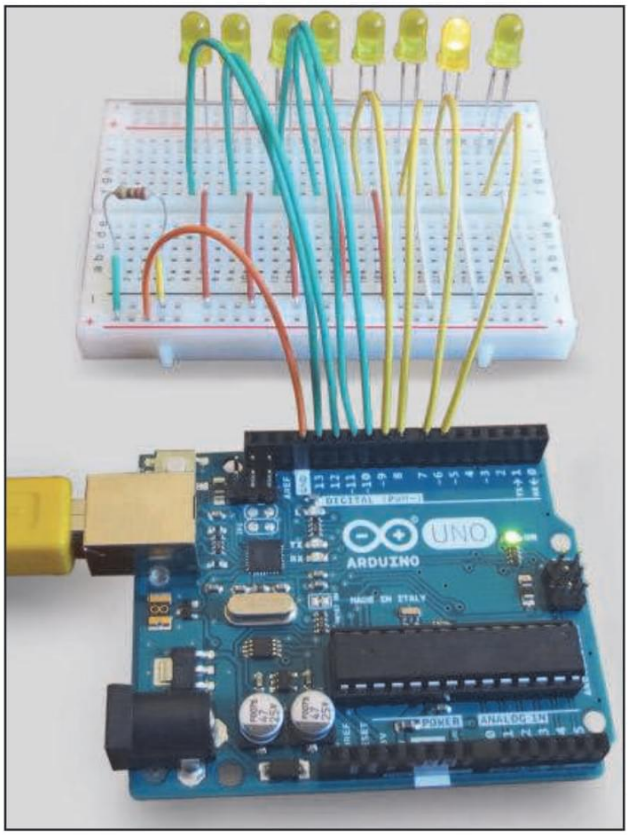
07




```
1  void setup()
2  {
3      int a = 0 ;
4      for ( a = 6 ; a < 14 ; a++)
5          pinMode( a , OUTPUT) ;
6  }
7
8  void loop()
9  {
10     int b = 0 ;
11     for ( b = 6 ; b < 14 ; b++)
12     {
13       digitalWrite( b , HIGH) ;
14       delay (500) ;
15       digitalWrite( b , LOW);
16       delay (500) ;
17     }
18 }
19
```

Suba el sketch a la placa Arduino, en este punto verifique que la compilación sea correcta, y solucione cualquier posible error de sintaxis que pudiera aparecer.

08



Si la carga del sketch se realiza en forma correcta, debería ver los LEDs parpadeando uno a la vez, siguiendo una secuencia ordenada y con intermitencias de medio segundo.

09

sketch_apr12b

```
1  void setup()
2      {
3          int a = 0 ;
4          for ( a = 6 ; a < 14 ; a++)
5                pinMode( a , OUTPUT) ;
6      }
7
8  void loop()
9    {
10         int b = 0 ;
11         for ( b = 6 ; b < 14 ; b++)
12           {
13             digitalWrite( b , HIGH) ;
14             delay (10) ;
15             digitalWrite( b , LOW);
16             delay (10) ;
17           }
18   }
19
```

Para lograr otros efectos, puede tocar el código y subirlo nuevamente a la placa Arduino. Por ejemplo, para que el juego de luces vaya más rápido, reemplace las líneas `delay(500)` por `delay(10)`. Cargue el nuevo sketch y vea cómo funciona.

Teniendo como base este código, es posible generar algunas modificaciones para lograr efectos interesantes, por ejemplo, analicemos el siguiente sketch:

```
voidsetup()
     {
int a = 0 ;
for ( a = 6 ; a < 14 ; a++)
pinMode( a , OUTPUT) ;
     }

voidloop()
  {
int b = 0 ;
for ( b = 6 ; b < 12 ; b++)
          {
digitalWrite( b , HIGH) ;
delay (50) ;
digitalWrite( b , LOW);
delay (50) ;
          }
for ( b = 13 ; b > 6 ; b--)
          {
digitalWrite( b , HIGH) ;
delay (50) ;
digitalWrite( b , LOW);
delay (50) ;
          }
}
```

Al subirlo a la placa veremos que las luces LEDs se encienden en forma progresiva, pero llegan hasta el final de la línea y vuelven, encendiéndose una a una. Otra posibilidad es la siguiente:

```
voidsetup()
     {
int a = 0 ;
for ( a = 6 ; a < 14 ; a++)
```

```
pinMode( a , OUTPUT) ;
     }

voidloop()
  {
int b = 0 ;
for ( b = 6 ; b < 14 ; b=b+2)
          {
digitalWrite( b , HIGH) ;
delay (100) ;
digitalWrite( b , LOW);
delay (100) ;
          }
for ( b = 5 ; b < 14 ; b=b+2)
          {
digitalWrite( b , HIGH) ;
delay (100) ;
digitalWrite( b , LOW);
delay (100) ;
          }
```

En este caso, los LEDs se encenderán, pero saltándose un lugar, posteriormente repite la secuencia, pero se irán encendiendo los LEDs que en el primer ciclo permanecieron apagados.

ENCENDER SEIS LEDS EN SECUENCIA

Realizaremos un sistema de conexión distinto al que venimos utilizando para los LEDs. Utilizaremos seis LEDs de diferentes colores, cada uno conectado a su propia resistencia; los LEDs se activarán en secuencia, ya sea desde el exterior hasta el centro o viceversa; se encienden los LEDs de igual color en cada iteración. Ubicaremos los LEDs de forma que los colores aparezcan como una secuencia organizada de luces. Para completar estas instrucciones, necesitamos lo siguiente:

- **Dos LEDs azules.**
- **Dos LEDs rojos.**
- **Dos LEDs amarillos.**
- **Seis resistencias de 220 ohms.**
- **Placa Arduino UNO.**
- **Protoboard.**
- **Cables de puente.**

setup()

Como hicimos en los proyectos anteriores, en primer lugar es necesario configurar los pines que utilizaremos; en este caso se trata de los pines 4 al 9 como **OUTPUT**. Como ya sabemos, es posible hacerlo de la siguiente forma:

```
pinMode( 4 , OUTPUT);
pinMode( 5 , OUTPUT);
pinMode( 6 , OUTPUT);
pinMode( 7 , OUTPUT);
pinMode( 8 , OUTPUT);
pinMode( 9 , OUTPUT);
```

Pero podemos hacer que este código sea más eficiente gracias a un ciclo **for**:

```
voidsetup()
     {
intLed = 0 ;
for ( Led = 4 ; Led< 10 ; Led++)
pinMode( Led , OUTPUT) ;
     }
```

loop()

Ahora que ya hemos configurado los pines que utilizaremos, es tiempo de trabajar sobre **loop()**. En esta ocasión, trabajaremos con dos esquemas de código, uno para controlar los primeros tres LEDs y el segundo para controlar los tres LEDs restantes.

El primer bloque de código que utilizaremos corresponde al grupo1, es el que manejará los tres primeros LEDs, conectados a los pines 4, 5 y 6. El código de esta función será el siguiente:

```
for (intLed = 4; Led< 7; Led++)
      {
digitalWrite(Led, HIGH);
digitalWrite(Led + 2 * paso + 1, HIGH);
delay(200);
digitalWrite(Led, LOW);
digitalWrite(Led + 2 * paso + 1, LOW);
delay(200);
        paso--;
       }
```

Ahora trabajaremos con el grupo2, el que controlará al segundo grupo de LEDs, es decir, a los que se encuentran conectados a los pines 7, 8 y 9.

```
        paso = 0;
for (intLed = 6; Led> 3; Led--)
           {
digitalWrite(Led, HIGH);
digitalWrite(Led + 2 * paso + 1, HIGH);
delay(200);
digitalWrite(Led, LOW);
digitalWrite(Led + 2 * paso + 1, LOW);
delay(200);
        paso++;
           }
```

Con esto, nuestro código quedará de la siguiente forma:

```
voidsetup()
     {
intLed = 0 ;
for ( Led = 4 ; Led< 10 ; Led++)
pinMode( Led , OUTPUT) ;
     }
```

```
voidloop()
  {
int paso = 2;
for (intLed = 4; Led< 7; Led++)
      {
digitalWrite(Led, HIGH);
digitalWrite(Led + 2 * paso + 1, HIGH);
delay(200);
digitalWrite(Led, LOW);
digitalWrite(Led + 2 * paso + 1, LOW);
delay(200);
        paso--;
       }
        paso = 0;
for (intLed = 6; Led> 3; Led--)
           {
digitalWrite(Led, HIGH);
digitalWrite(Led + 2 * paso + 1, HIGH);
delay(200);
digitalWrite(Led, LOW);
digitalWrite(Led + 2 * paso + 1, LOW);
delay(200);
        paso++;
           }
  }
```

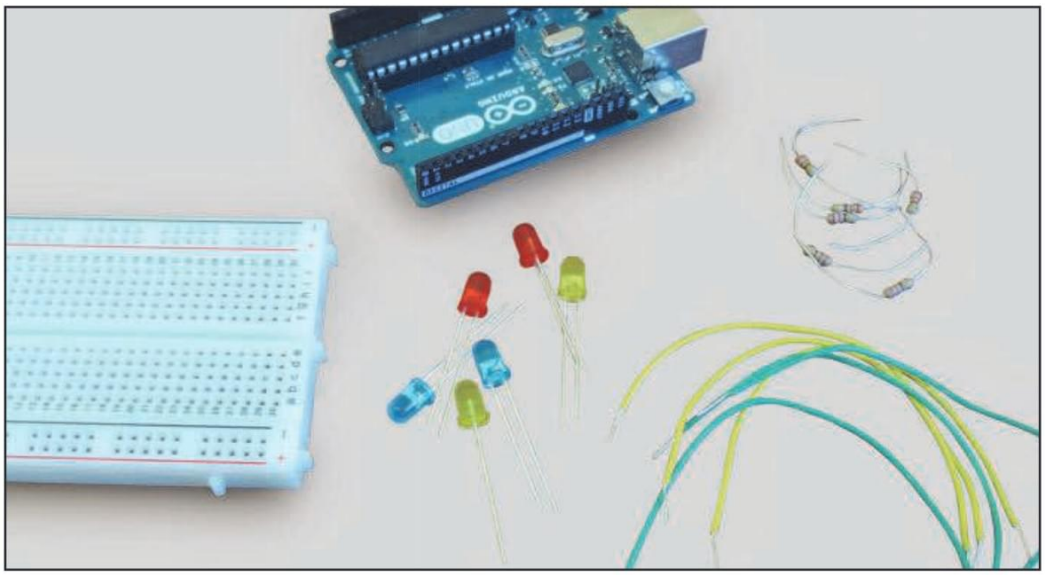
Ahora debemos iniciar el IDE de Arduino para ingresar el código y, luego, compilarlo para detectar cualquier problema que pueda aparecer. A continuación, armaremos el circuito adecuado para ejecutar el código que ya desarrollamos.

"Luego de escribir cualquier código debemos compilarlo para detectar posibles errores."

Paso a paso: Circuito para encender 6 LEDs

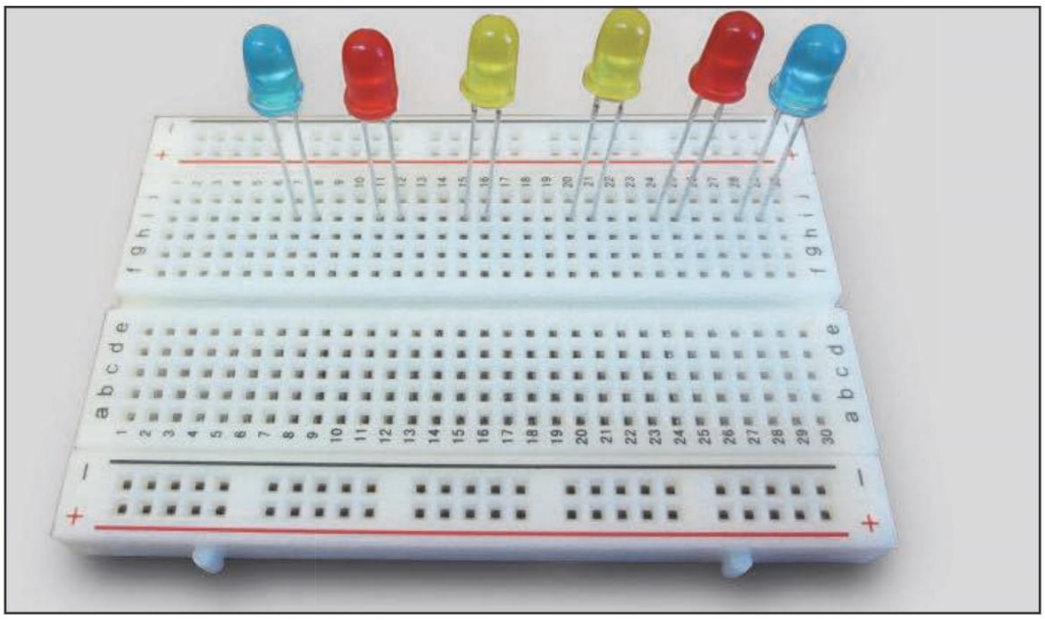
01

Para comenzar, reúna los elementos que necesitará para armar este circuito, en este caso se trata de seis LEDs en tres colores diferentes, seis resistencias, cables de puente, Arduino UNO y protoboard.



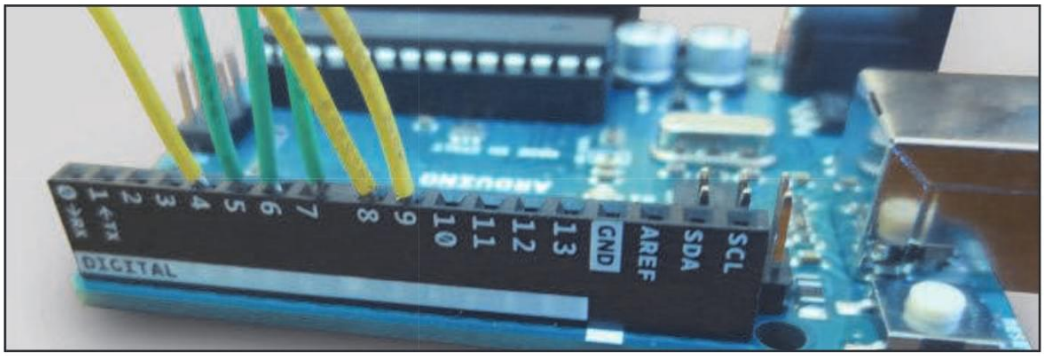
02

Conecte los LEDs en el protoboard, deberán quedar conectados en línea en el siguiente orden: azul, rojo, amarillo, amarillo, rojo, azul. Por comodidad, puede dejar algunos espacios entre cada LED.



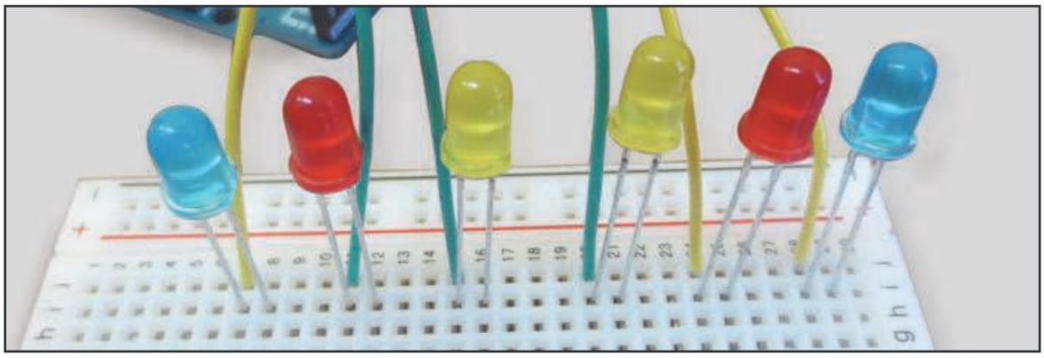
03

Como se declaró el uso de los pines 4 al 9, conecte cables de puentes a los pines adecuados en la tarjeta Arduino UNO.



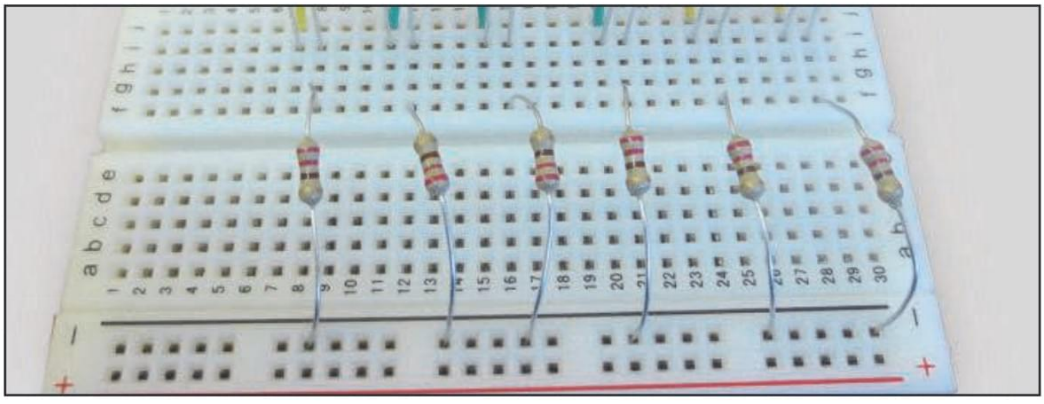
04

Para continuar, conecte los cables de puente a los LEDs que ubicó en el protoboard. Comience por el pin 4, que irá conectado al primer LED azul, el pin 5 que irá conectado al primer LED rojo y el pin 6, que irá conectado al primer LED amarillo. Posteriormente, los pines 7, 8 y 9 deberán conectarse a los segundos LEDs amarillo, rojo y azul.



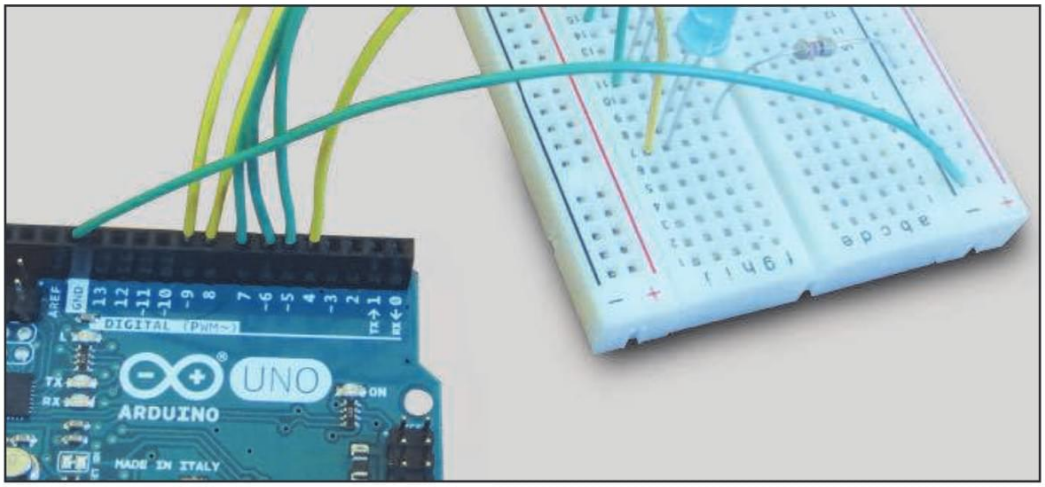
05

Conecte las resistencias para unir la línea negativa de la parte inferior del protoboard con los LEDs. Utilice una resistencia para cada LED.



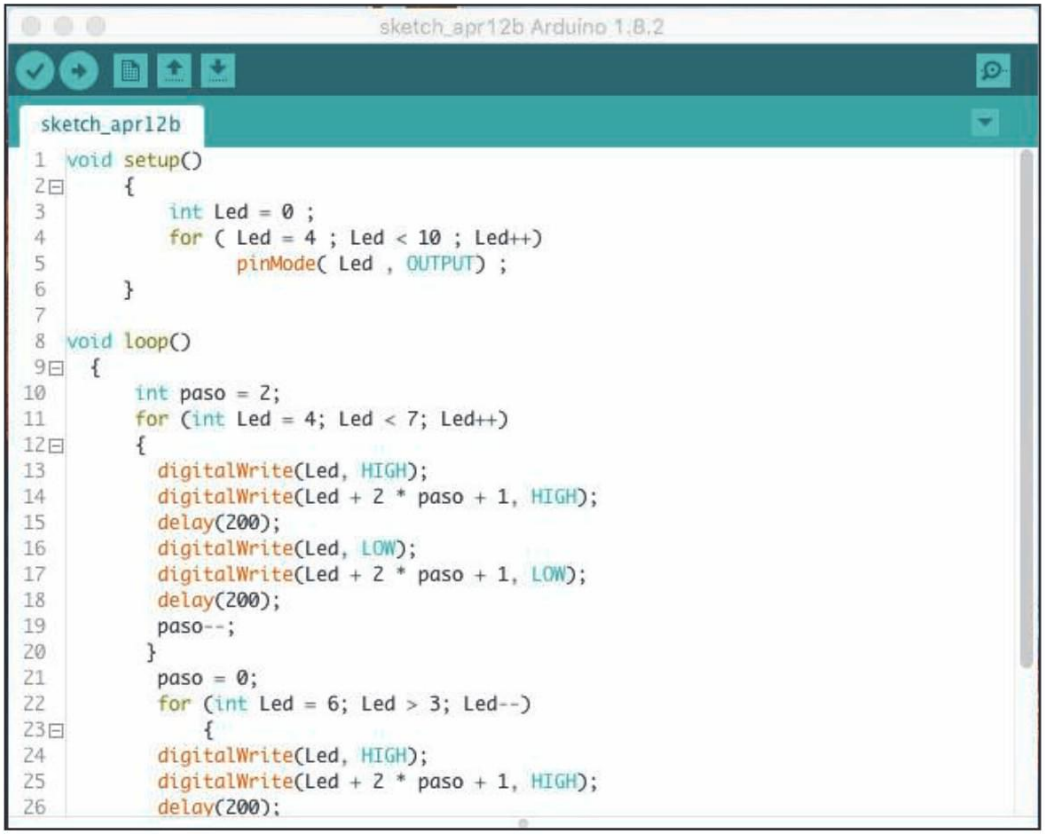
06

Conecte uno de los extremos de un cable de puente a la línea negativa del protoboard. Conecte el otro extremo al pin GND de la placa Arduino UNO.



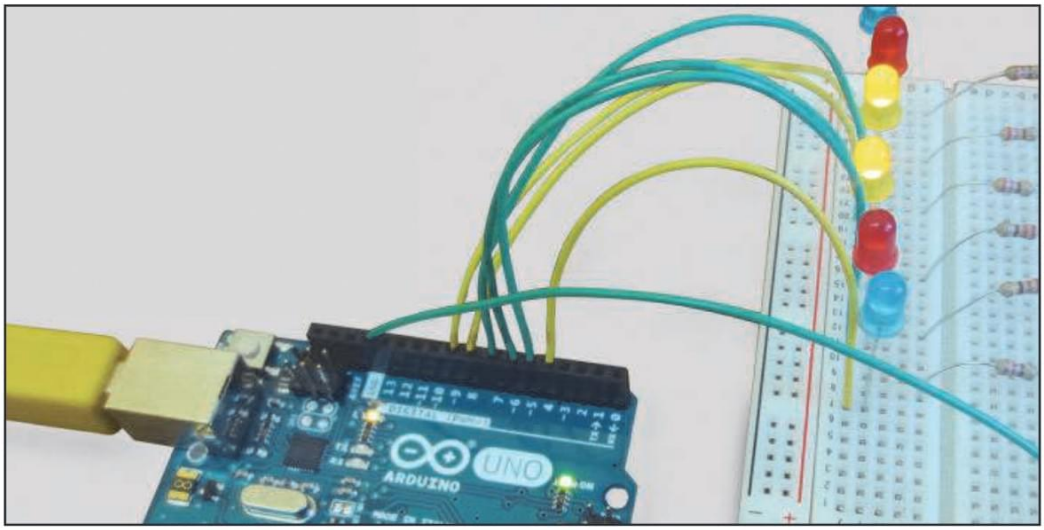
07

Con las conexiones ya realizadas, debe compilar el sketch que desarrolló antes. Verifique que la compilación se realice sin problemas, de lo contrario busque y solucione los posibles errores que puedan presentarse.




```
1  void setup()
2      {
3          int Led = 0 ;
4          for ( Led = 4 ; Led < 10 ; Led++)
5                pinMode( Led , OUTPUT) ;
6      }
7
8  void loop()
9    {
10       int paso = 2;
11       for (int Led = 4; Led < 7; Led++)
12       {
13         digitalWrite(Led, HIGH);
14         digitalWrite(Led + 2 * paso + 1, HIGH);
15         delay(200);
16         digitalWrite(Led, LOW);
17         digitalWrite(Led + 2 * paso + 1, LOW);
18         delay(200);
19         paso--;
20        }
21         paso = 0;
22         for (int Led = 6; Led > 3; Led--)
23             {
24         digitalWrite(Led, HIGH);
25         digitalWrite(Led + 2 * paso + 1, HIGH);
26         delay(200);
```

08 Conecte la placa Arduino UNO a la computadora mediante el cable USB y cargue el sketch mediante la opción `Subir`, que se encuentra en la barra superior de opciones. Con el sketch cargado, verá que los LEDs se encenderán en secuencia: los dos azules, luego los dos rojos y finalmente los dos amarillos.



SECUENCIA DE 8 LEDS

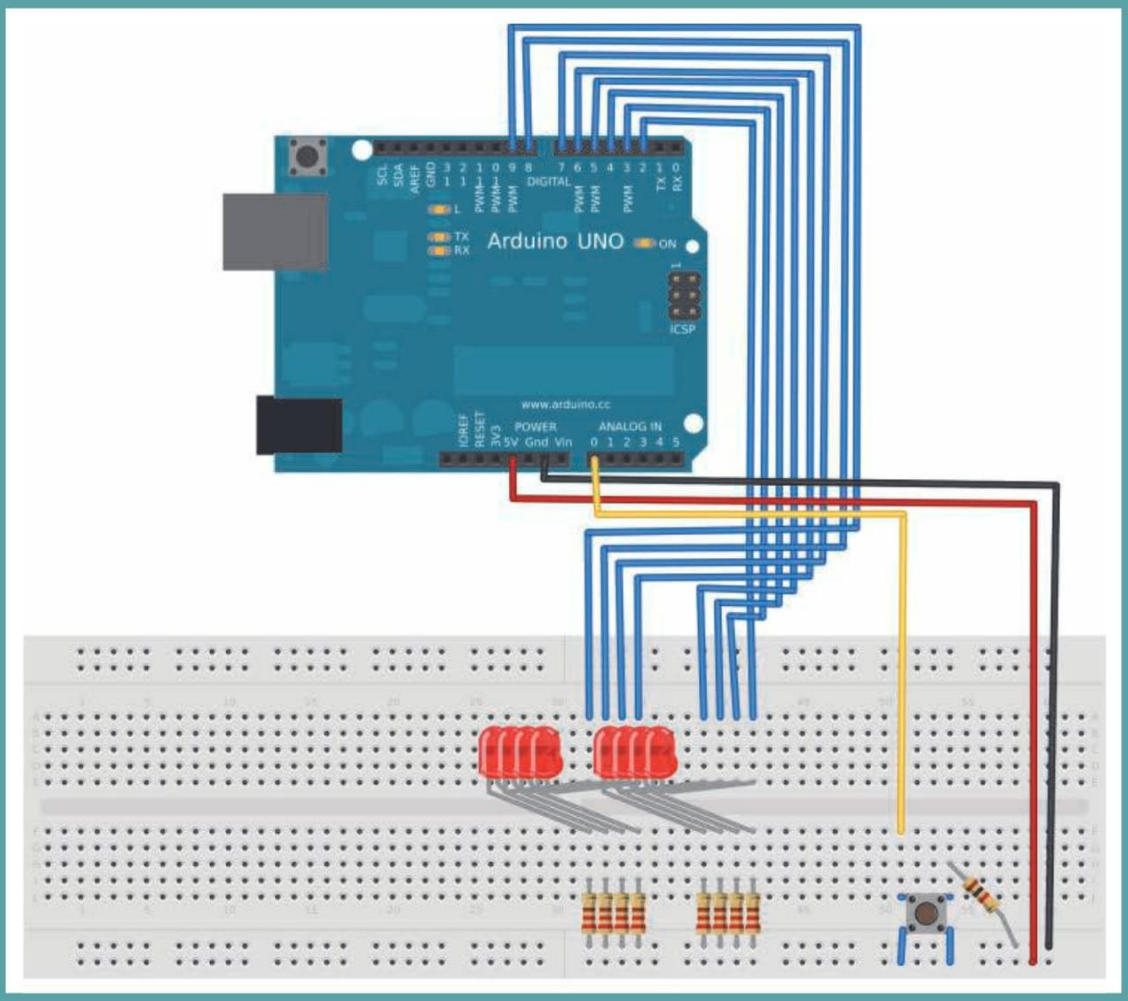
Para complejizar un poco lo que venimos haciendo con Arduino, trabajaremos con un grupo de ocho LEDs, pero en esta ocasión lo activaremos con diferentes secuencias de encendido e incorporaremos un pulsador que nos permitirá cambiar el tipo de secuencia que se utiliza.

En primer lugar realizaremos las conexiones necesarias para que nuestro circuito funcione; para ello necesitaremos la placa Arduino UNO, un protoboard, ocho LEDs, ocho resistencias de 220 Ω y una de 1 K, además de un pulsador.

Antes de conectar los diferentes elementos, hay que considerar lo siguiente. Para los LEDs, de un lado debemos conectar todos los cátodos a las resistencias de 220 Ω, las que a su vez se conectarán a GND, por otra parte, conectaremos los ánodos a los pines 2 al 9 de la placa Arduino.

Por su parte, el pulsador se debe conectar de un lado a 5 V y del otro a una resistencia a tierra. Necesitamos un cable conectado al pin A0 para permitir cambiar de secuencia cada vez que el pulsador sea presionado.

+ Diagrama de conexión



En esta imagen vemos el diagrama de conexión propuesto para este proyecto.

A continuación, analizaremos un ejemplo de código que podemos utilizar para encender los 8 LEDs en diferentes secuencias:

```
int saltar=0;
voidsetup() {

pinMode(A0, INPUT);
for(int i=2;i<=9;i++){
```

```
pinMode(i, OUTPUT);
}
}

voidloop() {
if (digitalRead(A0)==HIGH){
saltar++; // Cambia de secuencia
if (saltar>3){
saltar=0;
}
while (digitalRead(A0)==HIGH){}
}

if(saltar==0){
secuencia1();
}
if(saltar==1){
secuencia2();
}
if(saltar==2){
secuencia3();
}
if(saltar==3){
secuencia4();
}

}
```

Ahora nos ocuparemos de programar las diversas secuencias de encendido para los 8 LEDs. Para la primera secuencia usaremos lo siguiente:

```
void secuencia1(){
for (int i=2; i<=9; i++){ //Pin 2 al 9
digitalWrite(i, HIGH);
digitalWrite(i-1,LOW);
delay(50);
```

```
}
for (int i=9; i>=2; i--){
digitalWrite(i, LOW);
digitalWrite(i-1,HIGH);
delay(50);
}
}
```

Para crear la segunda secuencia, utilizaremos el siguiente código:

```
void secuencia2(){
int k=11;
for(int i=6; i<=9;i++){
digitalWrite(i, HIGH);
digitalWrite(k-i, HIGH);
delay(50);
}
for(int i=9; i>=2;i--){
digitalWrite(i, LOW);
digitalWrite(k-i, LOW);
delay(50);
}
}
```

Ahora crearemos la secuencia de encendido número tres, podemos utilizar un código como el siguiente:

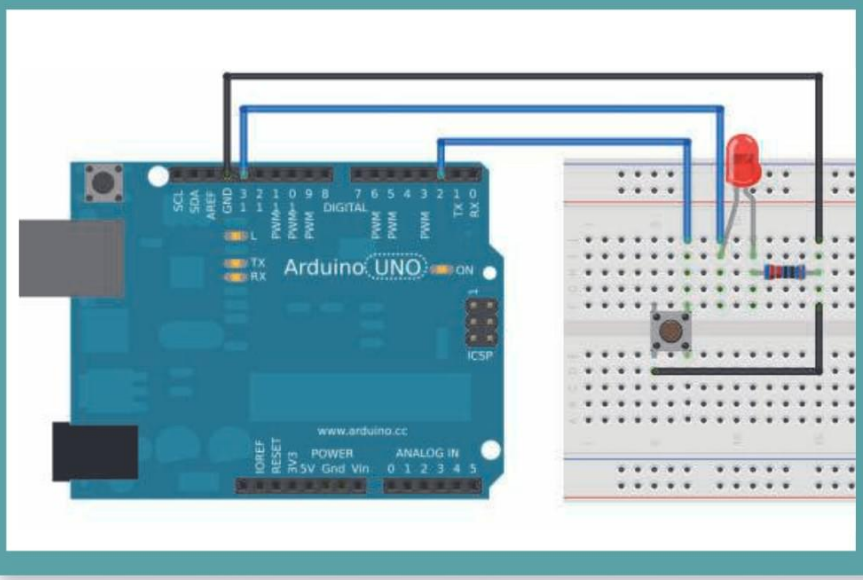
```
void secuencia3(){
for(int i=2; i<=9; i++){
digitalWrite(i,HIGH);
delay(50);
}
for(int i=9; i>=2;i--){
digitalWrite(i,LOW);
delay(50);
}
}
```

Finalmente, necesitamos una cuarta secuencia, podemos copiar el siguiente código de ejemplo:

```
void secuencia4(){
int k=11;
for(int i=2; i<=5;i++){
digitalWrite(i,HIGH);
digitalWrite(k-i,LOW);
}
delay(150);
for(int i=2; i<=5;i++){
digitalWrite(i,LOW);
digitalWrite(k-i,HIGH);
}
delay(150);
}
```

Una vez que creamos las secuencias de encendido, las probamos para ver su funcionamiento antes de implementar el circuito general. También es posible efectuar algunas modificaciones que nos entreguen variaciones en el encendido de los LEDs o rotarlos sin necesidad de que debamos usar el pulsador. Las posibles combinaciones son infinitas.

+ Diagrama de conexión 2



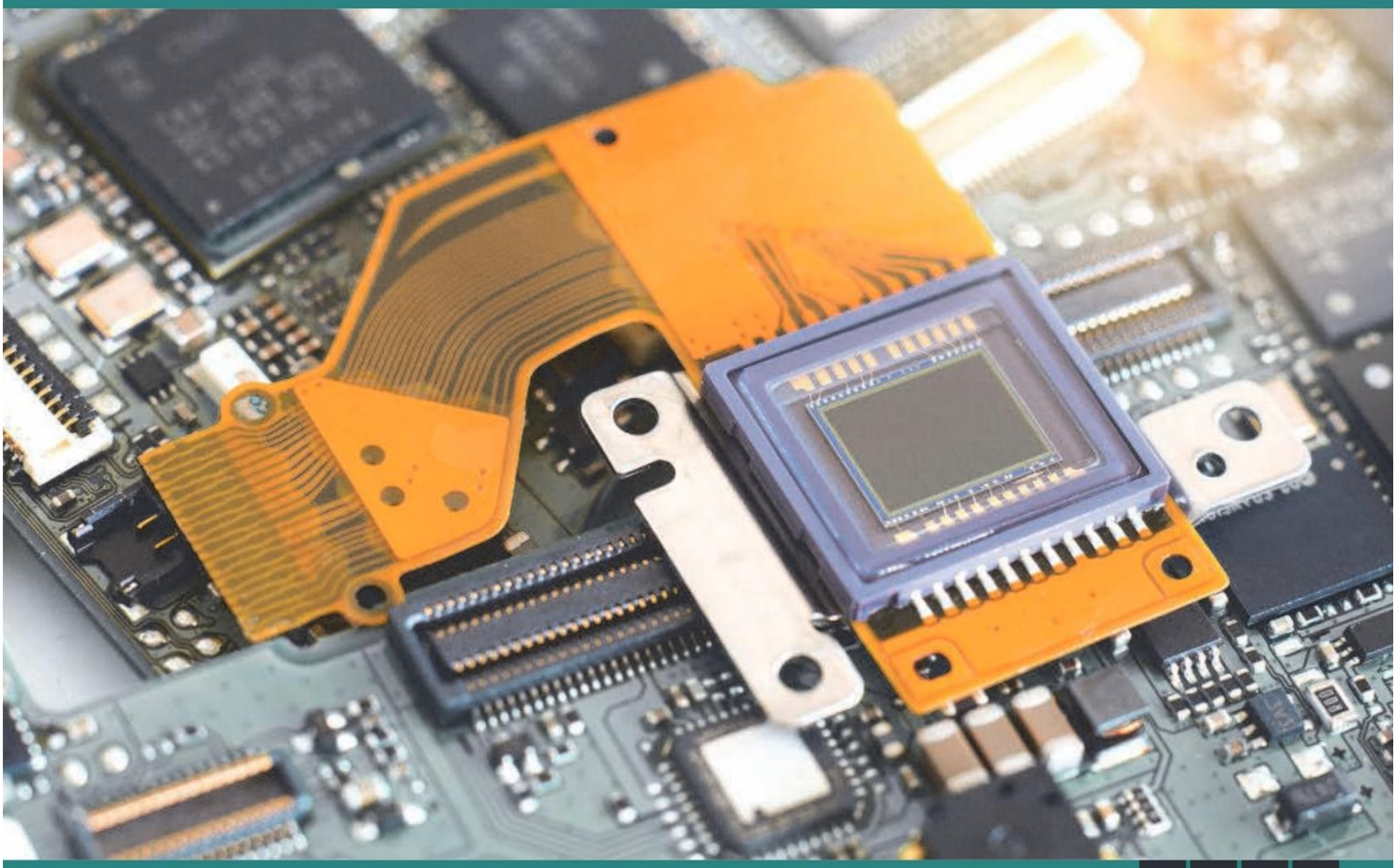
Lograremos diferentes variaciones teniendo como base la creación de distintas secuencias de encendido, a las que accederemos mediante un pulsador o en forma automática.

Resumen Capítulo 06

En este capítulo revisamos qué son los LEDs y para qué sirven. Analizamos sus principales características y aprendimos la forma en que podemos aprovecharlos en nuestros proyectos con Arduino. Trabajamos en diversos proyectos con LEDs, en los que aprendimos a conectarlos de diferentes formas, ya sea directamente o mediante un protoboard, generamos secuencias de encendido incorporando varios LEDs y creamos sketchs completos que subimos a la placa Arduino UNO para probar su funcionamiento.

07

Sensores

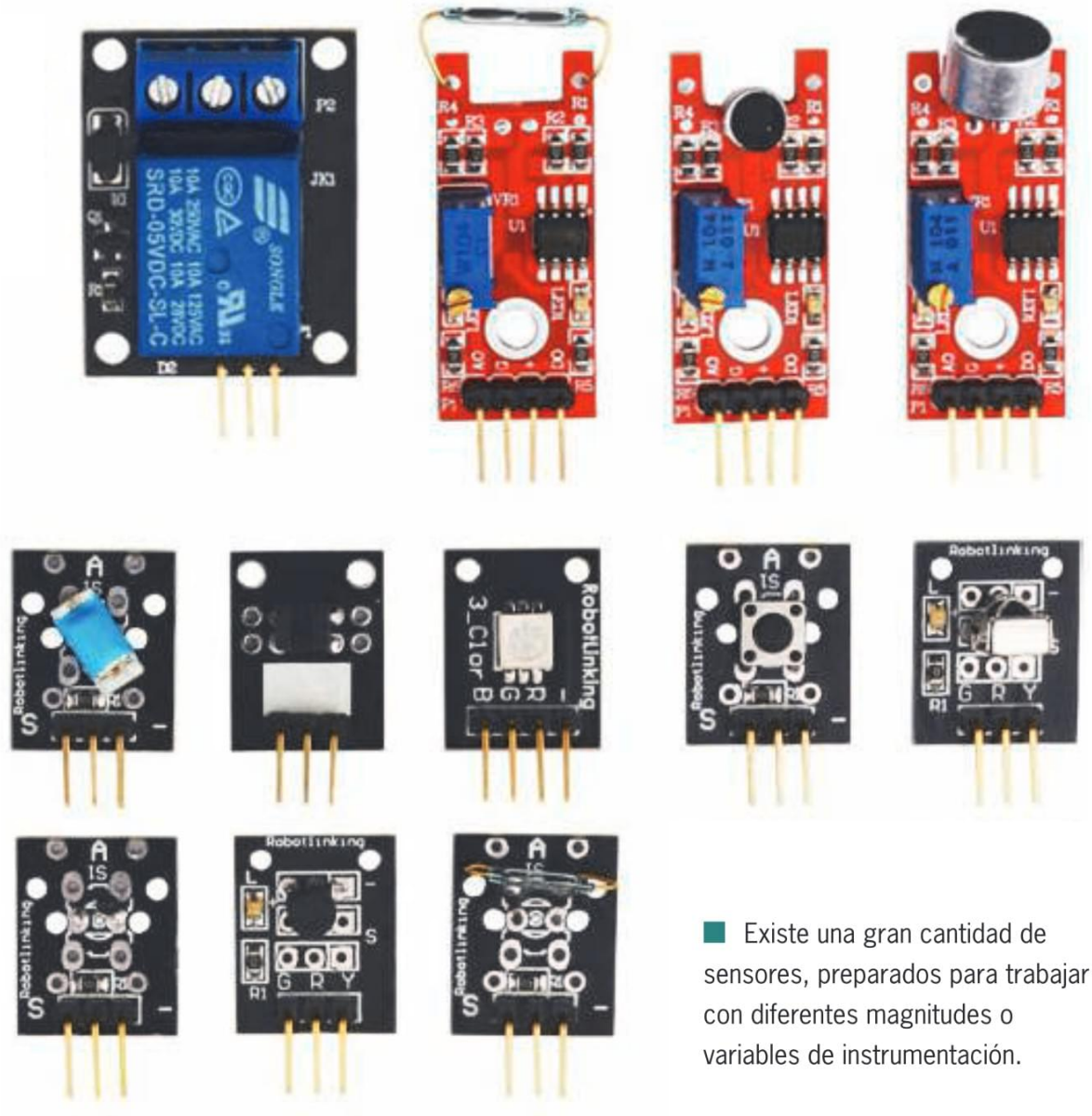


Ya pudimos trabajar en nuestros primeros proyectos básicos utilizando Arduino, para ello hicimos uso de una tarjeta Arduino UNO junto a algunos componentes sencillos.
En este capítulo conoceremos qué son los sensores y para qué sirven, además, veremos cómo pueden ayudarnos a complementar nuestros proyectos.

¿QUÉ ES UN SENSOR?

En pocas palabras, podemos definir un **sensor** como un dispositivo que tiene la capacidad de medir magnitudes físicas o químicas para convertirlas en magnitudes eléctricas, que pueden ser manejadas por una placa como Arduino.

Las magnitudes físicas o químicas que son detectadas por un sensor se denominan **variables de instrumentación**, estas son muy diversas, por ejemplo: temperatura, distancia, humedad, movimiento, presión, desplazamiento, ph, entre muchas otras.



Existe una gran cantidad de sensores, preparados para trabajar con diferentes magnitudes o variables de instrumentación.

El sensor tomará estas variables de instrumentación y las convertirá en magnitudes eléctricas, como resistencia eléctrica, capacidad eléctrica, tensión o corriente, etcétera.

Un sensor puede definirse mediante diferentes características, como las que mencionamos a continuación:

RANGO DE MEDIDA
Se trata del dominio en la magnitud medida donde puede aplicarse el sensor, es decir, corresponde al espacio en el que el sensor es capaz de detectar la magnitud o la variable de instrumentación para la que fue creado.

PRECISIÓN
Esta característica corresponde al error de medida máximo esperado cuando se utiliza un sensor específico para exponerlo a una variable de instrumentación.

OFFSET O DESVIACIÓN DE CERO
Es el valor de la variable de salida cuando el sensor se enfrenta a una variable de entrada nula o cero. Si nos enfrentamos a un rango de medida que no presenta un valor nulo para la variable de entrada, puede establecerse otro punto de referencia para lograr la definición de offset.

RESOLUCIÓN
Se trata de la mínima variación de la magnitud de entrada que puede ser detectada en la salida del sensor.

RAPIDEZ DE RESPUESTA
Esta característica se relaciona con la existencia de un tiempo fijo o la dependencia de cuánto varíe la magnitud que deseamos medir con el sensor.

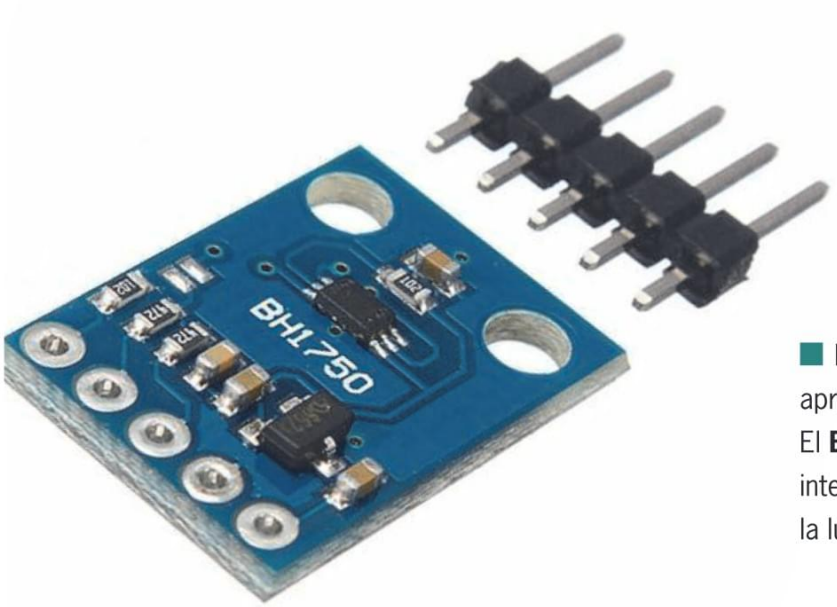
DERIVAS
Se trata de magnitudes que son capaces de influir en la variable de salida que es entregada por el sensor utilizado. Por ejemplo, condiciones ambientales, como humedad o temperatura, o desgaste del sensor.

REPETITIVIDAD
Se relaciona con los errores esperados cuando repetimos la misma medida en varias oportunidades.

Clasificación

Los sensores, dependiendo de los datos de salida que entregan, pueden clasificarse en dos grandes grupos: **digitales** y **analógicos**.

En primer lugar, debemos entender que los sensores digitales son aquellos capacitados para cambiar de estado de cero a uno o de uno a cero (HIGH, LOW) cuando se enfrentan a un estímulo que sean capaces de leer. No pueden ofrecer estados intermedios, y los valores de tensión correspondientes son solo dos: 5 V y 0 V, aunque también podrían presentarse valores muy cercanos.



En esta imagen podemos apreciar un sensor de luz. El **BH1750** es un sensor de intensidad que puede medir la luz ambiente.

En correspondencia, una señal eléctrica digital es interpretada por los microcontroladores como un valor binario, es decir, 0 - 1 o FALSE - TRUE o LOW - HIGH.

Al trabajar con Arduino el 0 o LOW se corresponde con el intervalo 0 V – 1 V y el 1 o HIGH corresponde al intervalo 2 V – 5 V. Es importante tener en cuenta que un valor entre esos intervalos es interpretado con un solo valor binario.

Hasta aquí tenemos todo claro, existen sensores que son capaces de entregarnos magnitudes que se ajustan a 0 y 1, pero la verdad es que, en nuestro contacto con el mundo exterior, rara vez nos encontraremos solo con la posibilidad de medir este tipo de magnitudes, por ejemplo, la temperatura no va solo de frío a caliente, o la humedad no va desde húmedo a no húmedo, es decir, existen magnitudes que nos ofrecerán lecturas de salida que se extienden en una línea donde existen diversos

valores consecutivos. Para nuestra referencia a la temperatura, podemos encontrar valores que van, por ejemplo, desde -5 grados hasta 35 grados, dependiendo del rango de medida del sensor que estemos utilizando.

Como ya hemos visto, los sensores digitales son capaces de transformar los datos que extrae del entorno en un valor que va desde 0 V a 5 V, catalogándolos como 0 y 1 o LOW y HIGH. Por otra parte, los sensores analógicos son capaces de diferenciar cualquier valor intermedio, por lo tanto, resultan adecuados para medir variables de instrumentación, como temperatura, ph o luz, entre otras magnitudes.

Aquí vemos un sensor que es capaz de detectar gas.

Una señal analógica es capaz de tomar cualquier valor en su voltaje, se trata de una señal que se puede representar mediante funciones matemáticas, por ejemplo, la señal eléctrica que utilizamos en casa, pues se trata de una señal que presenta una onda senoidal.

En la **Tabla 1**, conoceremos algunos ejemplos de sensores digitales y analógicos.

"En el trabajo con Arduino podemos incorporar sensores digitales y analógicos."

SENSORES DIGITALES Y ANALÓGICOS

MAGNITUD	ANALÓGICO	DIGITAL
Posición lineal y angular	Potenciómetro	Sensor Hall
Desplazamiento y deformación	Galga extensiométrica Magnetorresistivo	
Velocidad lineal y angular	Dínamo tacométrica	Detector inductivo
Aceleración	Acelerómetro	
Presión	Piezoeléctrico	Manómetro digital
Caudal	Turbina	
Temperatura	Termistor NTC	
Proximidad	Capacitivo Inductivo Fotoeléctrico	
De luz	Fotorresistencia	

En esta tabla vemos algunos ejemplos de sensores digitales y analógicos.

ENTRADAS EN ARDUINO

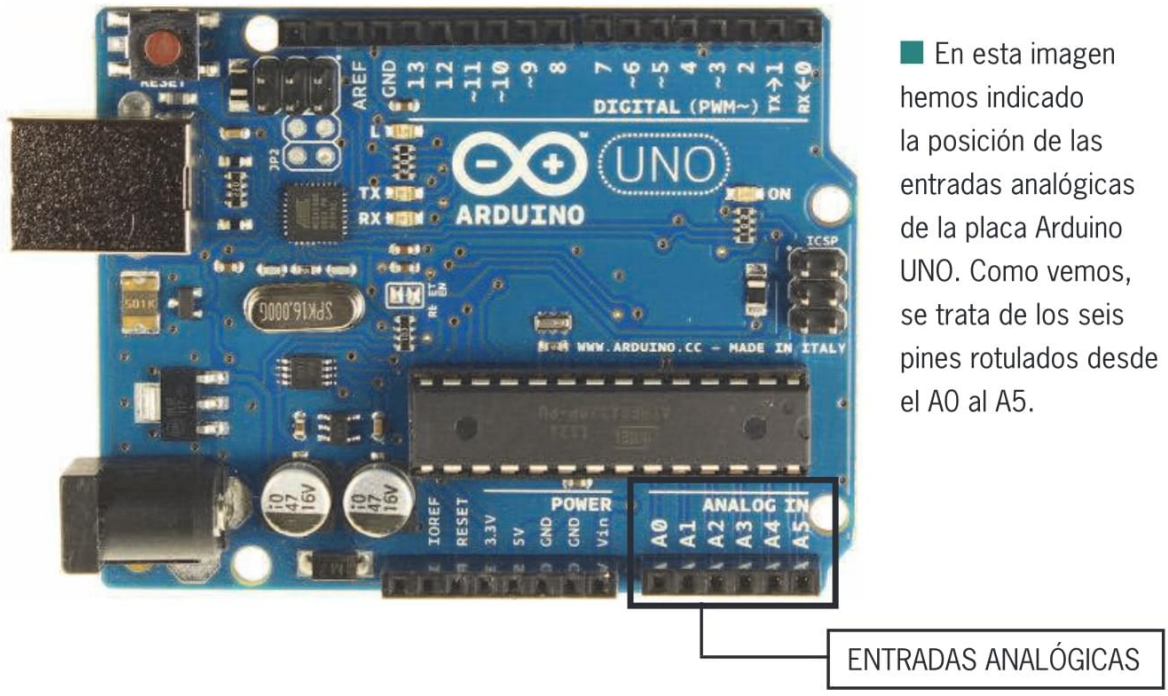
Como sabemos, la placa Arduino no solo es capaz de enviar señales, sino que además puede leerlas, para ello utilizará entradas tanto analógicas como digitales ubicadas en los extremos de las tarjetas. El principal objetivo de estas entradas es recibir datos de los sensores conectados a la placa y también comunicarse con los shields, que conoceremos en un capítulo posterior.

Para seguir la línea que llevamos en esta obra, presentaremos las entradas analógicas y digitales considerando la placa Arduino UNO, que es la que venimos utilizando en nuestros proyectos.

Entradas analógicas

En la placa Arduino UNO, las entradas analógicas corresponden a los pines A0 al A5, es decir Analógico 0, Analógico 1, Analógico 2, etcétera.

Estas entradas son capaces de recibir y leer valores de tensión que van desde 0 V a 5 V, con una resolución de 1024 o 10 bits.



En esta imagen hemos indicado la posición de las entradas analógicas de la placa Arduino UNO. Como vemos, se trata de los seis pines rotulados desde el A0 al A5.

La sintaxis básica de leer la información desde una de estas entradas es la siguiente:

```
leer = analogRead(pin);
```

Por supuesto, debemos reemplazar ciertas partes de este código dependiendo de lo que deseamos ejecutar. Por ejemplo, será necesario escribir el nombre de la variable donde almacenaremos la lectura en lugar de **leer**, y reemplazar **pin** por el número de pin que leeremos (este pin va desde el **0** hasta el **5**), por ejemplo:

```
valor1 = analogRead(2);
```

Este código almacenará la lectura que corresponde a la entrada analógica **2**, en la variable denominada **valor1**.

Leer un sensor de temperatura

Para ejemplificar el trabajo con las puertas analógicas, utilizaremos el sensor analógico de temperatura LM35, que conectaremos a la placa Arduino UNO, y leeremos los valores que nos presenta, utilizando el monitor serie del Arduino IDE.

El sensor que utilizaremos, el LM35, ofrece una precisión de 0.5 ºC mientras que su sensibilidad es de 10 mV/ºC, está calibrado para trabajar con grados Celcius y puede leer valores entre los -50 ºC y los 150 ºC.

Para ejemplificar su uso, realizaremos una pequeña aplicación que requiere de pocas conexiones, solo precisamos la placa Arduino, un protoboard, algunos cables de puente y el sensor LM35.

El montaje de los componentes es bastante fácil, debemos tener en cuenta que la pata +Vs debe conectarse al pin **5V**, la pata **Vout** se conectará al pin que utilicemos para realizar la lectura, y la pata restante deberá conectarse al pin **GND**.

Una vez que hemos realizado las conexiones adecuadas, trabajaremos en el código, para ello necesitaremos iniciar el Arduino IDE.

Para comenzar declaramos las variables adecuadas, una para almacenar la temperatura y otra para declarar el pin de entrada que usaremos:

```
float leer-temp;
intpinentrada = 0;
```

Como vemos, la variable **leer-temp** es de tipo **float**, porque necesitamos trabajar con valores decimales.

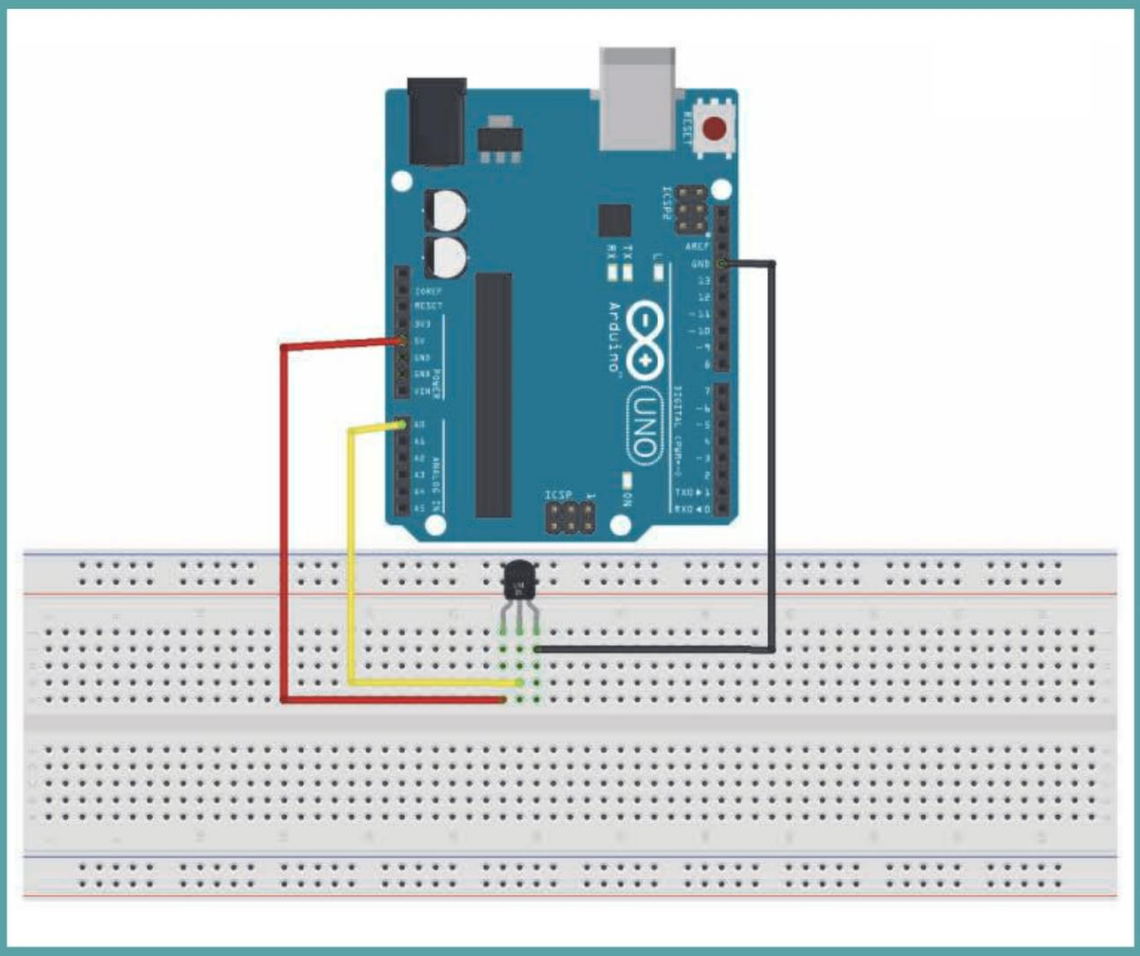
Para continuar, abrimos el puerto serial; como aprendimos en el **capítulo 5**, esto lo realizamos en **setup()**:

```
Serial.begin(9600);
```

Lo siguiente será escribir las líneas que nos permitan leer el pin adecuado, transformar el valor de lectura a grados y escribirlo por el puerto serial. Aunque parece una tarea bastante compleja, puede ser realizada en unas pocas líneas de código:

```
{
    leer-temp = analogRead(pinentrada);
```

+ Conexiones para el sensor LM35



En este esquema podemos verificar la manera correcta de efectuar las conexiones para, en forma posterior, leer los datos entregados por el sensor LM35.

Lectura analógica

Antes de leer la información desde una entrada analógica, debemos tener en cuenta algunos conceptos importantes. La lectura se encargará de entregarnos un valor que puede estar entre 0 y 1023. Este valor final será proporcional al nivel de la señal de entrada que se detecte en el pin. Por ejemplo, si tenemos una entrada nula, el valor que veremos es cero, si la entrada es de 2.5 V, el valor presentado será 511, por otra parte, para una lectura de 5 V, el valor que veremos será de 1023.

```
    leer-temp = (leer-temp / 1023 * 5 / 0.01);
Serial.print(leer-temp);
Serial.print(" grados \n");
delay(2000);
}
```

El código completo tendrá el siguiente aspecto:

```
float leer-temp;
intpinentrada = 0;

voidsetup()
{
Serial.begin(9600);
}

voidloop()
{
    leer-temp = analogRead(pinentrada);
    leer-temp = (leer-temp / 1023 * 5 / 0.01);
Serial.print(leer-temp);
Serial.print(" grados \n");
delay(2000);
}
```

Como ya tenemos experiencia en la carga de sketchs a la placa Arduino, lo único que nos queda es conectar la placa a la PC y efectuar los pasos necesarios para cargar el código.

Con la carga del sketch terminada y las conexiones correctamente realizadas, desplegamos el Monitor serial y veremos las lecturas que son realizadas a través del pin digital, estas corresponden a las mediciones efectuadas por el sensor de temperatura, con una espera de dos segundos entre cada medición. Si deseamos modificar esta espera, será necesario cambiar el valor contenido en la línea:

```
delay(2000);
```

En este esquema verificamos la manera correcta de efectuar las conexiones para, en forma posterior, leer los datos entregados por el sensor LM35.

Un código que puede servirnos de base para trabajar con cualquier sensor analógico es el que se describe a continuación, solo será necesario cambiar la relación voltaje/variable. Para ello, es preciso considerar tres pasos principales: ADC del voltaje analógico procedente del sensor, obtener el voltaje del sensor y, finalmente, obtener la variable del sensor. El código base, disponible en **www.digymakers.es**, es el siguiente:

```
floatadc;
float voltaje;
float variable;
floatrel_voltaje_variable = 100.00;

voidsetup()
{
Serial.begin(9600);
}

voidloop()
{

adc = analogRead(pin_sensor);
Serial.println(adc);

  voltaje = adc * 5 / 1023;
Serial.println(voltaje);
```

```
  variable = voltaje * rel_voltaje_variable;
Serial.println(variable);

delay(1000);
}
```

Entradas digitales

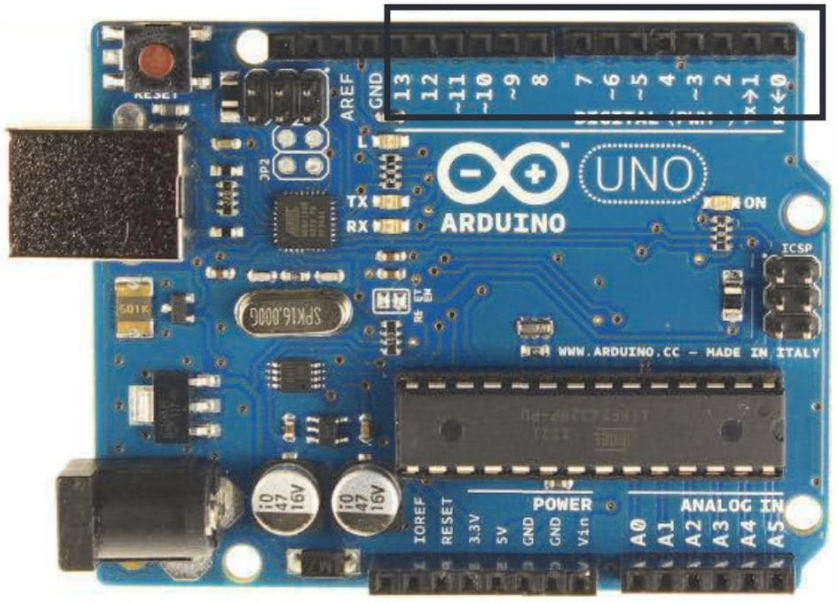
Las **entradas digitales** corresponden a los pines 1 al 13, es decir, los mismos que las salidas digitales. Se trata de entradas que, a diferencia de las analógicas, solo son capaces de comprender dos estados de señal de entrada: HIGH y LOW o lo que es igual 1 y 0, o valores cercanos a 5 V y 0 V.

Los pines digitales se encuentran configurados como entradas en forma predeterminada, pero también es posible hacerlo en forma manual y, para ello, utilizaremos el siguiente código:

```
pinMode(pinentrada,INPUT);
```

Por otra parte, si necesitamos almacenar en una variable los valores captados en una entrada digital, debemos usar el código:

```
variable2 = digitalRead(pinentrada);
```



Con las conexiones realizadas y el sketch cargado, utilizamos el monitor serial para ver las mediciones de temperatura informadas por el sensor LM35.

SENSORES PARA ARDUINO

Hasta este punto ya conocimos el funcionamiento y las características de los sensores; también, hemos apreciado la forma en que debemos conectarlos y acceder a sus lecturas desde Arduino.

Es importante considerar que existe una gran cantidad de sensores compatibles con Arduino disponibles en el mercado; por lo general, se trata de elementos que poseen un bajo costo por lo que integrarlos en nuestros proyectos no requerirá que desembolsemos una gran cantidad de recursos.

Para iniciarnos en el uso de los sensores y su conexión con Arduino, puede ser una buena idea adquirir un kit de sensores. Estos paquetes de elementos incorporan variadas opciones incluyendo sensores, pero también actuadores compatibles con Arduino.

No es el objetivo de este libro realizar un repaso completo por todos los sensores existentes en el mercado, tarea que, por lo demás, es casi imposible, pues cada día aparecen nuevos sensores o versiones actualizadas que poseen nuevas características. En este apartado, más bien buscamos presentar una pequeña selección de sensores para que el lector sea capaz de identificarlos y conocer su funcionamiento al tiempo que los utiliza en sus proyectos.

En esta imagen vemos uno de los kits de sensores más populares para Arduino. Se trata de un conjunto de 37 sensores y actuadores compatibles con Arduino.

Sensor de temperatura KY-001

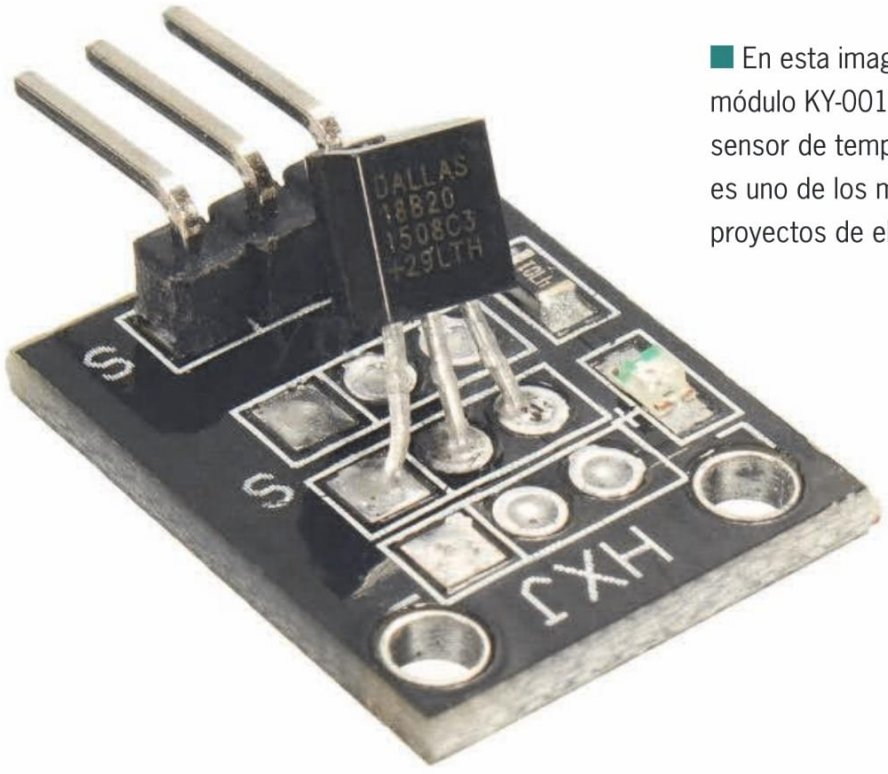
El módulo **KY-001** integra un sensor **DS18B20**, que permite medir la temperatura. Se trata de uno de los sensores más utilizados en proyectos de electrónica, por lo que su documentación es bastante amplia. Entre los usos más populares de este sensor, encontramos la posibilidad de medir la temperatura ambiente de una habitación, de un automóvil, del interior de una computadora, entre otras.

Sus principales características son las siguientes:

- **Rango de voltaje**: 3.0 V a 5.5 V
- **Rango de temperatura**: -55 °C a + 125 °C o 67 °F a 257 °F
- **Rango de precisión**: ±0.5 °C

Para utilizar este sensor, necesitaremos descargar la librería **OneWire**; luego la incluimos en el sketch y, en este caso, iniciamos en el pin 10, de la siguiente forma:

```
#include<OneWire.h>
OneWireds(10);
```

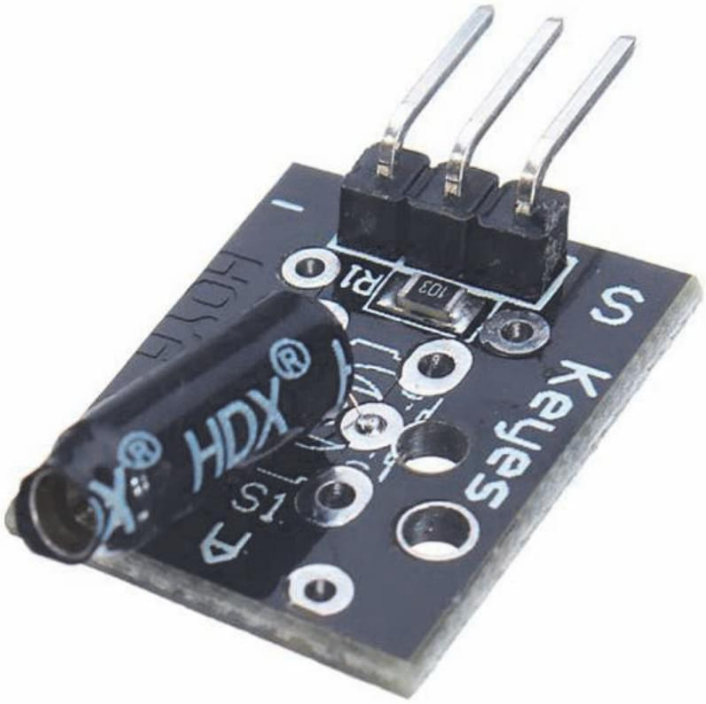


En esta imagen se presenta el módulo KY-001, que integra un sensor de temperatura DS18B20; es uno de los más utilizados en proyectos de electrónica.

Sensor de vibración KY-002

El funcionamiento básico de este módulo sensor de vibraciones es similar a un switch, pues es capaz de detectar una vibración y cierra un circuito. Si nos basamos en esto, podremos dotar a nuestro proyecto o placa Arduino de la capacidad de detectar movimientos.

Para utilizar este módulo, debemos conectarlo al pin 10 de la placa Arduino; al detectar una vibración, puede encenderse un LED que se encuentre conectado al pin 13.



Un sensor de vibración permite cerrar un circuito cuando se detecta una vibración, de esta forma se puede encender un LED para indicar la presencia de movimiento.

Un ejemplo del código que es posible usar con este sensor de vibración es el siguiente, se trata del código ofrecido para probar el funcionamiento del sensor **KY-002**:

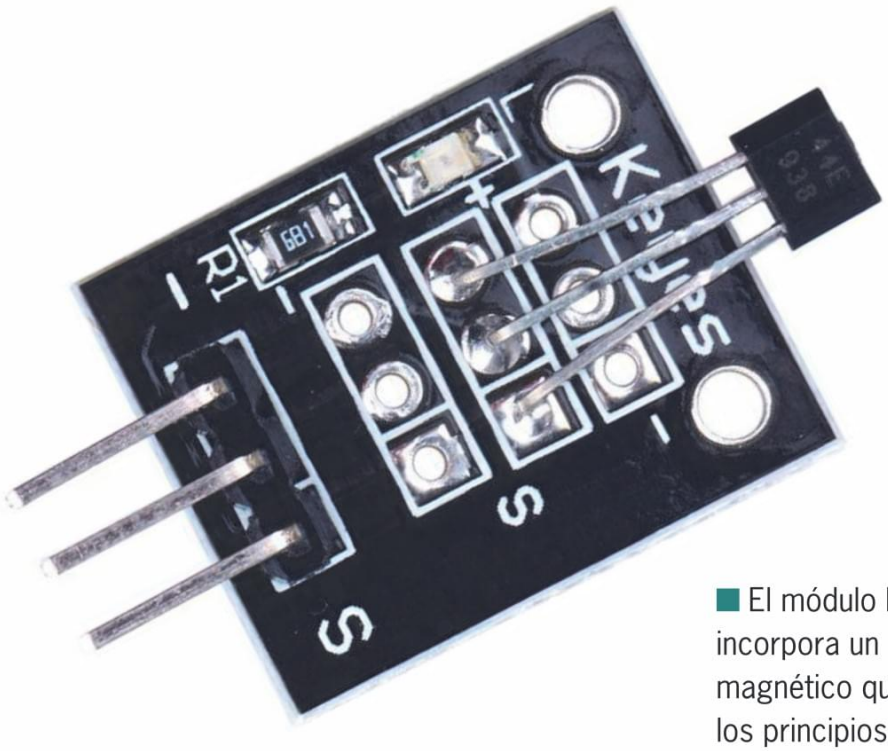
```
intLed = 13 ;
int Shock = 10;
int val;
voidsetup ()
{
pinMode (Led, OUTPUT) ;
pinMode (Shock, INPUT) ;
}
```

```
voidloop ()
{
val = digitalRead (Shock) ;
if (val == HIGH)
  {
digitalWrite (Led, LOW);
} else {
digitalWrite (Led, HIGH);
  }
}
```

Sensor de campo magnético KY-003

Este módulo incorpora un sensor de efecto Hall, que nos permite detectar la aparición de campos magnéticos. En presencia de un campo magnético se creará una señal en alto, que, dependiendo del código utilizado, puede encender un LED.

Los sensores de tipo Hall son muy utilizados tanto en la industria automotriz como en la medición de fluidos o en la detección de metales, entre otras aplicaciones.



■ El módulo **KY-003** incorpora un sensor de campo magnético que funciona según los principios del efecto Hall.

Su principal ventaja es que funcionan a distancia, aunque precisan estar cerca del campo magnético, pero sin contacto físico.

El principio de funcionamiento de este tipo de sensores se relaciona con la circulación de una corriente eléctrica en un semiconductor en presencia de un campo magnético; de esta forma, los electrones serán desviados gracias al campo magnético, ofreciendo una tensión que es perpendicular a la corriente y a dicho campo.

Al medir esta tensión que se origina por el efecto Hall, es posible trabajar con sensores y medidores de campo magnético. Un ejemplo de código que nos permite trabajar con este sensor de campo magnético es el que vemos a continuación:

"Los sensores de campo magnético no necesitan establecer contacto físico para detectar alteraciones magnéticas."

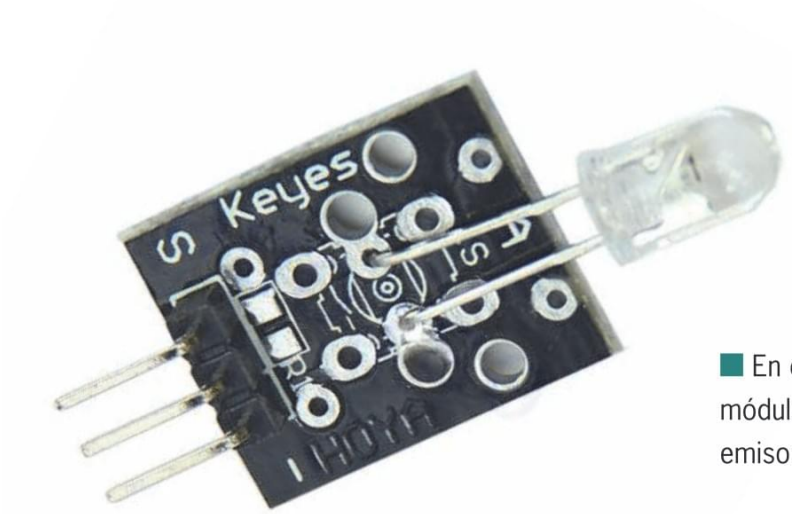
```
intLed = 13 ;
int SENSOR = 10 ;
int val ;

voidsetup ()
{
pinMode (Led, OUTPUT) ;
pinMode (SENSOR, INPUT) ;
}

voidloop ()
{
val = digitalRead (SENSOR) ;
if (val == HIGH)
  {
digitalWrite (Led, HIGH);
} {
digitalWrite (Led, LOW);
  }
}
```

Sensor emisor infrarrojo KY-005

En nuestro entorno, estamos en contacto con sensores infrarrojos en diversas ocasiones, por ejemplo, en tareas tan sencillas como manipular televisores o reproductores de música.

El módulo **KY-005** incorpora un sensor emisor infrarrojo y posee las siguientes características:

- **CORRIENTE**: 30 a 60 mA
- **CORRIENTE DE PULSO**: 0.3 a 1 A
- **VOLTAJE INVERSO**: 5 V
- **POTENCIA DE DISIPACIÓN**: 90 mW
- **RANGO DE TEMPERATURA**: -25 a + 80 °C
- **TEMPERATURA DE ALMACENAMIENTO**: -40 a +100 °C



En esta imagen podemos ver el módulo KY-005, se trata de un sensor emisor de infrarrojo.

Un ejemplo de código que puede utilizarse con el módulo KY-005 es el siguiente:

```
# Include<IRremote.h>
int RECV_PIN = 11;
IRrecvirrecv (RECV_PIN);
decode_resultsresults;
voidsetup ()
{
Serial.begin (9600);
irrecv.enableIRIn ();
```

```
}
voidloop () {
if (irrecv.decode (&results)) {
Serial.println (results.value, HEX);
irrecv.resume ();
}
}
```

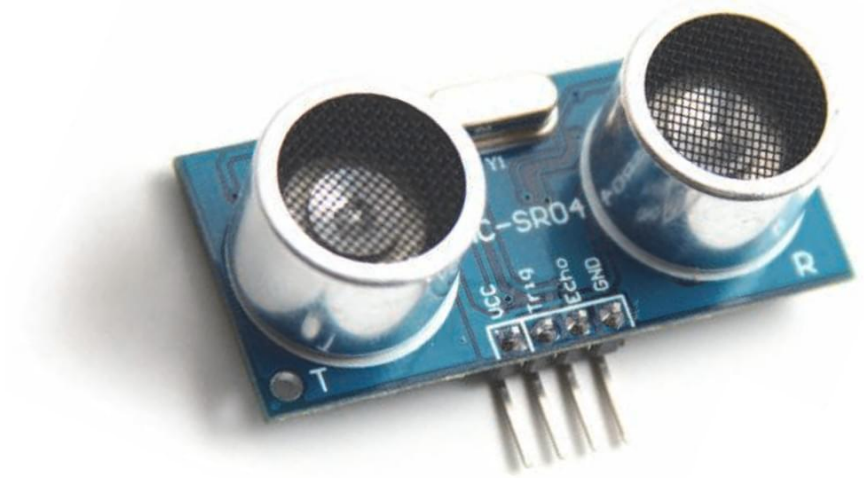
Sensor ultrasónico HC-SR04

A diferencia de lo que podemos pensar, este tipo de sensor es adecuado para medir distancias y, también, esquivar obstáculos. Se trata de un sensor bastante versátil, cuyas principales aplicaciones se relacionan con la robótica.

Su funcionamiento se basa en la emisión de ultrasonidos, que no pueden ser detectados por el oído humano. Esto se realiza a través de uno de los dos cilindros que se presentan en el sensor: el sonido enviado rebotará en los objetos y será captado por el otro cilindro que compone al sensor.

Uno de los usos más comunes de este sensor es la medición de distancias, por lo que debemos considerar su rango de operación, pues ofrece una sensibilidad que va entre los 2 y los 400 cm, con una precisión de 3 mm.

La información que nos entrega este sensor es el tiempo que transcurre entre la emisión del ultrasonido y su recepción, por la misma razón, debemos proceder a traducir esto en una distancia.



El sensor ultrasónico HC-SR04 se compone de dos cilindros, uno de los cuales emitirá un ultrasonido que rebotará en los objetos y será captado a través del otro cilindro.

Para lograrlo, tengamos en cuenta que la velocidad del ultrasonido en el aire presenta un valor de 340 m/s, o lo que es igual 0,034 cm/microseg. De esta forma, para proceder a calcular la distancia, debemos considerar que v=d/t (velocidad es igual a la distancia dividida en tiempo). Por lo tanto d=vt.

Teniendo esto en cuenta, podemos probar el funcionamiento del sensor ultrasónico realizando las conexiones adecuadas. Este sensor presenta cuatro pines, que deben ser conectados de la siguiente forma:

- VCC debe conectarse a la salida de 5 V de Arduino.
- Trig debe conectarse al pin digital de la placa que utilizaremos para enviar el pulso ultrasónico.
- Echo debe conectarse al pin digital que recibirá el eco del pulso enviado.
- GND debe conectarse a tierra.

Finalmente, veamos un ejemplo del código que utilizaremos con el sensor ultrasónico HC-SR04:

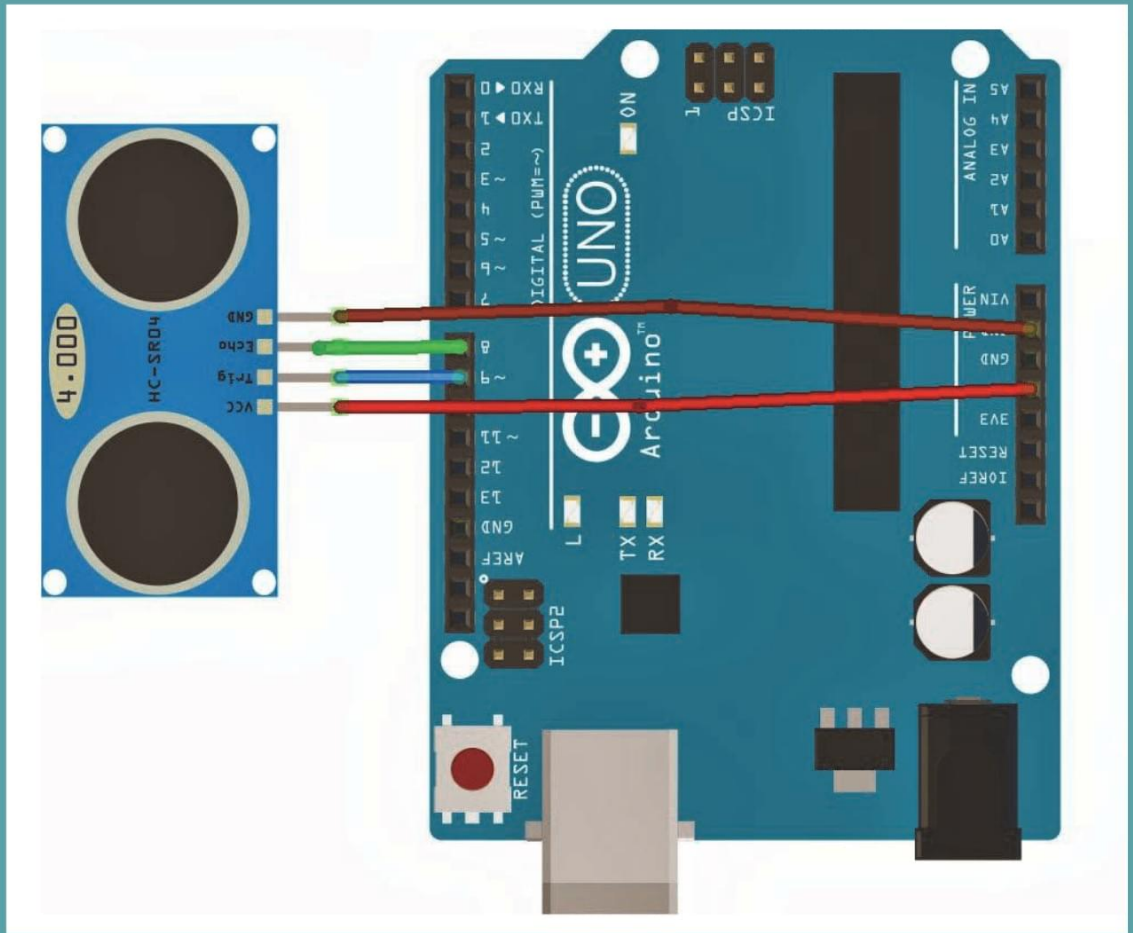
```
long distancia;
long tiempo;
voidsetup(){
Serial.begin(9600);
pinMode(9, OUTPUT);
pinMode(8, INPUT);
}

voidloop(){
digitalWrite(9,LOW);
delayMicroseconds(5);
```

Sensor de humo

Se trata de un sensor bastante utilizado en sistemas de detección de incendios. En esencia, este sensor es capaz de detectar humo y gas. Se puede calibrar mediante el uso de un potenciómetro, y la señal de salida que entrega puede ser conectada a una entrada analógica de una placa Arduino.

```
digitalWrite(9, HIGH);
delayMicroseconds(10);
  tiempo=pulseIn(8, HIGH);
  distancia= int(0.017*tiempo);
Serial.println("Distancia ");
Serial.println(distancia);
Serial.println(" cm");
delay(1000);
}
```

+ Conexión entre sensor ultrasónico y placa Arduino



En esta imagen vemos una representación de la conexión que debemos realizar entre el sensor ultrasónico y la placa Arduino.

Sensores LDR

El **sensor LDR** o **fotorresistor** es un tipo de sensor capaz de variar su resistencia en función de una magnitud física. En realidad se trata de un componente cuya resistencia varía con la cantidad de luz que percibe. Es utilizado para medir la iluminación en dispositivos electrónicos: si existe más luz, menor será la resistencia eléctrica; si existe menos luz, la resistencia eléctrica será mayor.

Un código que podemos usar como ejemplo para trabajar con un sensor LDR es el siguiente:

```
intLed1 = 2;
intLed2 = 3;
intLed3 = 4;
intpinLDR = 0;

intvalorLDR = 0;

voidsetup()
{
pinMode(Led1, OUTPUT);
pinMode(Led2, OUTPUT);
pinMode(Led3, OUTPUT);

Serial.begin(9600);
}

voidloop()
{
digitalWrite(Led1, LOW);
digitalWrite(Led2, LOW);
digitalWrite(Led3, LOW);

valorLDR= analogRead(pinLDR);

Serial.println(valorLDR);

if(valorLDR> 256)
```

```
  {
digitalWrite(Led1, HIGH);
  }
if(valorLDR> 512)
  {
digitalWrite(Led2, HIGH);
  }
if(valorLDR> 768)
  {
digitalWrite(Led3, HIGH);
  }
delay(200);
}
```

Resumen Capítulo 07

En este capítulo conocimos en detalle qué es un sensor y para qué sirve. Analizamos las características de este tipo de componentes y los clasificamos en dos grandes grupos: los sensores analógicos y los sensores digitales. Conocimos las entradas analógicas y digitales en Arduino y practicamos la forma en que podemos leer un sensor de temperatura conectado a la placa Arduino UNO. Después analizamos en detalle algunos de los sensores disponibles en el mercado, y revisamos ejemplos de su uso y del sketch que podemos utilizar en nuestros primeros proyectos.

Detección de luz

08

En el capítulo anterior conocimos qué son los sensores y cómo podemos aprovecharlos en nuestros proyectos con Arduino. En esta ocasión, trabajaremos en detalle con un sensor LDR, para lograr proyectos que sean capaces de efectuar la detección del nivel o la intensidad de la luz.

FOTORRESISTENCIA

Ya tenemos los conocimientos básicos que nos permitirán enfrentar un proyecto algo más complejo. Sabemos crear nuestros primeros sketchs y también subirlos a una placa Arduino, aprendimos a compilar los programas y a detectar las posibles fallas que pudieran presentarse. Con todo esto sobre la mesa, además de la información que hemos recabado sobre los diversos componentes que pueden ser utilizados en proyectos electrónicos, podemos realizar un proyecto como el que presentamos a continuación.

En el **capítulo 7** de este libro conocimos los sensores, vimos qué son y para qué podemos utilizarlos, aprendimos que existe una gran gama de sensores, pero analizamos las características esenciales de algunos de ellos, por ejemplo, de los sensores LDR.

Este tipo específico de sensor, denominado **sensor LDR** o **fotorresistencia** es capaz de variar su resistencia en función de la luz que recibe, por esta razón lo utilizaremos en un proyecto que nos permita medir el nivel de luz existente en un momento determinado.

Las fotorresistencias pueden encontrarse en diversos tamaños, pero todas cumplen con una función esencial, modificar su resistencia en función de la luz a la que son expuestas.

Sabemos que una fotorresistencia no es más que un sensor de luz de tipo resistivo, por lo tanto, es capaz de cambiar su valor según la cantidad de luz que incide sobre ella. Teniendo esto en cuenta, la conectaremos a Arduino para leer la cantidad de luz que existe en el ambiente ya sea interior como exterior.

Funcionamiento

Para entender el funcionamiento de una fotorresistencia, es necesario conocer la relación matemática que existe entre la iluminancia y la resistencia de un sensor LDR, esto se expresa en una función matemática:

$$\frac{I}{I_0} = \left(\frac{R}{R_0}\right)^{-gamma}$$

Al analizar esta fórmula, saltan a la vista algunos elementos importantes, y es necesario definirlos: **R0** corresponde a la resistencia a una intensidad **I0**, ambos valores son conocidos.

Por otra parte, la **constante gamma** corresponde a la pendiente de la gráfica logarítmica o la pérdida de resistencia por década. Su valor típico es de **0.5** a **0.8**.

Si queremos obtener valores más precisos para la fotorresistencia que deseamos utilizar, es una buena idea consultar el datasheet que corresponde al componente, por ejemplo, para la familia de fotorresistores GL55 corresponden los datos que mostramos aquí.

DATOS DE LOS FOTORRESISTORES GL55

MODELO	GL5516	GL5528	GL5537-1	GL5537-2	GL5539	GL5549
Voltaje (V)	150	150	150	150	150	150
Temperatura (°C)	-30° + 70°	-30° + 70°	-30° + 70°	-30° + 70°	-30° + 70°	-30° + 70°
Pico espectral (nm)	540	540	540	540	540	540
Resistencia oscuridad (KΩ)	5-10	10-20	20-30	30-50	50-100	100-200
Resistencia luz brillante (KΩ)	500	1000	2000	3000	5000	10000
Gamma	0.5	0.6	0.6	0.7	0.8	0.9
Tiempo respuesta (ms)	30	25	25	25	25	25

Datos correspondientes a las fotorresistencias de la familia GL55.

En todo caso, debido a las variaciones propias del proceso de construcción de estos componentes, no podemos utilizar esta información en forma absoluta. La recomendación es efectuar un proceso de calibración preliminar, de esta forma, enfrentaremos la posibilidad de que pequeñas diferencias entre componentes den como resultado grandes variaciones en el momento de efectuar la medición.

Otros componentes necesarios

La fotorresistencia se convierte en el componente principal para la realización de este proyecto. Utilizaremos una fotorresistencia de la familia GL55, pues estas se encuentran entre las más comunes y accesibles y, además, necesitaremos contar con otros componentes.

Ya sabemos que la resistencia LDR, en este caso de la familia GL55, será el componente principal, pero vamos a definir otros elementos que también son necesarios.

Como siempre, contaremos con una placa Arduino. Podemos elegir entre muchas opciones disponibles en el mercado, pero, siguiendo el trabajo que hemos realizado en este libro, utilizaremos la Arduino UNO. Ya hemos visto que se trata de una placa bastante potente y versátil, por eso no tendremos problemas a la hora de implementar las conexiones y los códigos necesarios.

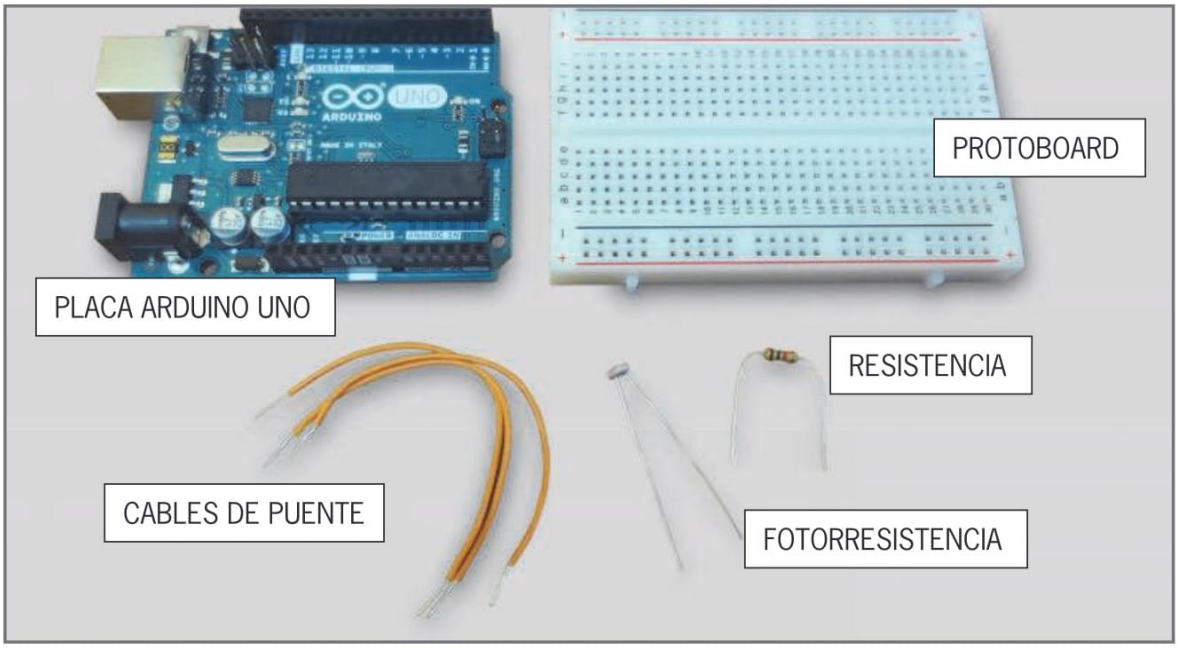
No es posible efectuar la conexión de la placa Arduino a la fotorresistencia en forma directa, o por lo menos no es lo recomendado, por esta razón también utilizaremos un protoboard. Este elemento nos permitirá conectar en forma precisa todos los componentes involucrados

Diferencias de construcción

Existen muchas fotorresistencias diferentes; sus variaciones no solo dependen del fabricante, sino también del material utilizado en su construcción. Las resistencias LDR solo reducen su resistencia si existe una radiación luminosa situada en una determinada banda de longitudes de onda. Por ejemplo, las construidas utilizando **sulfuro de cadmio** serán sensibles a todas las radiaciones luminosas visibles, mientras que las que han sido construidas con **sulfuro de plomo** solo serán sensibles a las radiaciones infrarrojas.

en el proyecto y trabajar con mayor comodidad. Además, debemos contar con una resistencia y varios cables de puente.

Con estos elementos básicos en nuestro poder, estaremos listos para iniciar la implementación del proyecto.



Componentes que necesitamos para desarrollar este proyecto.

EL PROYECTO

Para entender lo que perseguimos con la ejecución de este proyecto, debemos profundizar aún más en las características del componente principal: el fotorresistor.

Un fotorresistor o *Light Dependent Resistor* es un dispositivo capaz de variar su resistencia en función de la luz que recibe, por esta razón, lo utilizaremos en conjunto con las entradas analógicas de la placa Arduino, para obtener una estimación del nivel de luz existente.

La constitución de un fotorresistor se basa en la presencia de un semiconductor, por lo general de sulfuro de cadmio (CdS). De esta forma, cuando la luz incide sobre él, algunos de los fotones que lo constituyen serán absorbidos; debido a esto los electrones pasan a la banda de conducción, lo que finalmente disminuirá la resistencia que presenta el componente o fotorresistor.

De esta forma, el fotorresistor es capaz de disminuir su resistencia a medida que aumenta la luz que se proyecta sobre él. Sus valores típicos serán de 1 Mohm, cuando enfrentamos el resistor a la oscuridad total, y de 50-100 Ohms, cuando se enfrenta a la luz brillante.

+ Símbolos electrónicos

Los materiales fotosensibles más utilizados para la fabricación de las resistencias LDR son el sulfuro de talio, el sulfuro de cadmio, el sulfuro de plomo y el seleniuro de cadmio. En esta imagen vemos sus símbolos electrónicos.

Debemos tener en cuenta que la variación de la resistencia en un fotorresistor es bastante lenta; por esta razón, no es un elemento eficiente cuando deseamos medir variaciones de luz rápidas, por ejemplo, las que podría producir una fuente de luz artificial que se encuentra alimentada por corriente alterna. La variación de la resistencia irá de 20 a 100 ms según el modelo de fotorresistencia que utilicemos.

Aunque esto puede parecer una enorme desventaja, dependiendo del proyecto en que estemos trabajando será más bien una característica deseable, pues la fotorresistencia es un sensor que nos ofrece una gran estabilidad. Pero siempre debemos considerar que no es posible aplicarlo a cualquier tipo de proyecto.

Por ejemplo, si deseamos medir la iluminancia, un fotorresistor no será adecuado, debido a su limitada precisión en variaciones de

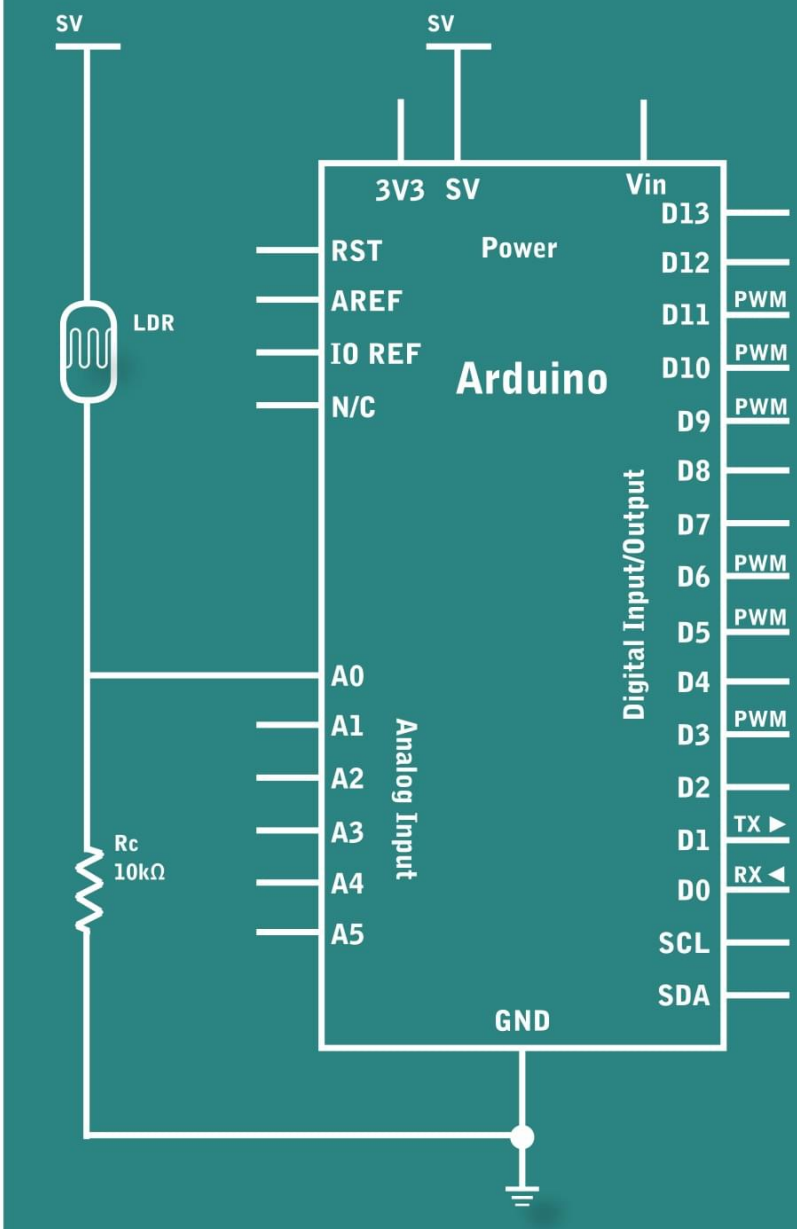
Aplicaciones de las fotorresistencias

Aunque en esta ocasión utilizaremos una fotorresistencia para crear un proyecto bastante sencillo, su aplicación puede apoyarnos en diversos proyectos avanzados, por ejemplo, en iluminación, apagado y encendido de alumbrado mediante interruptores crepusculares, en la implementación de alarmas, para manipular cámaras fotográficas y también en el funcionamiento de medidores de luz. Los fotorresistores que pertenecen a la clasificación **infrarroja** pueden ser utilizados tanto en el control de máquinas como en procesos de detección de objetos.

luz rápida, pero, en cambio, si deseamos medir la intensidad de la luz ambiental tanto en interiores como en exteriores, seguro estamos frente al sensor adecuado.

Teniendo en cuenta los puntos que mencionamos hasta este momento, utilizaremos este sensor LDR para averiguar medidas cuantitativas sobre el nivel de luz en interiores y en exteriores.

+ Conexiones para el sensor LM35



En esta imagen podemos ver el esquema que corresponde a la conexión de una fotorresistencia con la placa Arduino UNO. Se trata del esquema eléctrico que usaremos para este proyecto.

"Debemos conectar el sensor LM35 a una entrada analógica de Arduino."

Para lograrlo conectaremos este sensor a una entrada analógica de la placa Arduino UNO mediante el uso de un protoboard. Luego, podremos jugar con el código para, por ejemplo, hacer reaccionar un LED cuando el nivel de luz exceda un límite que hayamos establecido con anterioridad o para acceder a información sobre la lectura del nivel de iluminación que es detectada por el sensor LDR.

Resultados esperados

Considerando lo que hemos aprendido sobre el funcionamiento y la aplicación de los LEDs y también sobre el uso del Monitor serie del Arduino IDE, perseguiremos cuatro resultados básicos en el trabajo que llevaremos a cabo en este proyecto.

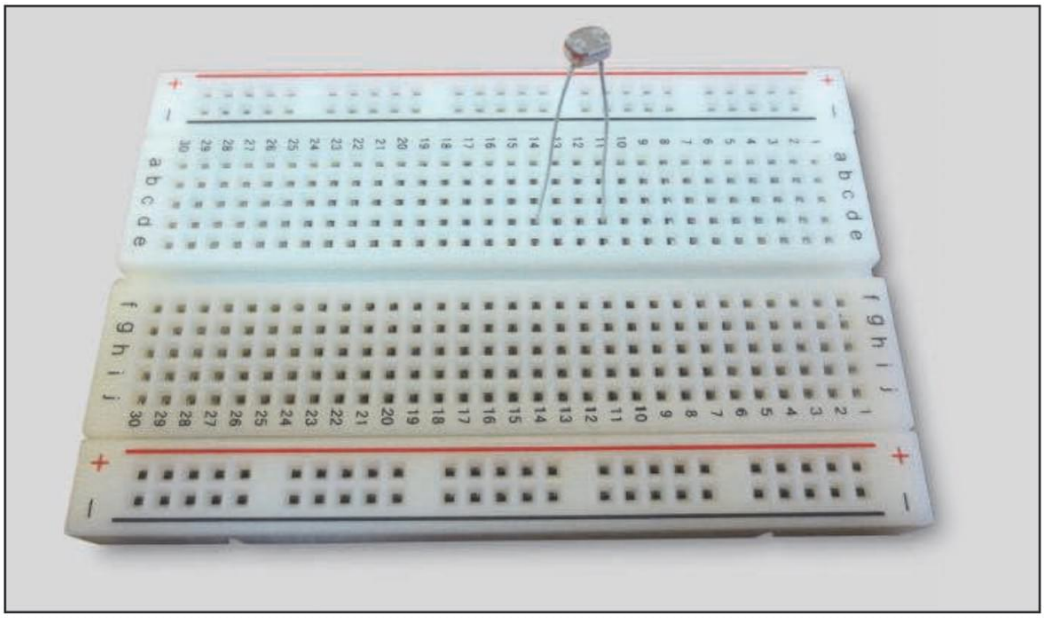
- **1.** Crear un sketch que nos permita activar un LED solo cuando la intensidad de la luz llegue a un nivel que definiremos con anticipación.
- **2.** Implementar el código necesario para hacer parpadear un LED mientras el sensor LDR conectado reciba la cantidad suficiente de luz.
- **3.** Implementar el sketch adecuado para leer los valores que corresponden al nivel de iluminación detectado por la fotorresistencia conectada a nuestro circuito.
- **4.** Encender algunos LEDs conectados al protoboard, dependiendo de la intensidad de luz que sea detectado por el sensor LDR.

CONECTAR EL CIRCUITO

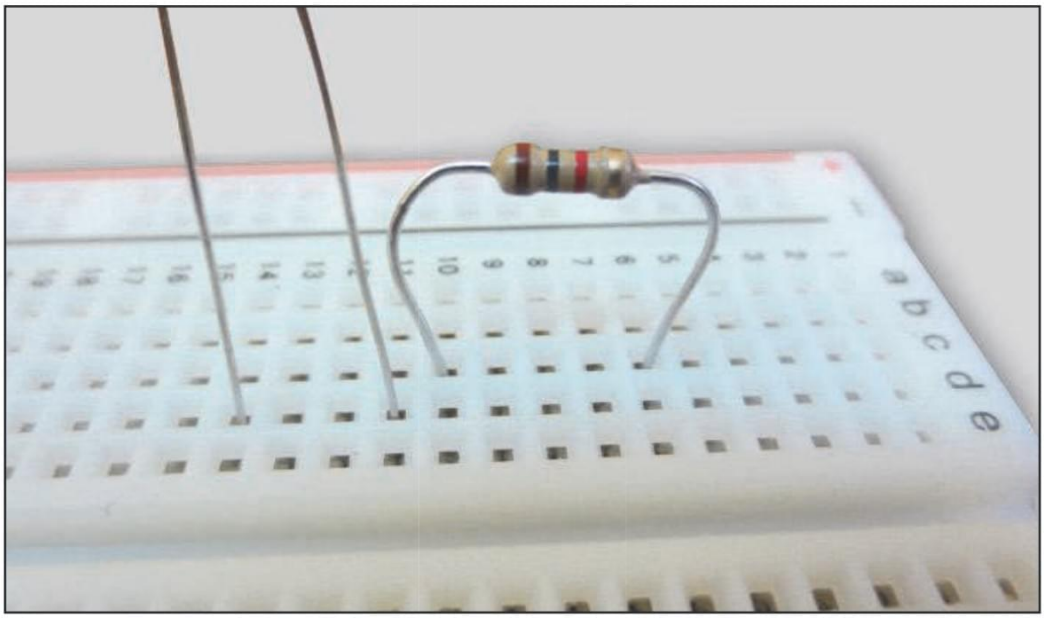
Ahora que ya conocemos el proyecto que deseamos llevar a cabo y también los resultados que esperamos, es momento de efectuar la conexión del circuito. En primer lugar reunimos los elementos necesarios, es decir, la placa Arduino UNO, la fotorresistencia, una resistencia y algunos cables de puente. Con todos los elementos dispuestos para trabajar, solo debemos seguir las instrucciones que presentamos en el siguiente **Paso a paso**.

Paso a paso: Conectar el circuito

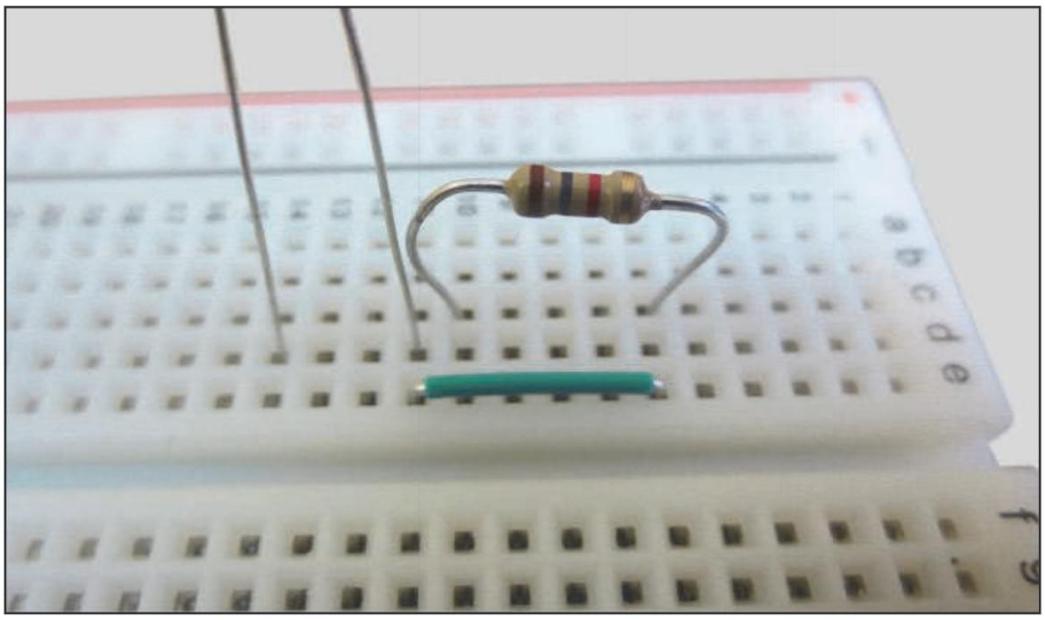
01 Para comenzar, conecte la fotorresistencia a la placa Arduino UNO, debe hacerlo en un espacio central, que le permita trabajar con comodidad con los demás elementos que es necesario incluir en el circuito.



02 Agregue una resistencia al circuito, conéctela justo en la línea siguiente a la fotorresistencia, esto es importante pues después las unirá mediante un cable de puente.

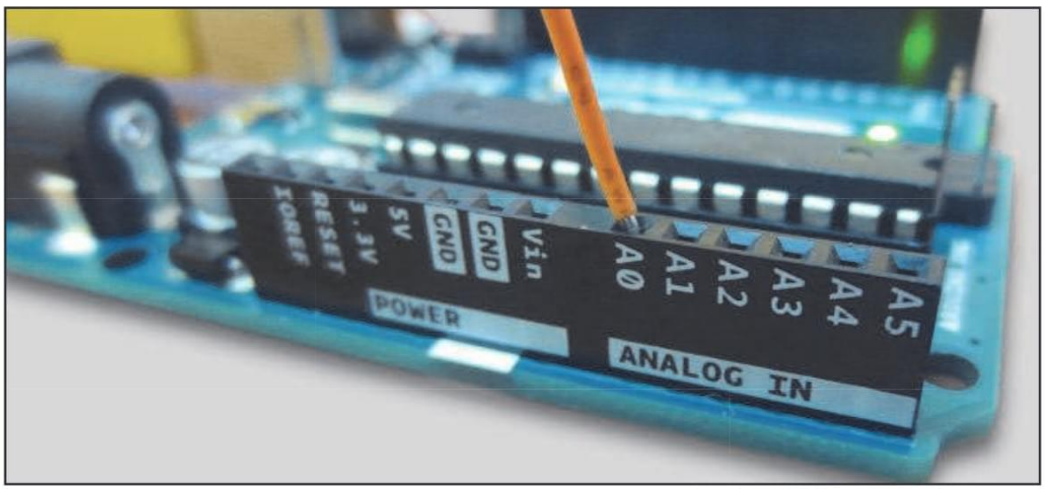


03 Ahora una la resistencia y la fotorresistencia, para ello use un cable de puente que una el extremo derecho de un elemento con el extremo derecho del otro elemento, tal como se ve en la imagen de ejemplo.



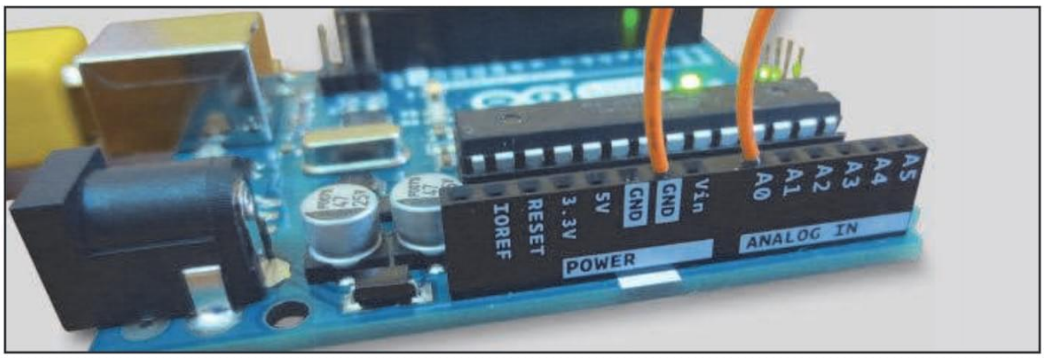
04 En este punto tiene las conexiones necesarias en el protoboard listas, debe conectarlo a la placa Arduino, pero eso será en un paso posterior. En este momento, prepare los tres cables de puente que usará para conectar Arduino con el protoboard.

05 En primer lugar, conecte el pin A0 de la placa Arduino UNO con un extremo de la resistencia. Para esto, debe conectar el cable de puente en la placa y luego en la línea que corresponde a uno de los extremos de la resistencia.



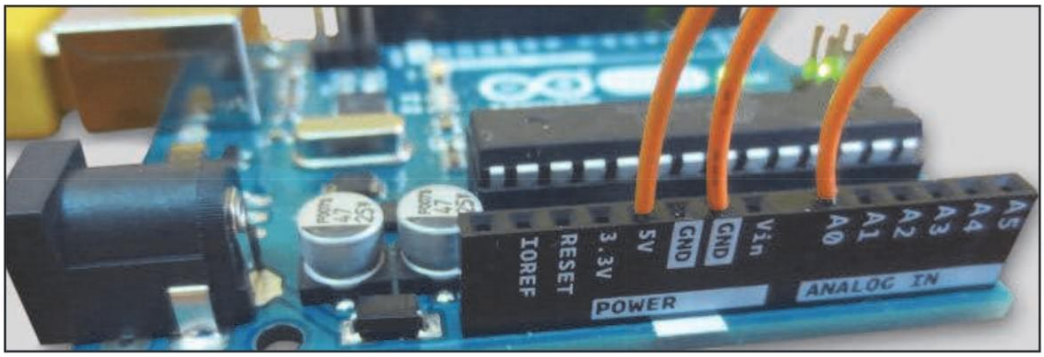
06

Conecte uno de los extremos del segundo cable de puente al pin GND de la placa Arduino UNO, el otro extremo del cable de puente deberá conectarse a la línea que corresponde al otro extremo de la resistencia, tal como se ve en la imagen de ejemplo.



07

El tercer cable de puente proporcionará el voltaje necesario a su circuito, para ello conecte uno de sus extremos al pin 5V de la placa Arduino UNO, el otro extremo debe ser conectado al extremo que queda libre en la fotorresistencia. Para conectarlo, utilice uno de los puntos de conexión que corresponde a la línea donde se encuentra conectada la fotorresistencia.



08

En este punto ya tiene su circuito creado, como ve, solo ha utilizado unos pocos componentes. Para terminar será necesario conectar la placa Arduino UNO a la computadora. Como siempre, tendrá que utilizar el cable USB adecuado. Efectúe la conexión con la PC y verifique que la placa se encuentre energizada, para ello observe los LEDs incorporados en Arduino UNO.

Ya tenemos el circuito creado, se trata de la disposición básica que nos servirá para probar diversos sketchs, que nos permitirán trabajar con la fotorresistencia. Es decir, trabajaremos teniendo en cuenta la cantidad de luz que recibe este sensor.

CREACIÓN DE LOS SKETCHS

Como veremos a continuación, será necesario crear un sketch básico que nos permita leer la información proporcionada por la fotorresistencia; partiendo de esto será posible efectuar algunos cambios para alterar el comportamiento de un LED conectado a la placa Arduino. Podremos, por ejemplo, encender un LED cuando la intensidad de la luz alcance cierto punto o hacer parpadear el mismo LED dependiendo de la luz que llega a la fotorresistencia. También es posible utilizar el Monitor serie para leer los valores que corresponden a la lectura de intensidad de luz que realiza la fotorresistencia y que leeremos a través del pin A0.

Para lograr todo esto, primero debemos crear un sketch básico y, posteriormente, iremos incorporando algunas modificaciones para lograr que nuestro proyecto sea cada vez más completo.

Hemos dividido el proyecto en cuatro resultados esperados, en cada uno de ellos intentaremos resolver uno de los planteamientos que efectuamos en una sección anterior, relacionados con este proyecto.

Resultado 1

Para concretar este resultado, antes recordemos el primer planteamiento:

- **1. Crear un sketch que nos permita activar un LED solo cuando la intensidad de la luz llegue a un nivel que definiremos con anticipación.**

Para efectuar esta tarea, iniciaremos el Arduino IDE y crearemos un sketch que contenga las siguientes porciones de código.

En primer lugar, definimos tres constantes, se tratará de los valores que corresponden al LED integrado en la placa Arduino UNO, el pin que utilizaremos para leer la información entregada por la fotorresistencia

y el valor que utilizaremos como punto de corte para encender o apagar el LED, para este ejemplo utilizaremos un valor de corte de **200**:

```
constint LED = 13;
constint LDR = A0;
constint valor = 200;
```

Luego escribiremos el apartado **setup()**, para este sketch utilizaremos las líneas de código que nos permitan establecer los pines que usaremos. Marcamos **LED** como salida y **LDR** como entrada, de la siguiente forma:

```
voidsetup() {
pinMode(LED, OUTPUT);
pinMode(LDR, INPUT);
}
```

En el apartado **loop()** utilizaremos el código que nos permita comparar la lectura obtenida por **LDR** con el número almacenado en la constante **valor**, recordemos que para este ejemplo se trata de **200**. Si es mayor que **valor**, encenderemos **LED**, de lo contrario lo apagaremos. El código que necesitamos es el siguiente:

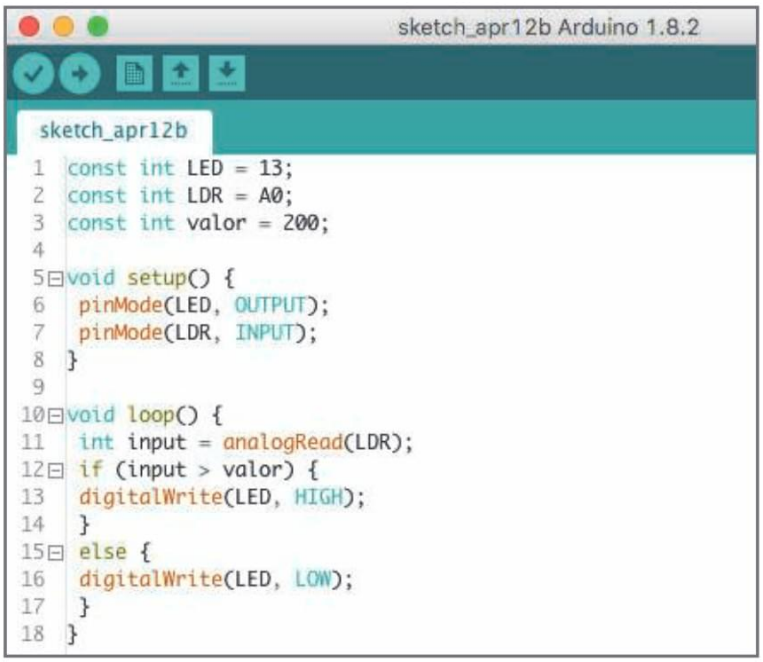
```
voidloop() {
int input = analogRead(LDR);
if (input >valor) {
digitalWrite(LED, HIGH);
 }
else {
digitalWrite(LEDPin, LOW);
 }
}
```

La apariencia del código completo es la siguiente, notemos que hemos definimos las contantes fuera de **setup()** y **loop()**.

```
constint LED = 13;
constint LDR = A0;
```

```
constint valor = 200;

voidsetup() {
pinMode(LED, OUTPUT);
pinMode(LDR, INPUT);
}

voidloop() {
int input = analogRead(LDR);
if (input > valor) {
digitalWrite(LED, HIGH);
 }
else {
digitalWrite(LEDPin, LOW);
 }
}
```

Una vez que hayamos ingresado el código completo en el Arduino IDE, debemos compilarlo, para ello solo es necesario hacer clic sobre la opción **Verificar/Compilar**. Si no encontramos ningún error, podremos cargarlo a la placa Arduino UNO; en caso de que se presente algún error, será necesario solucionarlo antes de continuar.



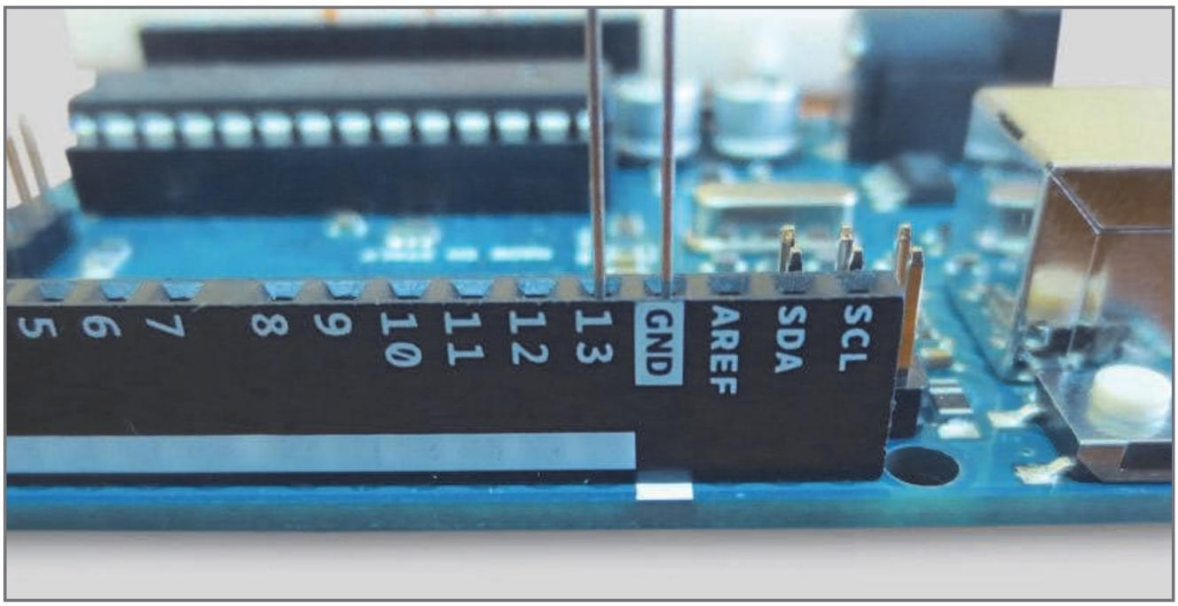
Como vemos en esta imagen, no solo hemos escrito el código que necesitamos para encender o apagar el LED dependiendo del valor que nos informe la fotorresistencia, también lo hemos compilado, y no encontramos errores.

Una vez que tengamos el sketch compilado, lo cargamos en la placa Arduino, mediante un clic en la opción **Subir**, que se encuentra en la barra superior de herramientas dentro del IDE. Cuando la carga del sketch haya finalizado, podremos verificar que el LED **L** de la placa se enciende en presencia de un nivel adecuado de luz, por ejemplo, si ponemos la mano sobre la fotorresistencia (con lo que recibirá menos luz), el LED **L** se apagará.

Podemos efectuar algunas modificaciones al sketch para probar nuevos resultados; es posible modificar la constante **valor** y, de esta forma, cambiaremos el valor de intensidad que utilizaremos para encender o apagar el LED **L**.

```
constint LED = 13;
constint LDR = A0;
constint valor = 150;
```

Otra modificación interesante que hará posible ver en forma más clara un LED encendido es conectar un LED directamente al pin 13 de la placa Arduino. Como sabemos, este pin incorpora una resistencia, por lo tanto, podemos utilizarlo para conectar un LED directamente.



Si conectamos un LED directamente al pin 13 de la placa Arduino UNO, podremos ver con mayor claridad cuándo se enciende o se apaga, dependiendo del nivel de luz que reciba la fotorresistencia.

Resultado 2

Para continuar con el proyecto que nos enseña a utilizar una fotorresistencia, recordemos el segundo planteamiento que nos impusimos:

- **2. Implementar el código necesario para hacer parpadear un LED mientras el sensor LDR conectado reciba la cantidad suficiente de luz.**

Para esto podemos basarnos en el código que utilizamos en la sección anterior y realizarle algunas modificaciones.

En primer lugar, necesitamos contar con las constantes adecuadas para este proyecto. Igual que en el caso anterior, usaremos el pin 13 para el LED y el pin A0 para leer la fotorresistencia, pero estableceremos un valor de corte de **150**:

```
constint LED = 13;
constint LDR = A0;
constint valor = 150;
```

En el apartado **setup()**, necesitamos las líneas de código que nos permitan indicar los pines necesarios y sus funciones. Utilizaremos **LED** como salida y **LDR** como entrada, tal como hicimos en el resultado anterior:

```
voidsetup() {
pinMode(LED, OUTPUT);
pinMode(LDR, INPUT);
}
```

En el apartado **loop()** efectuaremos algunos cambios. Si bien compararemos la lectura obtenida por **LDR** con el número almacenado en la constante **valor**, que para este ejemplo se trata de **150**, el cambio estará en que esta vez no solo encenderemos y apagaremos el LED adecuado, sino también lo haremos parpadear. El código que necesitamos es el siguiente:

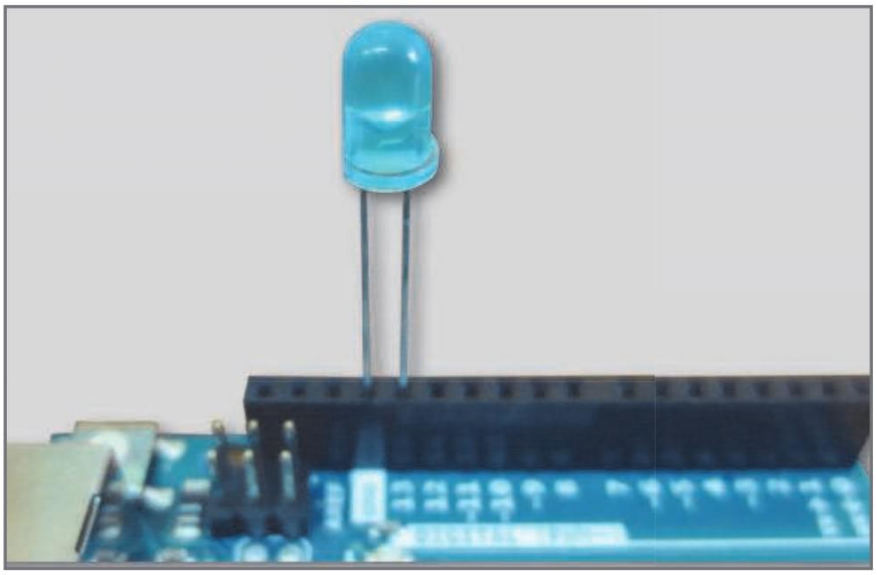
```
voidloop() {
int input = analogRead(LDR);
if (input > valor) {
```

```
digitalWrite(LED, HIGH);
delay(50);
digitalWrite(LED, LOW);
delay(50);
 }
else {
digitalWrite(LEDPin, LOW);
 }
}
```

El sketch completo es el siguiente:

```
constint LED = 13;
constint LDR = A0;
constint valor = 200;

voidsetup() {
pinMode(LED, OUTPUT);
pinMode(LDR, INPUT);
}

voidloop() {
int input = analogRead(LDR);
if (input > valor) {
digitalWrite(LED, HIGH);
delay(50);
digitalWrite(LED, LOW);
delay(50);
 }
else {
digitalWrite(LED, LOW);
 }
}
```

Una vez que hayamos compilado y cargado el sketch en la placa Arduino UNO, podremos verificar que el LED integrado parpadea mientras la fotorresistencia obtenga un nivel mayor de luz al indicado en la constante **valor**. Si cubrimos la fotorresistencia con la mano, el LED se apagará.



Al igual que sucede con el resultado obtenido en la sección anterior, el LED parpadeará mientras el nivel de luz recibida por la fotorresistencia sea mayor al establecido en **variable**, que en este caso se trata de **150**.

En este sketch podemos tocar algunos valores para obtener diferentes comportamientos. En primer lugar es posible cambiar la constante almacenada en **valor**, de esta forma modificaremos el umbral adecuado para que el LED parpadee, por ejemplo, para elevarlo podríamos establecerlo en **250**:

```
constint LED = 13;
constint LDR = A0;
constint valor = 250;
```

Otra modificación que es posible realizar se encuentra dentro del delay que sigue cada vez que el LED se apaga y enciende, así podemos lograr que este parpadeo se haga más rápido o más lento, por ejemplo:

```
if (input > valor) {
digitalWrite(LED, HIGH);
delay(150);
digitalWrite(LED, LOW);
delay(150);
 }
```

Resultado 3

La tercera actividad que nos propusimos lograr en este proyecto:

- **3. Implementar el sketch adecuado para leer los valores que corresponden al nivel de iluminación detectados por la fotorresistencia conectada a nuestro circuito.**

Como vemos, aquí daremos un paso más grande, pues intentaremos leer los valores recogidos por la fotorresistencia, relacionados con la intensidad de la luz que recibe, y mostrarlos utilizando el monitor serie. Para lograrlo, utilizaremos el siguiente sketch, que presentamos y explicamos por secciones.

Primero debemos establecer las constantes que utilizaremos en este sketch. Necesitamos cuatro constantes: resistencia en la oscuridad, resistencia a la luz, resistencia de calibración, y el pin que usaremos para leer los datos de la fotorresistencia (recordemos que según nuestro circuito se trata del pin A0):

```
constlong A = 1000;
constint B = 15;
constintResCal = 10;
constint LDR = A0;
```

También necesitaremos dos variables que nos permitirán efectuar algunos cálculos que veremos más adelante:

```
int Va;
intil;
```

Es el momento de trabajar en **setup()**, en esta ocasión necesitamos iniciar el monitor serie, pues lo utilizaremos para mostrar los valores que obtendremos desde la fotorresistencia:

```
voidsetup()
{
Serial.begin(115200);
}
```

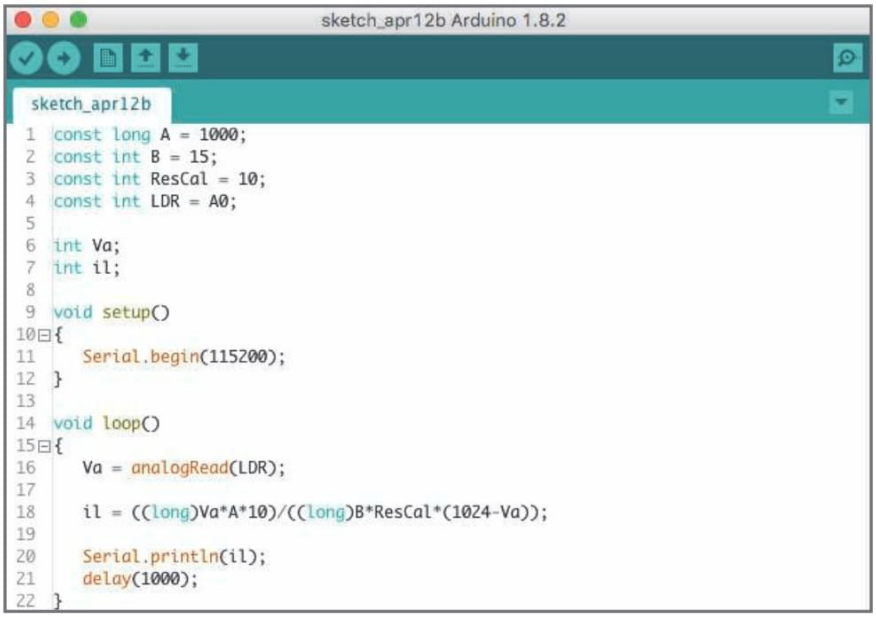
Para este sketch necesitamos incluir el siguiente código en **loop()**, se trata de los cálculos que precisamos para luego mostrar los datos adecuados utilizando el Monitor serie:

```
voidloop()
{
   Va = analogRead(LDR);

il = ((long)Va*A*10)/((long)B*ResCal*(1024-Va));

Serial.println(il);
delay(1000);
}
```

El sketch completo quedaría de la siguiente forma:

```
constlong A = 1000;
constint B = 15;
constintResCal = 10;
constint LDR = A0;

int Va;
intil;

voidsetup()
{
Serial.begin(115200);
}

voidloop()
{
   Va = analogRead(LDR);

il = ((long)Va*A*10)/((long)B*ResCal*(1024-Va));

Serial.println(il);
delay(1000);
}
```

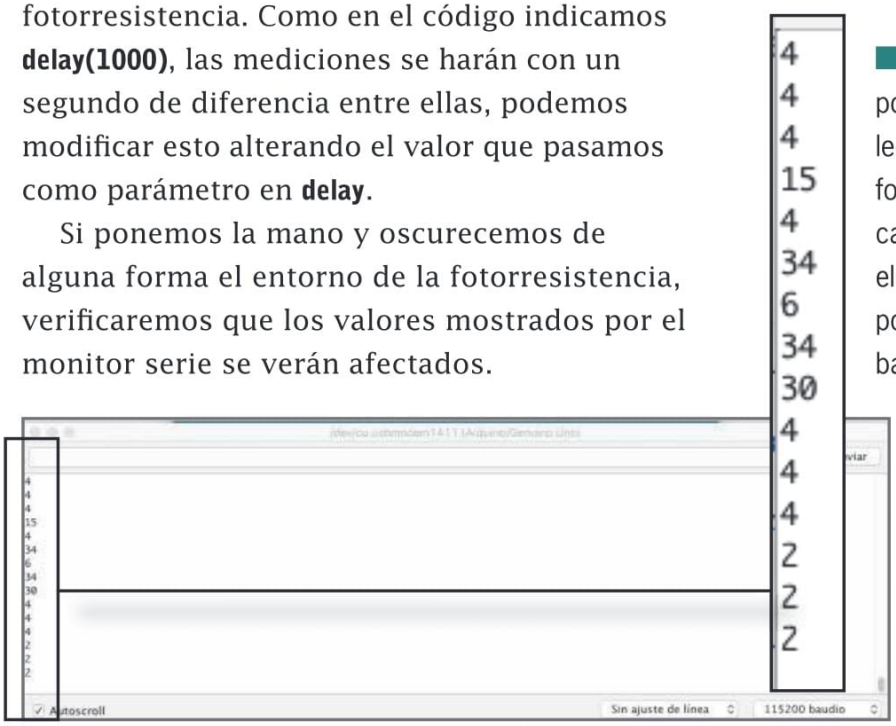
Luego de compilar y verificar que el sketch no contenga errores, lo subimos a la placa Arduino UNO.



■ En esta imagen podemos apreciar que nuestro sketch se ha subido a la placa Arduino sin problemas, por lo tanto, asumimos que no existían errores de codificación.

Si ejecutamos el Monitor serie, veremos que se escriben las lecturas entregadas por la fotorresistencia. Como en el código indicamos **delay(1000)**, las mediciones se harán con un segundo de diferencia entre ellas, podemos modificar esto alterando el valor que pasamos como parámetro en **delay**.

Si ponemos la mano y oscurecemos de alguna forma el entorno de la fotorresistencia, verificaremos que los valores mostrados por el monitor serie se verán afectados.



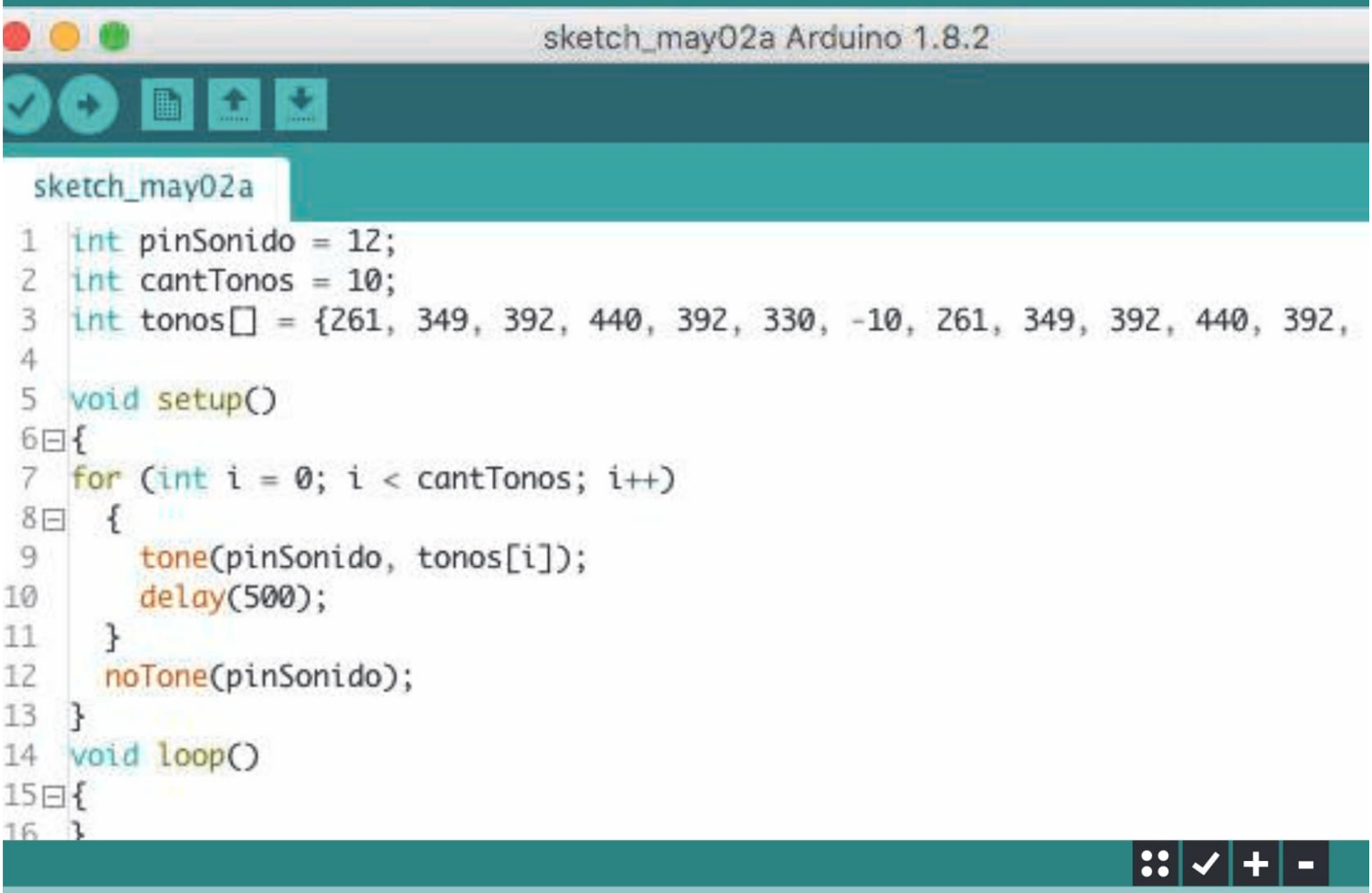
■ En el monitor serie es posible ver cada una de las lecturas informadas por la fotorresistencia. En este caso hemos oscurecido el ambiente para obtener, por pantalla, valores más bajos.

Resumen Capítulo 08

En este capítulo hemos trabajado con un sensor conocido como *fotorresistencia*. En primer lugar conocimos su funcionamiento y revisamos sus principales características. Posteriormente vimos la forma en que podemos aprovecharlo en un circuito y obtener mediciones desde Arduino. Creamos un proyecto en el que buscamos obtener diferentes resultados, partiendo desde un sketch básico que modificamos en diferentes etapas. Logramos que un LED se encienda dependiendo del nivel de luz que llegue a la fotorresistencia, también hicimos parpadear el LED dependiendo del nivel de luz existente y, por último, pudimos acceder a las lecturas relacionadas con el nivel de luz detectada por la fotorresistencia, utilizando el monitor serie.

Emisión de sonidos

09



En este capítulo agregaremos un componente más a nuestros proyectos con Arduino: la emisión de sonidos. Sin duda esto abre nuevas oportunidades y nos permite generar novedosas propuestas electrónicas.

ELEMENTOS NECESARIOS

Para llevar a cabo este proyecto, necesitamos contar con algunos elementos que hemos presentado y utilizado en capítulos anteriores: una tarjeta Arduino UNO, un protoboard y también cables de puente. Pero además agregaremos un buzzer o zumbador.

El **buzzer** o **zumbador** no es más que un transductor electroacústico capaz de producir un sonido o un zumbido continuo o intermitente de un mismo tono. Podemos utilizarlo como mecanismo de señalización o aviso y lo encontramos aplicado en automóviles, electrodomésticos o despertadores.

Este elemento también se conoce como **buzzer piezoeléctrico** o **piezo speaker**, y su principal característica es su capacidad de transformar la electricidad en sonido.

Los buzzers son componentes electrónicos muy accesibles, económicos y fáciles de adquirir.

Si nos detenemos en la construcción del buzzer, encontramos que se trata de elementos electrónicos que se forman mediante la combinación de dos discos de distintos materiales. Uno de estos discos es metálico y el otro puede ser de cerámica; como ambos poseen propiedades piezoeléctricas, al encontrarse frente a un voltaje los discos se repelen produciendo un sonido. Si la diferencia de tensión se pone a cero, los discos vuelven a su posición original y se produce un nuevo sonido.

Cuando el circuito se controla mediante un circuito oscilante externo, podemos hablar de un *transductor piezoeléctrico*. Por otra parte, si el circuito oscilador se incluye en el componente, podemos hablar de un *zumbador piezoeléctrico*.

En el mercado también se encuentran buzzers integrados en módulos.

Es posible encontrar los buzzer aplicados en el diseño de alarmas y controles acústicos, que requieran un rango de frecuencia estrecho, por ejemplo, en aparatos domésticos y de medicina.

Función tone

Aunque no se trata de un elemento físico, es necesario mencionar una función de Arduino que usaremos para completar este proyecto, se trata de **tone**.

Esta función nos permite crear sonidos en forma sencilla, solo seleccionando el pin de salida y la frecuencia. Su sintaxis es la siguiente:

```
tone(pinsalida,frecuencia);
```

Es necesario considerar que la función **tone** trabaja intercambiando los valores **HIGH/LOW** a una frecuencia específica en el pin de salida que indiquemos. Realizará esto hasta que indiquemos otra frecuencia o hasta que ordenemos que se detenga, para ello debemos usar la función **noTone**, de la siguiente forma:

```
noTone(pinsalida);
```

Mediante esta función lograremos un único tono al mismo tiempo; para lograr un tono distinto, primero debemos detener el anterior e invocar un nuevo tono con otra frecuencia. A continuación, vemos un ejemplo sencillo del uso de estas funciones:

```
constintpinSonido = 9;

voidsetup()
{
}

voidloop()
{
tone(pinSonido, 440);
delay(1000);

noTone(pinSonido);
delay(500);

tone(pinSonido, 523, 300);
delay(500);
}
```

EL PROYECTO

En la sección anterior conocimos en detalle qué es un buzzer y para qué sirve; como se trata de un componente electrónico nuevo para nosotros, trabajaremos en la realización de un pequeño proyecto que nos permita utilizarlo en conjunto con una placa Arduino.

En este proyecto buscamos lograr varias tareas:

- **1. Conectaremos el buzzer directamente a la placa Arduino UNO y generaremos el código necesario para que este componente emita sonidos.**
- **2. Con el buzzer conectado a la placa Arduino, intentaremos reproducir una serie de sonidos que representen una escala musical.**
- **3. Continuando con el mismo circuito original, intentaremos combinar una cantidad de sonidos para que se escuche una melodía reconocible, para ello trabajaremos sobre el sketch básico que resultó de la tarea número dos.**

Manos a la obra

Dividiremos el trabajo en secciones; en cada una de ellas crearemos el circuito y el código que necesitamos, de esta forma iremos completando el proyecto al tiempo que realizamos las tareas propuestas.

Resultado 1

Para comenzar a trabajar, recordaremos nuestra primera tarea:

- **1. Conectaremos el buzzer directamente a la placa Arduino UNO y generaremos el código necesario para que este componente emita sonidos.**

Para lograrlo, solo necesitamos tres componentes: una placa Arduino, un protoboard y un buzzer. La conexión que debemos efectuar es bastante sencilla; en realidad, podemos prescindir del protoboard pues solo necesitamos conectar ambos extremos del buzzer a los pines 13 y GND. Considerando que el buzzer posee polaridad, es necesario conectar el cable rojo al pin 13 y el cable negro al pin GND.

Una vez que hemos realizado las conexiones necesarias, debemos trabajar en el sketch. Al igual que el circuito, el código para hacer funcionar el buzzer será bastante sencillo. Crearemos las variables que utilizaremos en el sketch, en esta ocasión necesitamos dos variables, una para asignar el pin que utilizaremos y otra para almacenar la frecuencia a la que se emitirá el tono, en este caso usamos el pin 13 y trabajamos con una frecuencia de 220, que corresponde a la nota **la**:

```
intpinsonido = 13;
int frecuencia = 220;
```

Reproducir archivos de audio

Gracias a la librería **SDplayWAV**, es posible reproducir archivos de audio que se encuentren almacenados en una tarjeta SD conectada a la placa Arduino. Esta librería fue desarrollada por David Cuartielles, uno de los desarrolladores de Arduino, y la podemos conseguir visitando la dirección web **http://blushingboy.net/p/SDplayWAV**. La librería se descarga en formato zip y posee un peso de 59,71 KB.

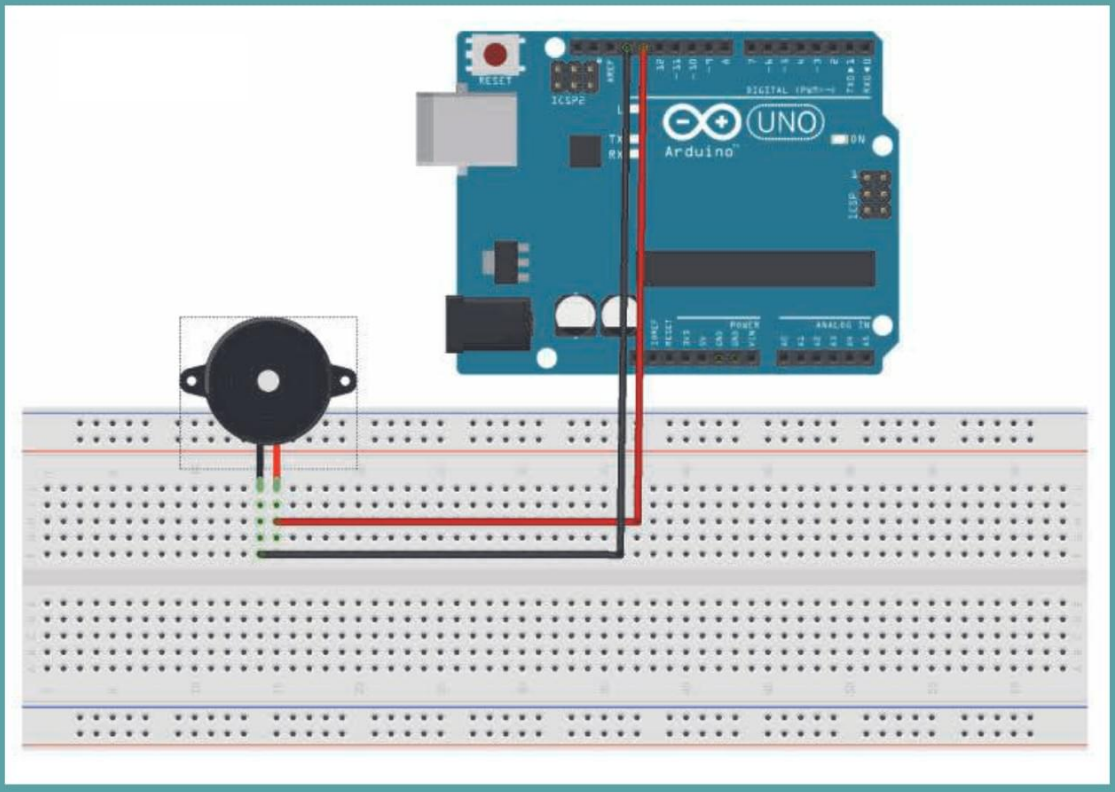
Para este sketch no necesitamos integrar código en **setup()**, por esta razón lo dejamos vacío, aunque sí lo escribimos:

```
voidsetup()
{
}
```

En el **loop()** es necesario establecer el inicio del zumbido, para ello usamos la función **tone**, que conocimos en la sección anterior; debemos pasar como parámetros las variables que definimos al inicio del sketch, de la siguiente forma:

```
tone(pinsonido,frecuencia);
```

+ Conexión del buzzer a la placa Arduino



Tal como vemos en este diagrama, la conexión del buzzer a la placa Arduino es realmente sencilla, pues solo necesitamos utilizar los pines 13 y GND.

Para detener el zumbido usamos la función **noTone**, pasando como parámetro el pin que estamos utilizando:

```
noTone(pinsonido);
```

Ya hemos definido el inicio del zumbido y también su final, pero, para que este bloque de instrucciones tenga sentido, es necesario anteponer un delay; de esta forma el zumbido se presentará de manera intermitente, por ejemplo, podemos utilizar un delay de un segundo:

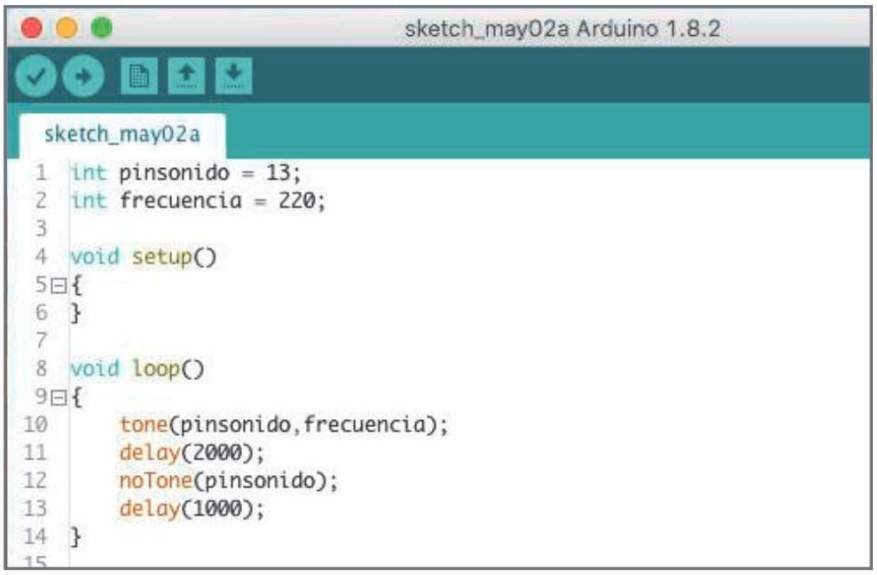
```
tone(pinsonido,frecuencia);
delay(1000);
noTone(pinsonido);
delay(1000);
```

Con esto, nuestro código completo quedará de la siguiente forma:

```
intpinsonido = 13;
int frecuencia = 220;

voidsetup()
{
}

voidloop()
{
tone(pinsonido,frecuencia);
delay(2000);
noTone(pinsonido);
delay(1000);
}
```

Ahora podemos ejecutar el Arduino IDE y cargar el código propuesto, luego lo compilamos para verificar que no exista algún error.

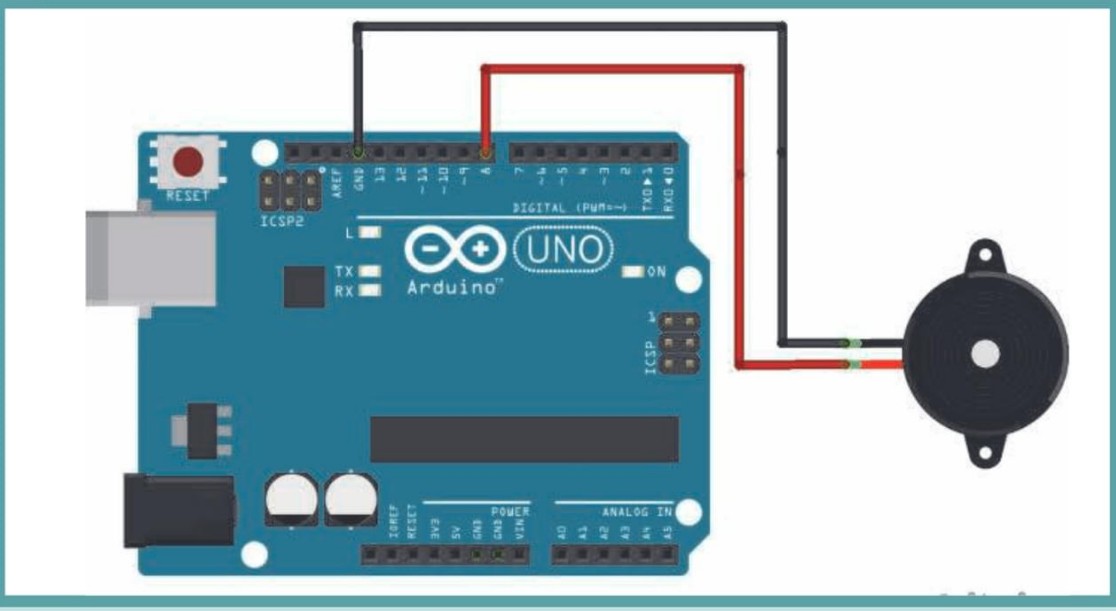
Una vez que hemos compilado el programa lo cargamos en la placa Arduino; al hacerlo notaremos que el buzzer emite un zumbido, esto se debe a que estamos usando el pin 13 que se posiciona en **HIGH** cuando energizamos la placa Arduino.



Después de cargar el código en el Arduino IDE, lo compilamos para verificar que no existan errores lógicos; con la compilación correcta, lo cargamos en la placa Arduino.

Podemos trabajar con este código modificando el delay que hemos asignado, de esta forma obtendremos nuevos resultados. También es posible modificar la frecuencia que establecimos, aunque consideremos que un zumbador tiene una capacidad limitada para reproducir sonidos en forma fiel, por lo tanto, con algunas frecuencias ni siquiera responderá.

+ Conectar el buzzer directamente a la placa



Podemos lograr el mismo resultado conectando el buzzer directamente a la placa Arduino, sin necesidad de utilizar el protoboard como intermediario.

Resultado 2

▶ **2. Con el buzzer conectado a la placa Arduino, intentaremos reproducir una serie de sonidos que representen una escala musical.**

Como ya hemos creado el circuito básico, solo nos queda trabajar sobre el código que necesitamos. Primero establecemos la variable que corresponde al pin que utilizaremos como salida, de la siguiente forma:

```
  intpinSonido = 13;
```

Además, es necesario definir una variable que almacene la cantidad de tonos que reproduciremos y, también, debemos definir un vector que almacene los tonos que correspondan a las notas de la escala natural, tal como vemos a continuación:

```
  intcantTonos = 10;
  int tonos[] = {261, 277, 294, 311, 330, 349, 370,
392, 415, 440};
```

En **setup()** crearemos un bucle **for** que pueda recorrer el vector que almacena los tonos de la escala; de esta forma, el buzzer emitirá una frecuencia por cada elemento que compone el vector, así escucharemos los diferentes tonos que, por supuesto, corresponden a la escala natural:

```
  for (int i = 0; i <cantTonos; i++)
    {
      tono(pinSonido, tonos[i]);
  delay(800);
    }
  noTone(pinSonido);
  }
```

En este caso no utilizaremos **loop()**, por lo que la dejamos vacía:

```
  voidloop()
  {
  }
```

Con todas las secciones definidas, nuestro sketch tendrá la siguiente apariencia:

```
  intpinSonido = 12;
  intcantTonos = 10;
  int tonos[] = {261, 277, 294, 311, 330, 349, 370,
392, 415, 440};

  voidsetup()
  {
  for (int i = 0; i <cantTonos; i++)
    {
  tone(pinSonido, tonos[i]);
  delay(800);
    }
  noTone(pinSonido);
  }
  voidloop()
  {
  }
```

Con el sketch completo, debemos cargarlo en el Arduino IDE, compilarlo y subirlo a la placa, para luego verificar su funcionamiento.

Resultado 3

Para lograr esta última tarea, recordemos la propuesta número tres:

- **3. Continuando con el mismo circuito original, intentaremos combinar una cantidad de sonidos para que se escuche una melodía reconocible, para ello trabajaremos sobre el sketch básico que resultó de la tarea número dos.**

Como vemos, hay que trabajar con el circuito conectado originalmente y también sobre el sketch que resultó de la tarea número dos.

Para este sketch, usaremos las mismas variables que creamos para el resultado número dos:

```
  intpinSonido = 12;
  intcantTonos = 10;
```

También necesitamos un vector que almacene los tonos que serán emitidos por el buzzer, pero en esta ocasión realizaremos algunas modificaciones, el código deberá quedar de la siguiente forma:

```
  inttonos[] = {261, 349, 392, 440, 392, 330, -10,
261, 349, 392, 440, 392, -10, -10, 261, 349, 392,
440, 392, 330, -10, 330, 349, 330, 261, 261};
```

Igual que en el caso anterior, necesitamos un bucle **for** que recorra el vector que contiene los tonos adecuados; de esta forma, el buzzer emitirá un sonido específico por cada elemento del arreglo:

```
  for (int i = 0; i <cantTonos; i++)
    {
  tone(pinSonido, tonos[i]);
  delay(500);
    }
  noTone(pinSonido);
```

Dejaremos **loop()** vacío, de la siguiente forma:

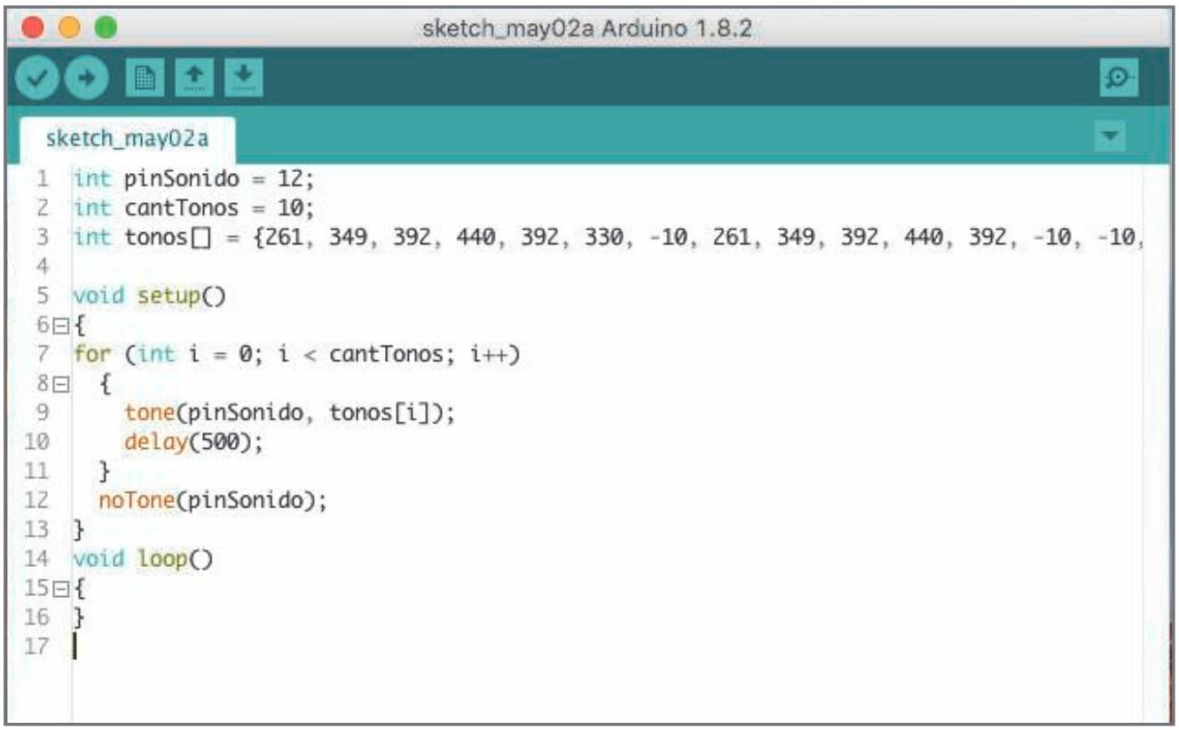
```
  voidloop()
  {
  }
```

La apariencia final del sketch es la siguiente:

```
  intpinSonido = 12;
  intcantTonos = 10;
  inttonos[] = {261, 349, 392, 440, 392, 330, -10,
261, 349, 392, 440, 392, -10, -10, 261, 349, 392,
440, 392, 330, -10, 330, 349, 330, 261, 261};

  voidsetup()
  {
  for (int i = 0; i <cantTonos; i++)
    {
```

```
tone(pinSonido, tonos[i]);
delay(500);
  }
noTone(pinSonido);
}
voidloop()
{
}
```

Una vez que hayamos escrito, compilado y cargado el código en la placa Arduino, podemos probarlo y verificar los sonidos que emite. Sobre este sketch es posible realizar las modificaciones que consideremos convenientes para obtener nuevos resultados, por ejemplo, podemos modificar los tonos y también el delay establecido para las pausas.



■ Ya hemos escrito y compilado el sketch en el Arduino IDE, podemos ver que la compilación se realizó sin errores lógicos, por lo que es posible subirlo a la placa Arduino.

Ejemplos del Arduino IDE

Lo que hemos logrado hasta este momento es solo una pequeña muestra de todas las posibilidades de la emisión de sonidos asociada a Arduino. Por ejemplo, podemos reproducir archivos .wav en forma directa gracias a la librería **TMRpcm**, que se consigue en la dirección web **https://github.com/TMRh20/TMRpcm/wiki**.

Encontramos otras posibilidades entre los ejemplos que se integran en la IDE; a continuación, revisamos el código que corresponde al ejemplo toneMelody, para el que solo necesitamos conectar un speaker al pin 8:

```
  #include "pitches.h"

  intmelody[] = {
    NOTE_C4, NOTE_G3, NOTE_G3, NOTE_A3, NOTE_G3, 0,
NOTE_B3, NOTE_C4
  };

  intnoteDurations[] = {
    4, 8, 8, 4, 4, 4, 4, 4
  };

  voidsetup() {
  for (intthisNote = 0; thisNote< 8; thisNote++) {
  intnoteDuration = 1000 / noteDurations[thisNote];
  tone(8, melody[thisNote], noteDuration);
  intpauseBetweenNotes = noteDuration * 1.30;
  delay(pauseBetweenNotes);
  noTone(8);
    }
  }

  voidloop() {
  }
```

Resumen Capítulo 09

En este capítulo profundizamos en un componente electrónico esencial para reproducir nuestros primeros sonidos desde una placa Arduino: el buzzer. Vimos sus características y también las diversas aplicaciones electrónicas en los que lo podemos encontrar. Definimos algunas tareas para completar mediante el uso de este componente y nos pusimos manos a la obra. En primer lugar, conectamos el buzzer a la placa Arduino y reprodujimos algunos sonidos, para ello usamos un código sencillo; más adelante realizamos ciertas modificaciones al código para lograr que el buzzer reprodujera una serie de sonidos que representan una escala musical. Finalmente, combinamos algunos sonidos para lograr reproducir una melodía.

Display LCD y reloj digital

10

A continuación, realizaremos un pequeño pero interesante proyecto: un reloj digital. Para lograrlo, utilizaremos algunos componentes que ya conocemos, pero también agregaremos una pantalla LCD.

DISPLAY LCD

Aunque estamos acostumbrados a asociar el término **LCD** con las pantallas de televisores o con los monitores de las computadoras, en realidad, los displays LCD poseen aplicaciones más diversas, por ejemplo, calculadoras, relojes, impresoras, y todo tipo de electrodomésticos.

Si nos remitimos a su significado, debemos saber que LCD corresponde a *Liquid Cristal Display* o, en español, 'pantalla de cristal líquido'. En pocas palabras, se trata de una pantalla plana que hace uso de una sustancia líquida entre dos placas de vidrio, por esta sustancia se hace pasar una corriente eléctrica que vuelve opacas ciertas zonas específicas; esto, sumado a la iluminación trasera, genera los caracteres que vemos en el display.

En las pantallas LCD de color, cada pixel se divide en tres subpixeles (rojo, verde y azul), por lo que cada pixel puede ser controlado para producir una gran variedad de colores.

Características

Para definir un display LCD debemos considerar diversas características, es importante tenerlas en cuenta para decidir qué tipo de display necesitamos para un proyecto en particular. A continuación, describimos algunas de estas características básicas en detalle:

Los displays LCD están presentes en una gran variedad de dispositivos electrónicos, por ejemplo, en una lavadora, un microondas o una calculadora.

TAMAÑO

Para medir el tamaño de un panel LCD, se utiliza el largo de su diagonal y, por lo general, se expresa en pulgadas. Pero, si estamos trabajando con displays pequeños, también pueden describirse sus dimensiones mediante la cantidad de caracteres que es capaz de mostrar. Por ejemplo, un display de 16×2 se refiere a un panel que puede mostrar dos filas de 16 caracteres, es decir, dos renglones con capacidad para 16 caracteres de manera horizontal al mismo tiempo.

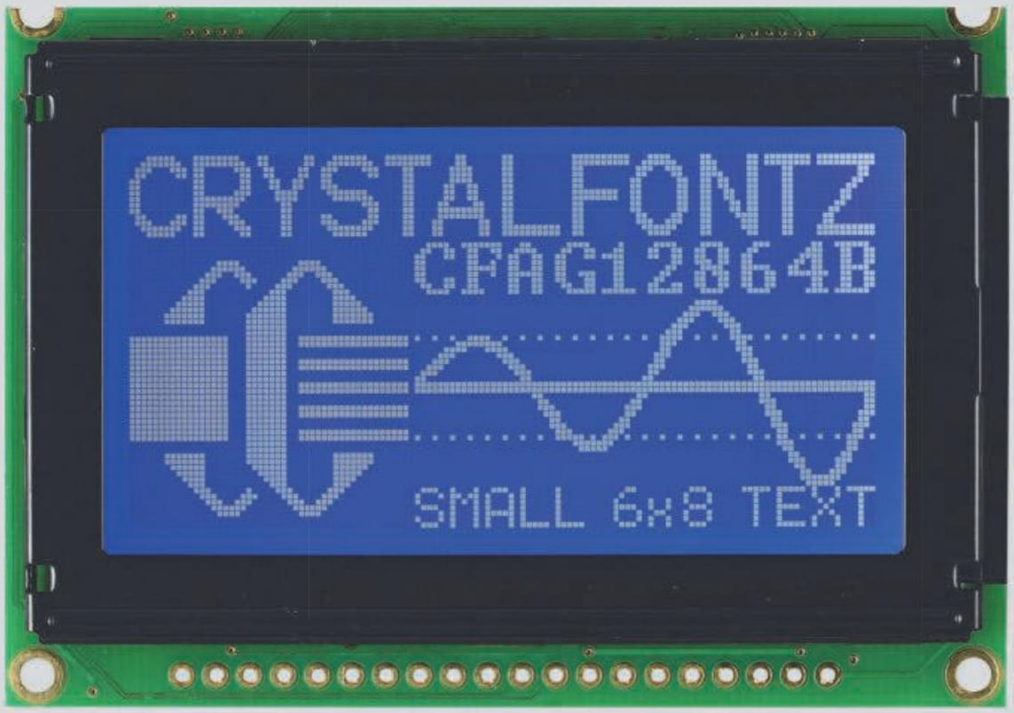
Aquí vemos un ejemplo de display LCD, se trata de un display que ofrece un total de cuatro líneas con la capacidad de mostrar veinte caracteres en cada línea, es decir, un display de 20x4.

Caracteres especiales

Aunque los dipslay LCD son capaces de mostrar cualquier caracter alfanumérico, en algunas ocasiones necesitaremos definir nuestros propios caracteres. Para lograrlo debemos saber que los caracteres se definen mediante un array de 8×8, de esta forma, será como si los dibujáramos en una cuadricula, por lo que será necesario rellenar cada segmento hasta lograr el carácter adecuado.

RESOLUCIÓN

Esta característica puede expresarse mediante las dimensiones horizontal y vertical. Por ejemplo, las pantallas HD poseen una resolución de 1920×1080. Un ejemplo de un panel LCD que podemos utilizar para aplicaciones y proyectos electrónicos es una LCD gráfica de 128×64, de color negro sobre fondo verde. Aunque puede parecer un display muy pequeño en comparación con los LCD HD, es más que suficiente y práctico para proyectos de electrónica, incluso, industriales.



Un display gráfico de 128x64, en este caso con luz azul, abre una serie de posibilidades para mostrar información en nuestros proyectos con placas Arduino.

BRILLO

Otra de las características importantes en un display LCD es su brillo; dependiendo de la aplicación en la que lo utilizaremos, requeriremos más o menos luz. Debemos tener en cuenta que la mayoría de estos paneles posee una luz de fondo y es posible controlar su luminosidad, por lo que estamos frente a una característica que podemos alterar. Por ejemplo, los displays que poseen iluminación CCFL ubican, en su parte trasera o en sus bordes, una matriz de ▶

CCFL, pero presentan un mayor consumo con un tiempo de vida más reducido que la iluminación LED. Por otra parte, la iluminación LED puede encontrarse en un solo color o también en RGB, aunque la iluminación blanca es más usada. La iluminación LED también puede presentarse como una matriz de fondo o estar en los bordes del display.

CONTRASTE

Se refiere a la relación que existe entre la intensidad más brillante y la más oscura.

ÁNGULO DE VISIÓN

Esta característica corresponde al ángulo máximo que el usuario puede utilizar para visualizar lo que aparece en el display sin que se pierda demasiada calidad.

NÚMERO DE CARACTERES

Como imaginamos, existen diversos tamaños de LCD. Para nuestros proyectos electrónicos, necesitamos saber la cantidad de caracteres que es capaz de mostrar, por ejemplo, existen algunos tamaños más utilizados, como 16×2, 20×4 u 8×2 caracteres.

Este es uno de los displays más utilizados en proyectos electrónicos de pequeña envergadura, se trata del panel de 8x2.

Comunicación

Ya conocemos qué es un display LCD y cuáles son sus características, ahora revisaremos las formas en que podemos comunicarnos con este componente:

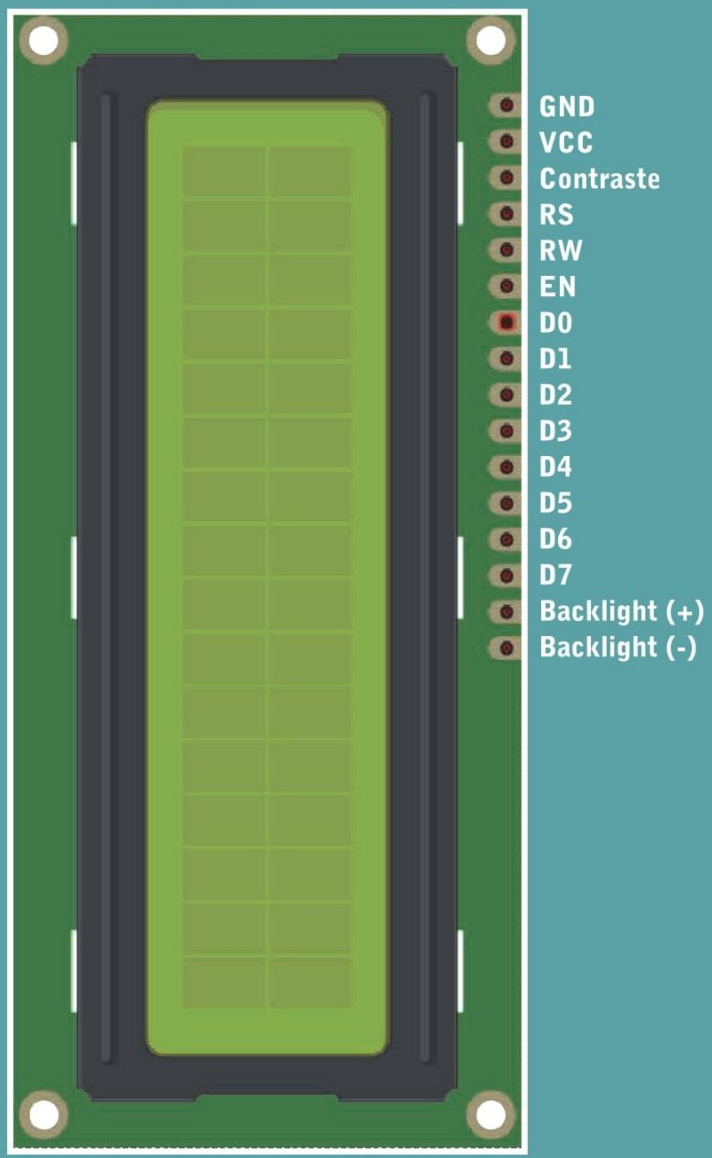
- **SERIAL UART**: se debe utilizar un **backpack** para comunicarse usando los puertos TX y RX del microcontolador. Podemos conseguir uno de estos elementos con soporte para 2 o para 4 líneas, y también para gráficos entre 160×128 y 128×64.
- **SERIAL A PARALELO**: necesita contar con un circuito integrado que se conecte por I²C o SPI al puerto SDA, y el reloj al SCL de Arduino con la librería **Wire**.
- **EN PARALELO**: se trata de la opción más utilizada, se puede realizar la comunicación en forma directa con Arduino utilizando los pines del 0 al 7, o del 0 al 4 si queremos enviar la mitad de los datos; esto hará el proceso más rápido y la diferencia no es detectada por el ojo humano.

Por ejemplo, para conectar un display de 16x2 necesitamos considerar que, en el panel LCD encontraremos los siguientes pines:

- **RS (Register Select)**: se trata del pin que controla la memoria del LCD e indica qué registro de la memoria será el que se lee o escribe. El pin RS se encarga de controlar en qué lugar de la memoria LCD se escriben los datos, por lo tanto, mantiene lo que veremos en pantalla o donde se buscarán los siguientes datos para mostrar.
- **RW (Read/Write)**: es el pin de lectura y escritura que dirá si se escribe en memoria o si se lee en cada momento, por lo tanto, permite elegir el modo de lectura o de escritura.
- **E (Enable)**: se trata del pin que habilita los registros.
- **DB0-DB7**: corresponden a los pines de datos, es decir, son los pines desde los que se obtienen los bits que llegan al registro. La numeración de estos pines puede variar dependiendo del fabricante o también del modelo del display. Debemos tener en cuenta que los valores de estos pines son bits que se escriben en un registro o valores que se están leyendo.
- **Vo**: es el pin de contraste. Mediante este pin es posible alterar el contraste de la pantalla para que los caracteres se aprecien de mejor forma.
- **Vdd**: se trata del pin de alimentación, por el que entra la tensión que normalmente es de +5 V.

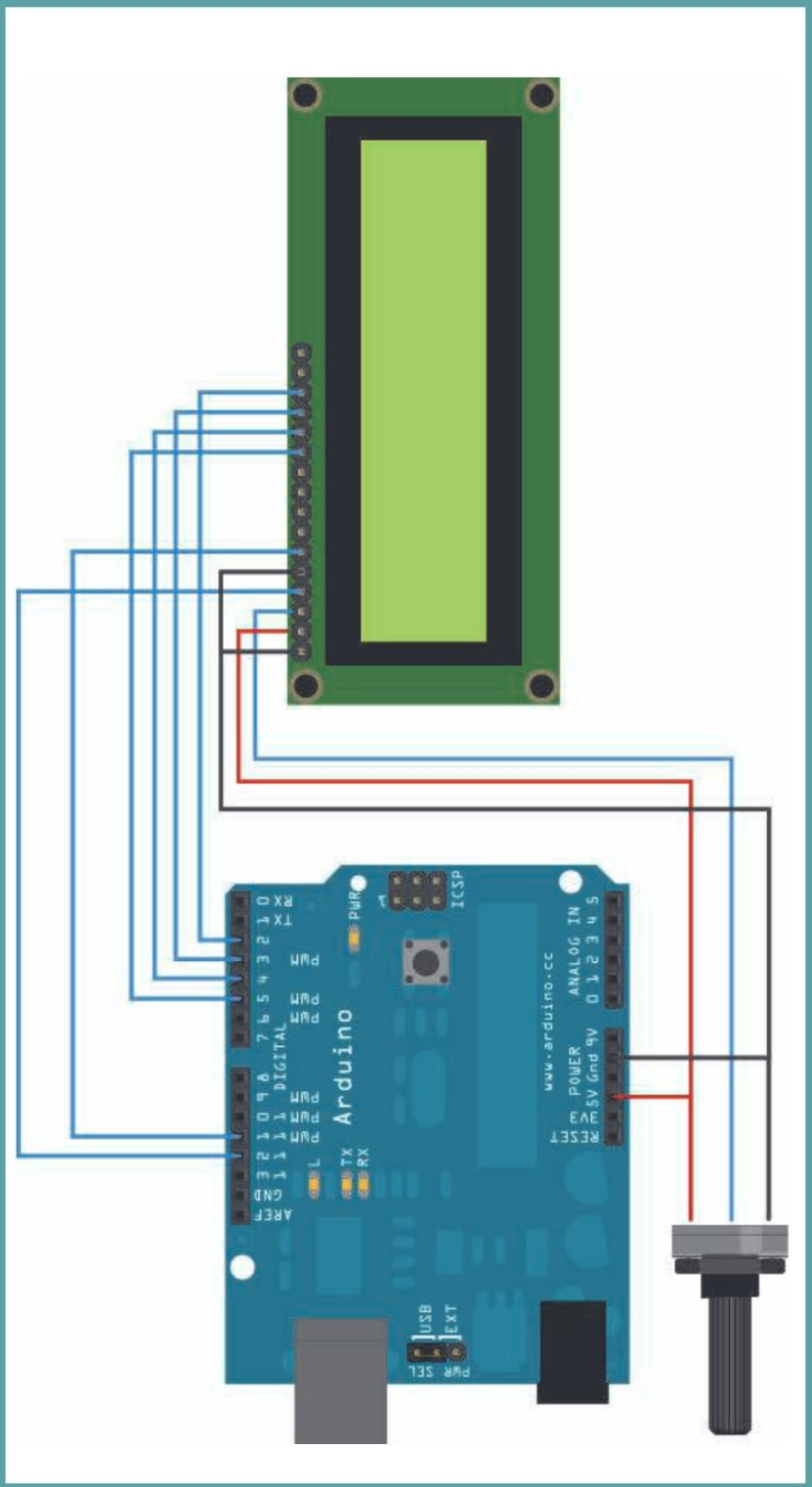
- **GND**: este pin se encarga de complementar al pin anterior, se trata del pin de alimentación que se conecta a tierra.
- **BL1 y BL2**: corresponden a los pines de retro iluminación. Se encargan de controlar la luminosidad del display, también pueden aparecer con los símbolos Bklt+ y Bklt-.

+ Pines de un display LCD



En esta imagen vemos un esquema de los pines que encontramos en un display LCD. En este caso se trata de un display de 16x2.

Conexión de un display LCD a una placa Arduino



En este esquema observamos un ejemplo de conexión de un display LCD con una placa Arduino. En este caso se muestra la conexión directa, es decir, sin utilizar un protoboard.

Para lograr una conexión en paralelo de un display LCD con la placa Arduino, debemos observar lo siguiente:

- El pin RS del display debe conectarse a la E/S digital en el pin 12.
- El pin Enable del display se conecta a la E/S digital en el pin 11.
- Los pines D4 a D7 se conectan a las E/S digitales desde el pin 5 hasta el pin 2 de la placa Arduino.
- Los pines de voltaje y tierra se conectan a +5 V y GND.
- El pin Vo, que se encarga de controlar el contraste, se conecta a un potenciómetro. Luego de esto, es posible ajustar el potenciómetro para que el texto se presente con el contraste que deseemos.

Para lograr una conexión correcta del display LCD, podemos hacer uso de un esquema de conexión, esta es la forma más rápida de entender de qué manera debemos conectar la placa, utilizando líneas claras y mostrando los componentes mediante símbolos electrónicos.

Con la conexión realizada, intentemos mostrar nuestros primeros caracteres en el display. En primer lugar debemos incluir la librería **LiquidCrystal**, que conoceremos más adelante, en este mismo capítulo:

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

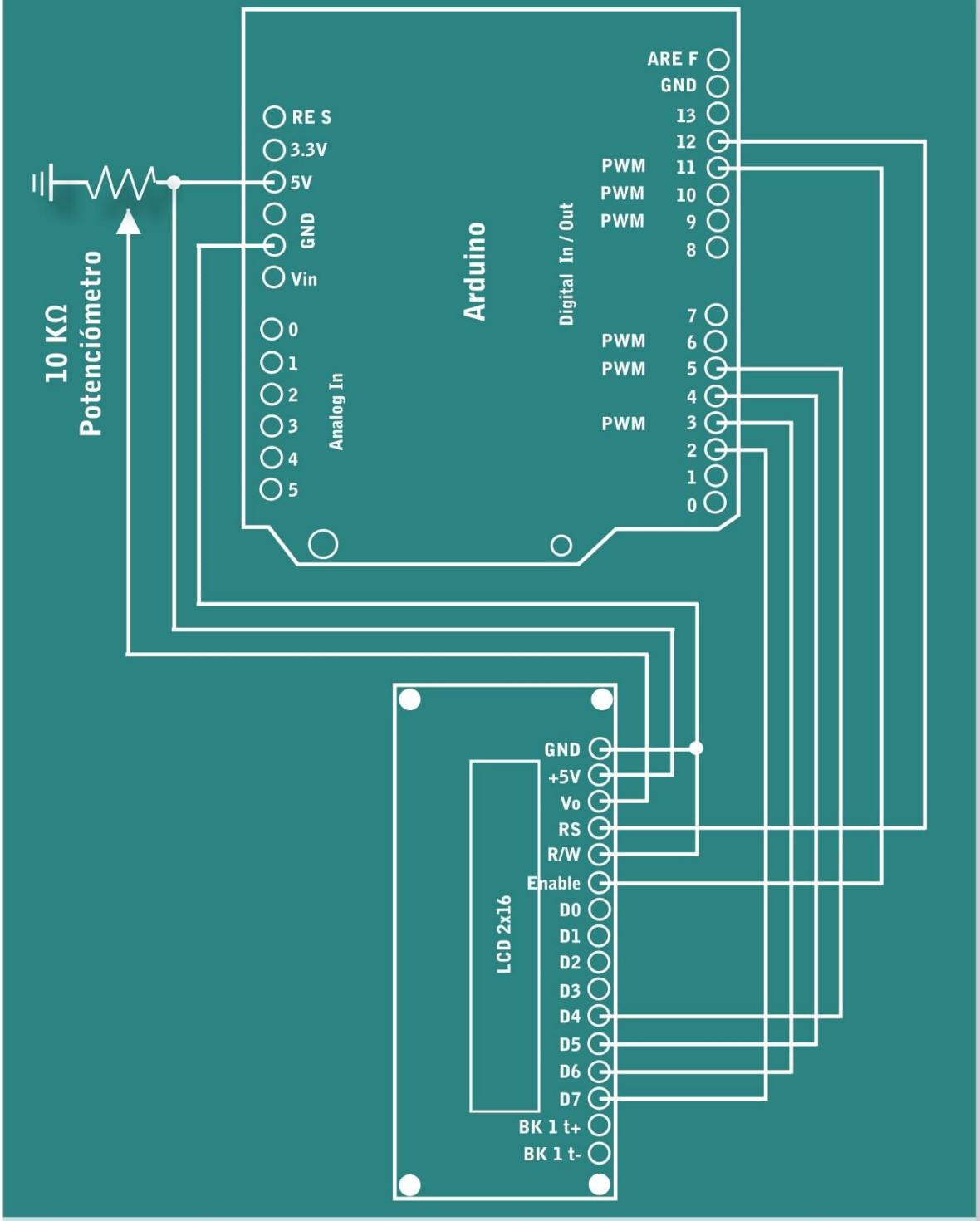
También es necesario que iniciemos los pines que utilizaremos, de la siguiente forma:

```
LiquidCrystal lcd(12,11,5,4,3,2);
```

Display LCM 1602C

Por ejemplo, en el caso de este display, cuando miramos desde atrás nos daremos cuenta de que la numeración de los pines comienza de derecha a izquierda y se presenta de la siguiente forma: 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1. En este caso, el pin1 concierne a GND; el 2 es la entrada de tensión Vdd; el 3 corresponde a Vo; el 4, a RS; el 5 es el pin RW; el pin 6 es E; 7 a 14 corresponden a Dbx; 15 es BL1, y 16 es BL2.

Esquema de conexión de un display LCD



Si seguimos este esquema, es realmente sencillo lograr la conexión correcta de un display LCD con la placa Arduino.

Para continuar, indicamos el número de columnas que corresponden al display que utilizaremos, en este caso se trata de un panel de 16x2:

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

Ahora sí podemos imprimir un mensaje por pantalla, para lograrlo usamos **lcd.print**:

```
lcd.print(«Primer mensaje»);
```

Finalmente, pondremos el cursor en la columna 0 de la línea 1, para señalar el lugar desde el que comenzarán a imprimirse los caracteres indicados más arriba:

```
lcd.setCursor(0,1);
```

El código completo quedará de la siguiente forma:

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12,11,5,4,3,2);

void setup() {
 lcd.begin(16, 2);
 lcd.print("Primer mensaje");
}

void loop() {
 lcd.setCursor(0,1);
}
```

Como sabemos, ahora solo nos queda compilar este código y luego subirlo a la placa Arduino. Si el texto no se aprecia correctamente, podemos utilizar el potenciómetro para ajustar el contraste.

"En este punto el potenciómetro nos ayudará en el ajuste de contraste del display LCD."

Como vemos en esta imagen, hemos compilado este sencillo código que nos permitirá mostrar un pequeño mensaje en la pantalla del display de 16x2.

Librería LiquidCrystal

Esta librería nos proporciona lo que necesitamos para trabajar con un display LCD conectado a una placa Arduino. Su principal función es permitirnos crear un objeto que pueda representar al display, para el que existan las operaciones de bajo nivel, y que podamos gestionarlas en forma sencilla. Esta librería nos ofrece diversos métodos simples, tal como mencionamos a continuación:

- **Método LiquidCrystal()**: se trata del constructor de la clase **LyquidCrystal**, es adecuado para crear un objeto de esta clase, que usaremos para gestionar el display LCD y realizar las tareas que necesitamos. Los argumentos que recibe son un conjunto de números que corresponden a pines de la placa Arduino conectados a los pines del display. Algunos ejemplos de su uso son los siguientes:

```
LiquidCrystal MiDisplay (12, 11, 5, 4, 3, 2);
LiquidCrystal(Rs, E, D4, D5, D6, D7);
LiquidCrystal(Rs, rw, E, D4, D5, D6, D7);
LiquidCrystal(Rs, E, D0, D1, D2, D3, D4, D5, D6,
D7);
```

- **Método begin()**: es necesario para inicializar el display, puede recibir dos argumentos –la anchura en caracteres y la altura en número de filas– que corresponden al display.
- **Método clear()**: este método se encarga de limpiar el display y poner el cursor en el primer carácter de la primera fila. Corresponde a borrar la pantalla del display.
- **Método home()**: este método se encarga de situar el cursor en el primer carácter de la primera fila, pero a diferencia del método **clear()**, no borrarrá el display.
- **Método setCursor()**: este método permite poner el cursor en una ubicación específica, los argumentos que recibe son la posición en la fila y también el número de fila. Para contar los espacios debemos comenzar desde cero.
- **Método write()**: se encarga de escribir una cadena en el display, el argumento que debemos pasar al método es la cadena que deseamos mostrar. A continuación vemos un pequeño ejemplo de este método:

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(7, 8, 9, 10, 11 , 12);

void setup() {
  lcd.begin(16, 2);
  lcd.write("Texto mostrado");
}

void loop() { }
```

- **Método print()**: este método es parecido a **write()**, pero permite enviar números enteros al display en distintas bases de numeración. Un ejemplo de su uso es el siguiente:

```
ObjetoLCD.print(numero, BASE);
```

En este código tenemos el parámetro **BASE**, se trata de una constante que puede ser BIN (binario), OCT (octal), DEC (decimal) o HEX (hexadecimal).

- **Método cursor() y noCursor()**: se utilizan para que el cursor sea visible en el display y para hacerlo invisible, respectivamente; no requieren parámetros.
- **Método blink() y noBlink()**: podemos usar estos métodos para hacer que el cursor parpadee o no lo haga, primero debemos hacerlo visible mediante el método cursor().
- **Método display() y noDisplay()**: mediante estos métodos es posible activar o apagar el display LCD. Estos métodos no reciben argumentos.
- **Método autoscroll() y noAutoscroll()**: gracias a estos métodos podemos hacer que el contenido más antiguo que se escriba en el display se desplace, desapareciendo por el principio de la línea, de esta forma dejará espacio al final de la línea para mostrar los nuevos contenidos. Por otra parte, si deseamos desactivar el autoscroll, debemos usar el método **noAutoscroll()**.
- **Métodos scrollDisplayLeft() y scrollDisplayRight()**: estos métodos hacen que el contenido se desplace un carácter a la izquierda o a la derecha. En conjunto con el método **autoscroll()** lograremos que los caracteres se desplacen en el sentido que deseemos.
- **Métodos leftToRight() y rightToLeft()**: mediante estos métodos podemos indicar el sentido de la escritura en el display.
- **Método createChar()**: nos permite crear hasta ocho caracteres adicionales e identificarlos con los números del 0 al 7. Los caracteres se crean a partir de una matriz de cinco puntos de ancho por ocho de alto. Gracias a **createChar()**, es posible conseguir símbolos realmente vistosos; para probarlo podemos implementar el siguiente código de ejemplo, que dibuja un corazón en el display:

```
byte simbolo[8] = { B00000,
                    B01010,
                    B11111,
                    B11111,
                    B01110,
                    B00100,
                    B00000,
                    B00000 };
```

Como podemos darnos cuenta, el dibujo de un corazón se forma gracias a la disposición de los unos y los ceros en la pantalla, por

supuesto, los unos corresponden a los espacios que se mostrarán, y los ceros se mantendrán ocultos. Siguiendo la misma lógica, podríamos crear un icono de una batería llena:

```
byte simbolo[8] = { B01110,
                    B11111,
                    B11111,
                    B11111,
                    B11111,
                    B11111,
                    B11111,
                    B11111 };
```

O también, ensayar la creación de una batería vacía o con media carga, de la siguiente forma:

```
byte simbolo2[8] ={ B01110,
                    B10001,
                    B10001,
                    B10001,
                    B10001,
                    B10001,
                    B10001,
                    B11111 };

byte simbolo3[8] ={ B01110,
                    B10001,
                    B10001,
                    B10001,
                    B10001,
                    B11111,
                    B11111,
                    B11111 };
```

Para probarlo debemos integrar el código en un sketch completo, tal como vemos a continuación:

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(7,8,9,10,11,12);
```

```
byte simbolo1[8] = { B00000,
                     B01010,
                     B11111,
                     B11111,
                     B01110,
                     B00100,
                     B00000,
                     B00000 };

byte simbolo2[8] ={ B01110,
                     B10001,
                     B10001,
                     B10001,
                     B10001,
                     B10001,
                     B10001,
                     B11111 };

byte simbolo3[8] ={ B01110,
                     B10001,
                     B10001,
                     B10001,
                     B10001,
                     B11111,
                     B11111,
                     B11111 };

byte simbolo4[8] ={ B01110,
                     B11111,
                     B11111,
                     B11111,
                     B11111,
                     B11111,
                     B11111,
                     B11111 };

void setup() {
          lcd.createChar(0, simbolo4);
```

```
            lcd.createChar(1, simbolo3);
            lcd.createChar(2, simbolo2);
            lcd.createChar(3, simbolo1);

            lcd.begin(16, 2);

            lcd.write(byte(0));
            lcd.write(byte(1));
            lcd.write(byte(2));
            lcd.write(byte(3));
}
void loop() {}
```

RELOJ DIGITAL

Ya conocimos en detalle las características y el funcionamiento de un display LCD, también vimos algunos ejemplos básicos de funcionamiento utilizando la librería **LiquidCrystal**. Con toda esta información, podemos intentar crear un sencillo pero fascinante proyecto: un reloj digital. En primer lugar seguiremos las indicaciones que mencionamos en una sección anterior de este capítulo para conectar correctamente el display LCD a nuestra placa Arduino. Podemos seguir las indicaciones que vemos en el esquema adjunto.

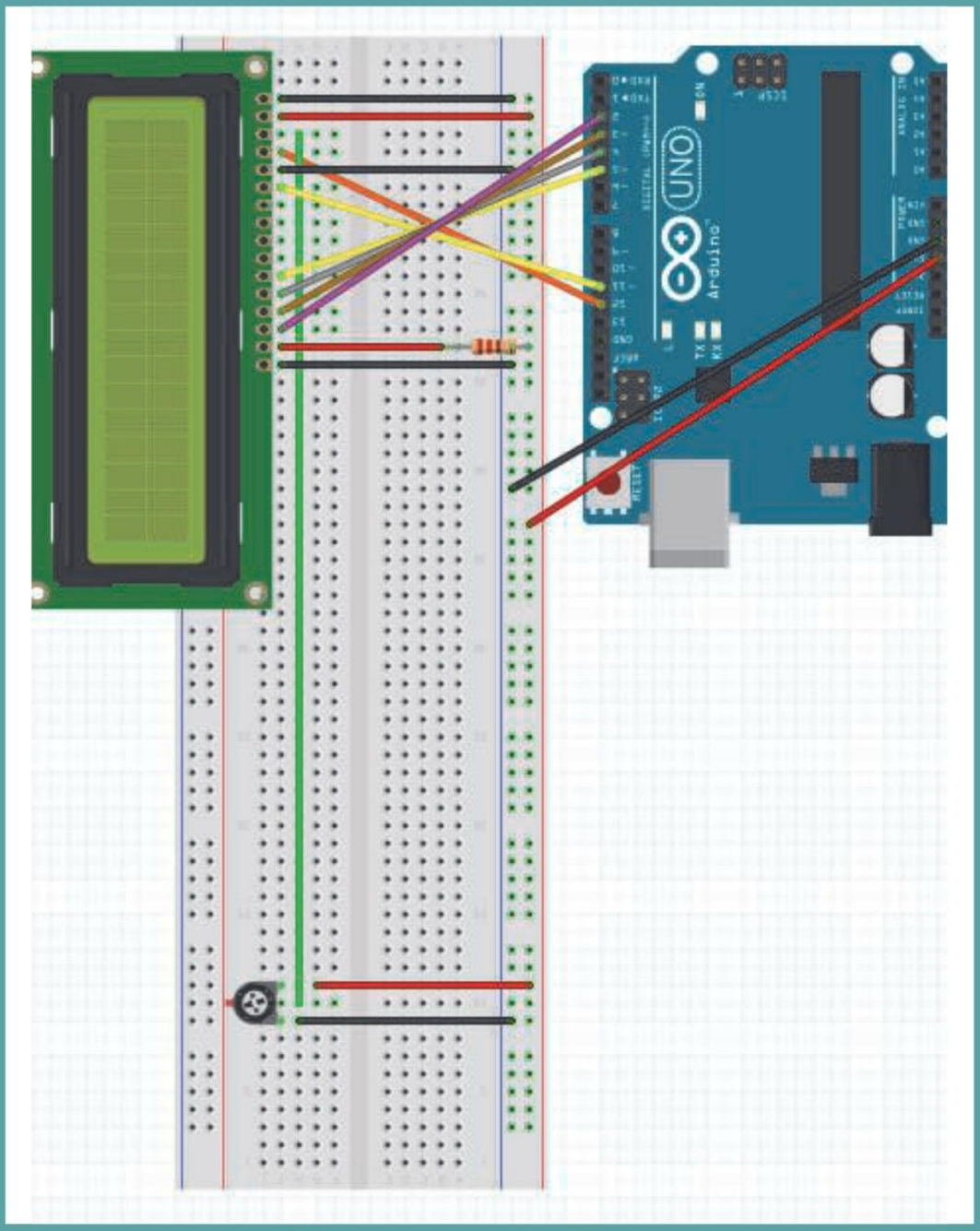
Para crear el sketch, en primer lugar incluiremos la librería **LiquidCrystal**:

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

Luego debemos indicar los pines que utilizaremos y, también, es necesario definir las variables en las que almacenaremos los segundos, los minutos y las horas (s, m y h, respectivamente):

```
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
int s = 0;
int m = 32;
int h = 16;
```

+ Conexión de un display LCD con protoboard



En este esquema se muestra la forma en que debemos conectar el display LCD a la placa Arduino, utilizando un protoboard para que tengamos más espacio disponible y podamos efectuar las conexiones en forma holgada.

En **setup()** usaremos el método **lcd.begin()**, que ya conocimos en una sección anterior, sabemos que es necesario para inicializar el display; en este caso usaremos los argumentos adecuados para un display de 16x2:

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

En **loop()** indicamos la forma en que aumentarán los segundos, hasta 60 en forma indefinida:

```
for (s = 0; s < 60; s++){
    lcd.clear();
    lcd.setCursor(3, 0);
    lcd.print("HRS");

    lcd.setCursor(4, 1);
    lcd.print(":");

    lcd.setCursor(7, 0);
    lcd.print("MIN");

    lcd.setCursor(7, 1);
    lcd.print(m);

    lcd.setCursor(9, 1);
    lcd.print(":");

    lcd.setCursor(11, 0);
    lcd.print("SEG");

    lcd.setCursor(12, 1);
    lcd.print(s);

delay(1000);
  }
```

También es necesario indicar que, cuando los segundos pasen de 60, vuelvan a cero y aumenten en uno los minutos. Si los minutos pasan a sesenta, vuelven a cero y las horas aumentarán en uno:

```
  if(s > 59){
  s = 00;
  m = m+1;
  }

  if(m > 59){
  m = 00;
  h = h+1;
  }

  if(h == 24){
  h = 00;
  m = 00;
  s = 00;
  }

}
```

Una vez que tenemos los segmentos de código, debemos copiarlos en el Arduino IDE, y su apariencia será la siguiente:

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
int s = 0;
int m = 32;
int h = 16;

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

void loop() {
  for (s = 0; s < 60; s++){
    lcd.clear();
    lcd.setCursor(3, 0);
    lcd.print("HRS");

    lcd.setCursor(4, 1);
    lcd.print(":");
```

```
      lcd.setCursor(7, 0);
      lcd.print("MIN");

      lcd.setCursor(7, 1);
      lcd.print(m);

      lcd.setCursor(9, 1);
      lcd.print(":");

      lcd.setCursor(11, 0);
      lcd.print("SEG");

      lcd.setCursor(12, 1);
      lcd.print(s);

  delay(1000);
    }

    if(s > 59){
    s = 00;
    m = m+1;
    }

    if(m > 59){
    m = 00;
    h = h+1;
    }

    if(h == 24){
    h = 00;
    m = 00;
    s = 00;
    }

  }
```

Con el sketch generado, solo nos queda compilarlo y, en caso de que no encontremos errores lógicos, lo subimos a la placa Arduino para probar su funcionamiento.

Resumen Capítulo 10

En este capítulo conocimos los detalles y las características relacionados con los displays LCD, revisamos sus pines y también los tipos de displays LCD más comunes en el trabajo con Arduino. Analizamos la comunicación que estos componentes realizan con una tarjeta Arduino y presentamos la librería LiquidCrystal, que nos simplifica las tareas de comunicación y la forma en que podemos mostrar mensajes e interactuar con el display. Vimos algunos ejemplos de código que nos permitieron utilizar un display de 16x2 y, para terminar, completamos un pequeño proyecto que nos permitió crear un reloj digital.

Potencial de Arduino

11

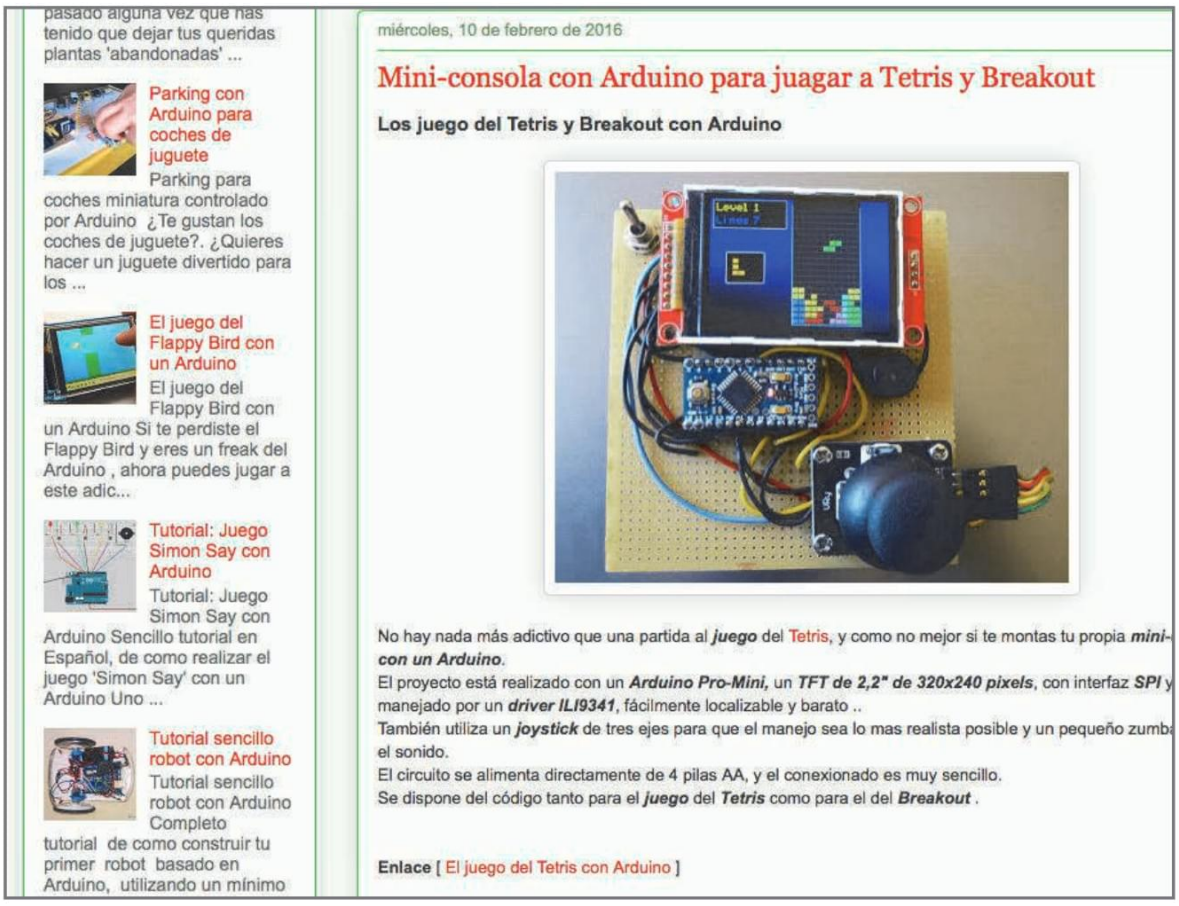
En los capítulos que componen esta obra, hemos dado los primeros pasos en el mundo de Arduino. Aunque construimos algunos proyectos interesantes, debemos tener en cuenta que el potencial de Arduino es enorme, y así lo demuestra la gran gama de posibilidades que conoceremos en este capítulo.

POSIBILIDADES

A través de los capítulos que componen esta obra, pudimos iniciar nuestro camino en el mundo de Arduino. Conocimos sus características, sus principales usos y, también, creamos nuestros primeros proyectos.

Aunque trabajamos con elementos tales como LEDs, sensores ultrasónicos y displays LCD, el potencial de Arduino está lejos de ser solo lo que aprendimos hasta este momento.

Arduino es mucho más. Esta plataforma de hardware y software flexible proporciona lo que necesitamos para construir cualquier proyecto que nos imaginemos; por esta razón siempre debemos tener en cuenta que las posibilidades solo están definidas por nuestro conocimiento de Arduino y por nuestra creatividad.



■ En el sitio web **http://proyectos-sobre-arduino.blogspot.cl**, encontramos una colección de proyectos que pueden ser realizados mediante el uso de una placa Arduino.

Ya dimos los pasos básicos en el trabajo con Arduino, vimos la forma en que podemos trabajar con LEDs y diversos tipos de sensores; también aprendimos a mostrar información mediante el monitor serie y a través de un display LCD. Ahora solo nos queda explorar este mundo más profundamente y lograr lo que nuestra imaginación nos dicte.

Para analizar el verdadero potencial de Arduino, conoceremos algunos proyectos interesantes que han sido desarrollados por la comunidad que se encuentra alrededor de esta plataforma, de esta manera podremos darnos cuenta de las reales posibilidades que nos ofrece.

Arduino en domótica

Uno de los campos más interesantes donde Arduino ha encontrado enormes posibilidades de aplicación es la **domótica**.

La domótica es definida como el conjunto de técnicas que buscan automatizar una vivienda, que integran la tecnología en los sistemas de seguridad, gestión energética, bienestar o comunicaciones. Y es precisamente en esto donde Arduino puede ayudarnos, permitiéndonos efectuar tareas como las siguientes:

- Control de iluminación con control de potencia.
- Control de persianas y toldos.
- Control de climatización.
- Control de calefacción con temperaturas de consignas y control de riego automático.
- Programación de horarios y también programación de escenas o ambientes.
- Posibilidad de acceso y control desde fuera de la red local.
- Control de todo tipo de dispositivos que usen controles remotos.

"Las posibilidades que ofrece Arduino para el control domótico son muy interesantes, en www.descubrearduino.com encontramos algunos ejemplos en este ámbito."

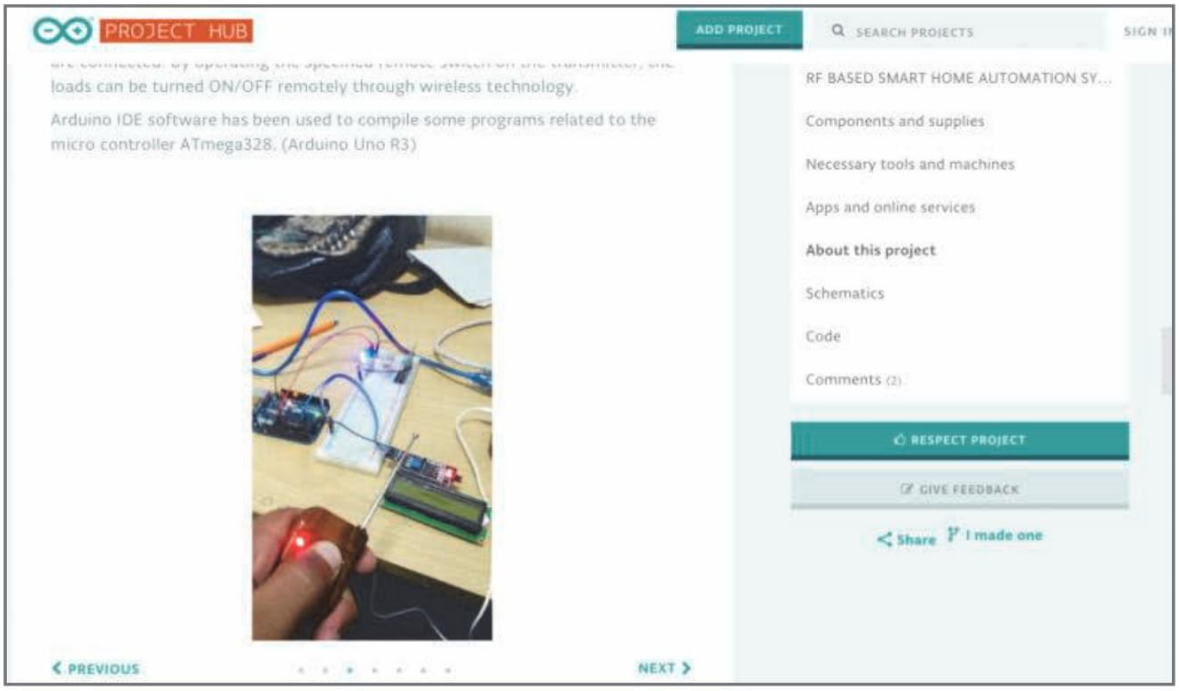
Entre los proyectos que relacionan Arduino con la domótica, encontramos un sistema inteligente para el hogar, basado en el uso de la radiofrecuencia. En este proyecto se busca desarrollar un completo sistema de automatización sustentado en un transmisor de radiofrecuencias.

En su construcción se usa la placa Arduino UNO y algunos componentes adicionales, como un display LCD.

Se trata de un proyecto sencillo que podemos replicar siguiendo las indicaciones publicadas en la Web, para ello debemos visitar la dirección **https://create.arduino.cc/projecthub/dennismwanza**.

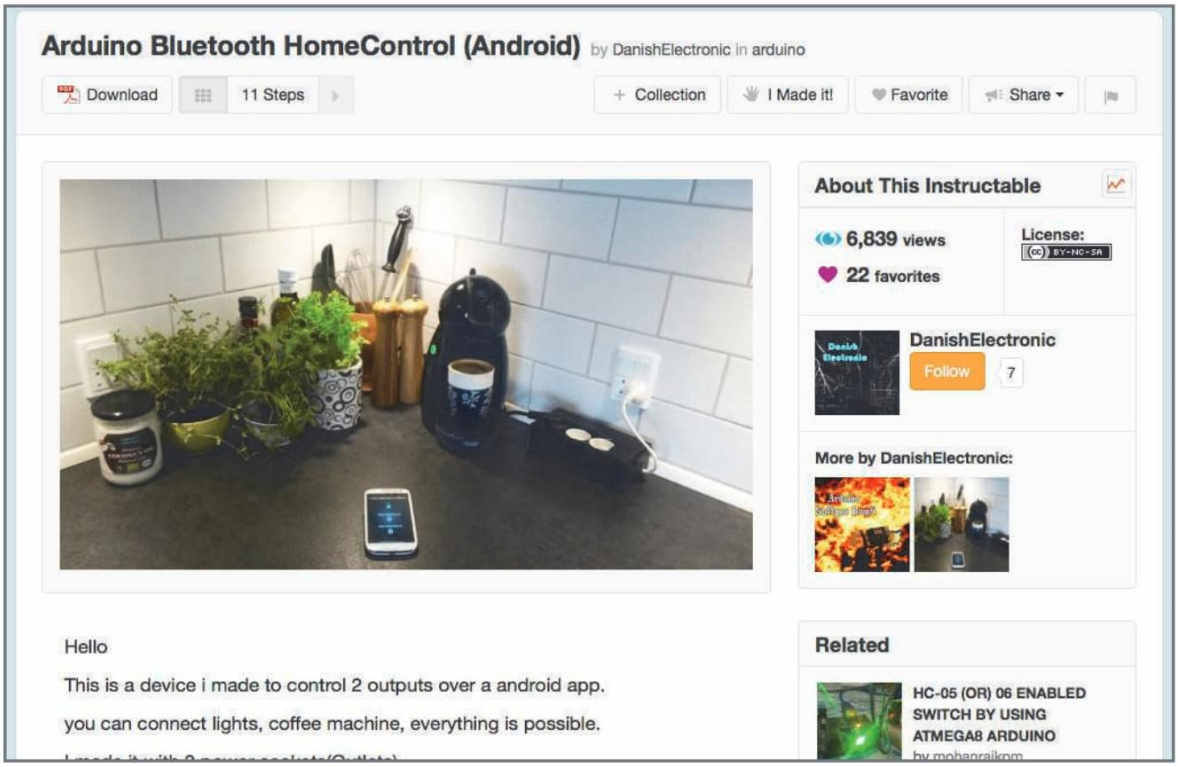
Si nuestro objetivo es utilizar Android para controlar diversos dispositivos electrónicos presentes en el hogar, podemos hacer uso de Arduino. En la web **www.instructables.com/id/Arduino-Bluetooth-HomeControl-Android** se encuentran las indicaciones completas para replicar este proyecto de domótica. En este caso se hace uso de una placa Arduino Nano, aunque también podemos usar Arduino UNO o Mega. En términos simples, la placa Arduino se conectará a una base de enchufes y, utilizando un equipo móvil con sistema Android, se conectarán los aparatos mediante bluetooth.

Pero no solo es posible controlar dispositivos electrónicos que comúnmente se encuentran en un hogar, también podemos extender el control a un pequeño jardín. Para lograrlo hay que replicar el proyecto que se encuentra en la dirección **www.arielmax.com.ar/proyecto-arduino-riego-automatico-para-plantas**.



En la web del proyecto obtendremos el listado de componentes que necesitamos y las instrucciones paso a paso para replicarlo.

Gracias a este proyecto, controlaremos el riego en forma automática y también mediremos la humedad en nuestras plantas.



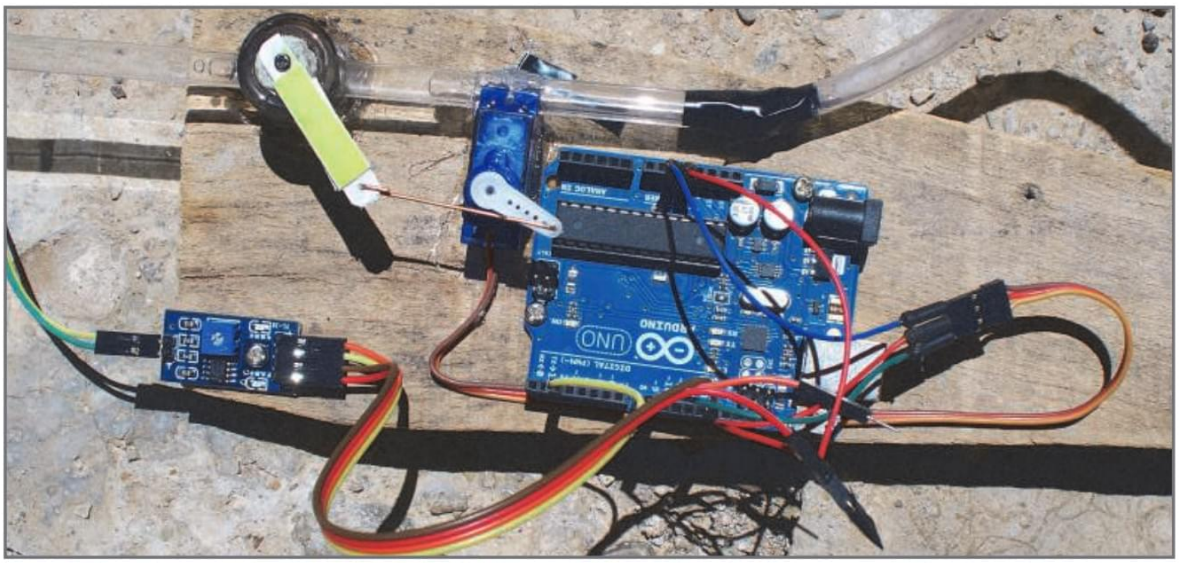
■ Este proyecto de domótica con Arduino se divide en 11 pasos. Es posible descargar las instrucciones en formato PDF, haciendo clic en **Download**.

Para replicar este proyecto de riego automático, debemos seguir las instrucciones propuestas por el autor en su sitio web oficial y utilizar el siguiente código:

Luces ambientales

Un interesante proyecto que utiliza una placa Arduino se relaciona con la implementación de luces ambientales para el monitor. Gracias a este proyecto, que encontramos en el sitio web **http://dhowdy.blogspot.cl/2011/09/diy-arduino-ambilight-using-shiftbrites.html**, se agregan luces en la parte posterior de un monitor para generar un ambiente más acorde con la actividad que realizamos frente a la PC.

```
#include ;
constintsensorPin = 2;
constintledPin = 12;

int estado = 0;
intestadoOff =0;
intsensorState = 0;
Servo myservo;

voidsetup(){
pinMode(ledPin, OUTPUT);
pinMode(sensorPin, INPUT);
myservo.attach(9);
}

voidloop(){
sensorState = digitalRead(sensorPin);

if (sensorState == LOW){
if(estado == 0){
myservo.attach(9);
```



En la web oficial de este proyecto, el autor nos entrega una imagen que corresponde al prototipo ya creado y en funcionamiento.

```
myservo.write(120);
delay(500);
myservo.detach();
estado = 1;
estadoOff = 0;
}
}
else {
if(estadoOff == 0){
myservo.attach(9);
myservo.write(0);
delay(500);
myservo.detach();
estadoOff = 1;
estado = 0;
}
}
}
```

Para obtener una explicación más precisa sobre las diferentes secciones de este código, visitemos el sitio web oficial del proyecto.

Arduino en robótica

Sin duda, la robótica es una de las áreas en las que, gracias a la ayuda de la plataforma Arduino, se han creado los proyectos más impresionantes.

Uno de estos proyectos es **GarabatoBOT**, un robot creado para dibujar garabatos sencillos sobre una pizarra acrílica vertical.

El funcionamiento de este robot se basa en una estructura impresa en 3D, una placa Arduino Pro Mini y otros componentes que se listan en su sitio web oficial: **https://madebyfrutos.wordpress.com/2012/03/25/proyecto-de-mig-3**.

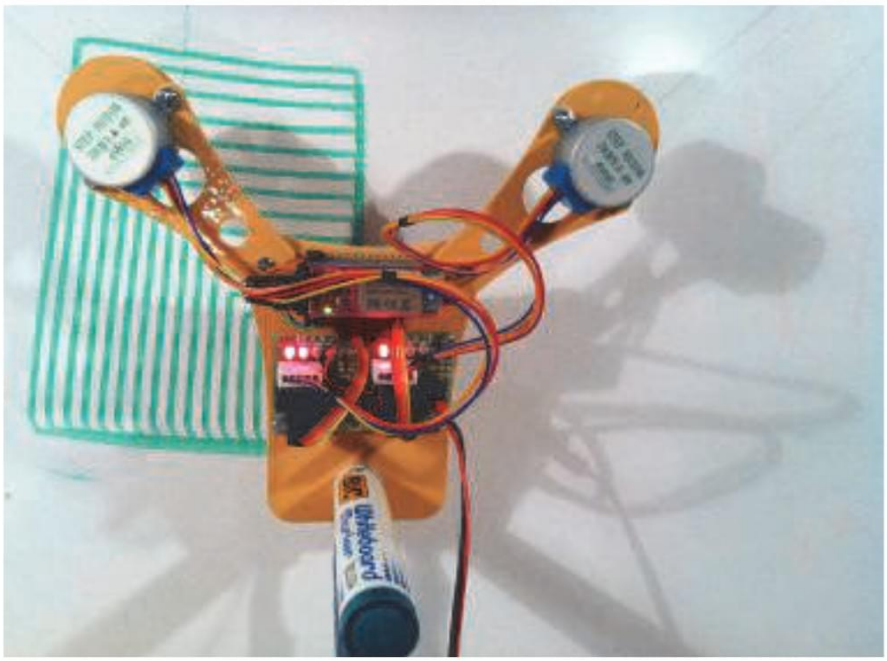
Como vemos en el sitio oficial del proyecto, su instalación es simple pues se utilizan dos cuerdas adosadas a las esquinas superiores de la pizarra y conectadas a poleas que sujetan los motores, de esta forma se logra el movimiento y las marcas realizadas por un lápiz para pizarra asegurado al cuerpo del robot.

DoodleBOT

De madefrutos 25 marzo, 2012 MAF 28 comentarios

Proyecto de Miguel Ángel de Frutos

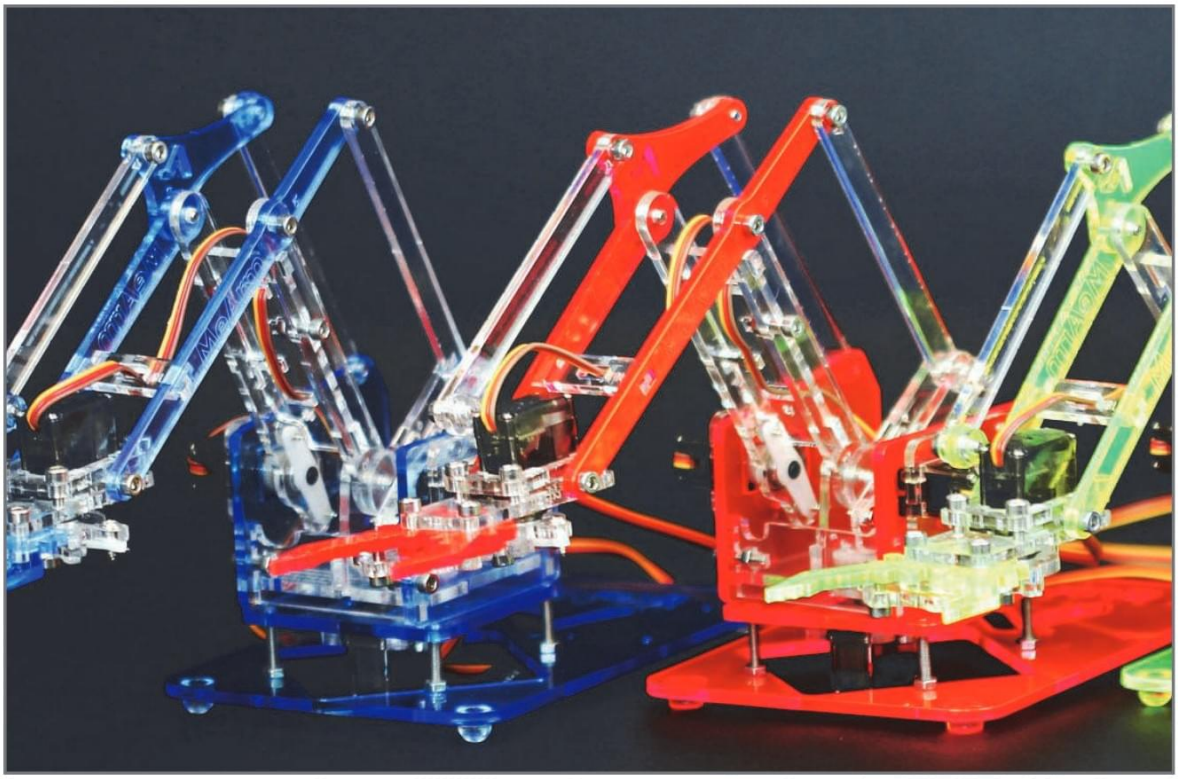
Hoy os presento a GarabatoBOT!



■ Este proyecto, creado por Miguel Ángel de Frutos, nos presenta una de las posibilidades de Arduino en la robótica.

Lector de huellas

Las posibilidades de Arduino también se relacionan con la seguridad, por ejemplo, en lo relacionado con la lectura de las huellas digitales. En el proyecto que encontramos en el sitio web **www.instructables.com/id/DIY-Fingerprint-Scanning-Garage-Door-Opener**, se utiliza una placa Arduino UNO para programar un ATtiny85 y un ATmega328 para detectar una huella y abrir la puerta de un garaje.



El kit que se comercializa gracias al proyecto MeArm entrega a los usuarios novatos todo lo necesario para construir su brazo robótico. Los usuarios con más experiencia pueden imprimir sus propias piezas.

Otra de las aplicaciones de Arduino en la robótica la encontramos en el proyecto **MeArm**, un brazo robótico que podemos adquirir a través de un kit que contiene todo lo necesario para construirlo.

Este brazo robótico, que se encuentra en la dirección web **https://store.hackaday.com/products/mearm-pocket-sized-robot-arm**, también puede construirse desde cero, solo debemos descargar los planos desde la web oficial e imprimirlo en 3D.

Arduino y drones

La construcción de drones no es un tema ajeno al mundo de Arduino, en la Web encontramos proyectos tales como **Ardupilot** (**http://ardupilot.org**) o **Arducopter** (**www.arducopter.co.uk**). Ambas opciones nos proporcionan completos tutoriales y ayuda para construir nuestros propios modelos voladores.

AeroQuad es un software de controlador de vuelo que funciona en Android; ofrece una completa documentación por lo que es una excelente oportunidad para trabajar en la construcción de drones y cuadricópteros. Su web se encuentra en **`http://aeroquad.com/content.php`**.

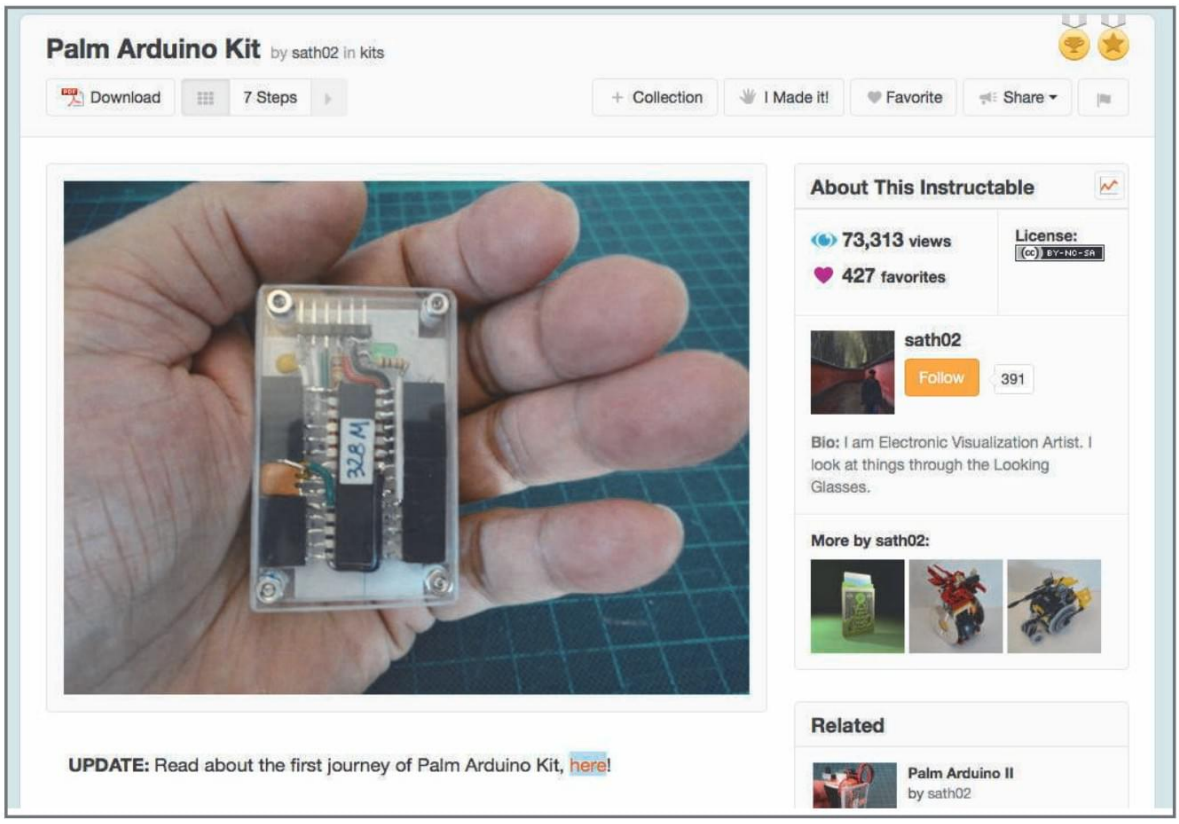
Aunque construir un dron puede parecer una misión demasiado compleja, la verdad es que se trata de uno de los proyectos más populares en la actualidad, y, gracias al controlador de vuelo YMFC-V2 3D, podremos hacerlo en forma sencilla, utilizando la placa Arduino UNO.

En la web oficial del proyecto (**www.brokking.net/ymfc-3d_v2_main.html**), su creador, Joop Brokking, nos ayuda en todo lo que necesitamos para construir nuestro propio dron basado en Arduino.

Construir tu propio Arduino

Hemos conocido algunos proyectos realmente fascinantes que se han creado con Arduino, pero eso no es todo, como ya hemos mencionado, las posibilidades son realmente infinitas. Por ejemplo, si consideramos que el proyecto Arduino es open source, ¿por qué no construir nuestra propia placa?

En realidad, podemos crear un clon compatible adaptando su tamaño y su forma a nuestras necesidades particulares. Aunque esto pueda parecer una tarea enorme, solo necesitamos contar con los componentes adecuados y seguir las instrucciones que se encuentran disponibles en el sitio web oficial.



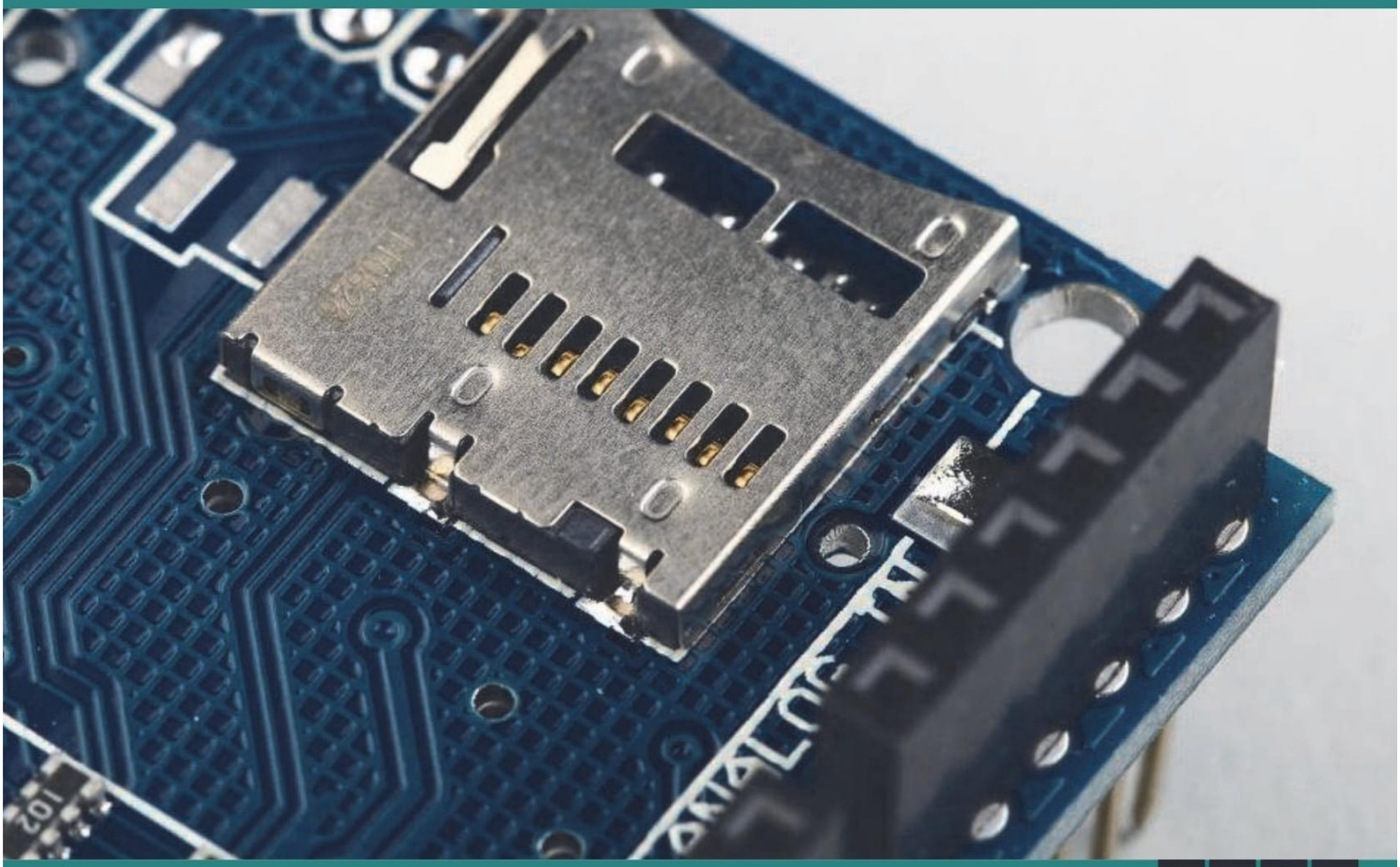
La construcción de un clon de Arduino es un proyecto realmente interesante. En el sitio **www.instructables.com/id/Palm-Arduino-Kit**, encontraremos las instrucciones detalladas para lograrlo.

Resumen Capítulo 11

En este capítulo hemos dado un paseo a través de las diversas aplicaciones en las que puede utilizarse una placa Arduino. Vimos que el mundo de Arduino es realmente más amplio de lo que pensábamos y que los proyectos que realizamos a través de los capítulos de este libro solo son un primer paso, ya que podemos lograr lo que nuestra creatividad sea capaz de vislumbrar. Entre los proyectos que conocimos en este capítulo se encuentran algunos relacionados con la domótica, la robótica y los drones. Además, hemos revisado la posibilidad de crear nuestra propia placa Arduino.

Ap

Shields



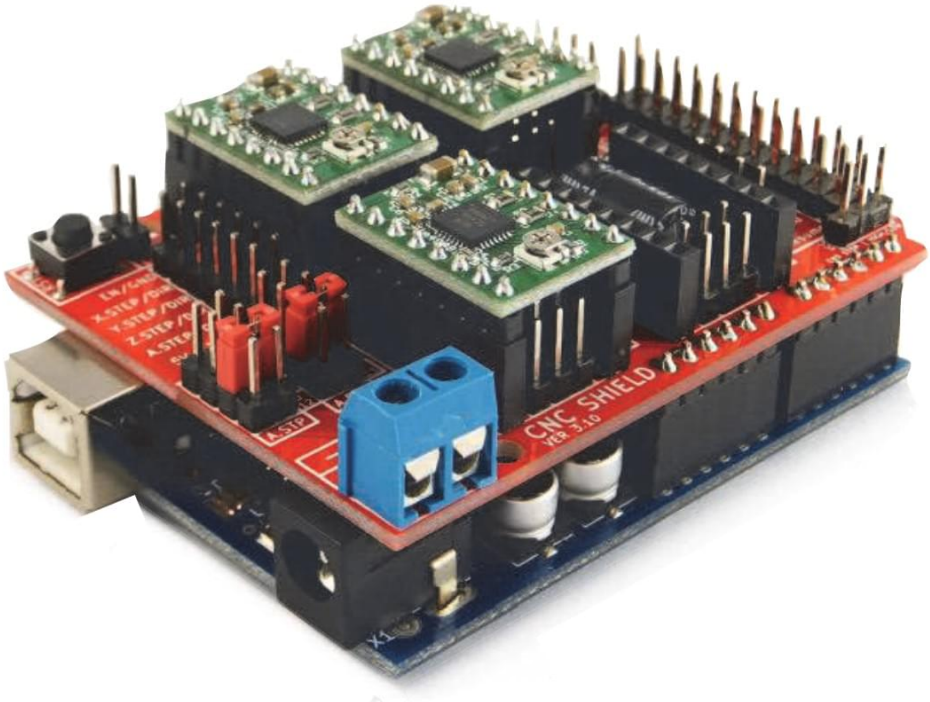
En este apéndice, conoceremos algunos elementos apropiados para dotar de mayores capacidades a nuestros proyectos con Arduino, se trata de las placas shield. Aquí veremos para qué sirven y cuáles son las opciones disponibles.

QUÉ ES UNA SHIELD

En forma sencilla, una **shield** no es más que una placa electrónica que puede ser conectada encima de la placa Arduino, para extender sus capacidades. Podemos pensar en una shield como en un dispositivo externo que conectamos a una computadora y que nos permite realizar funciones adicionales, por ejemplo, una impresora, un escáner, un teclado, un mouse, etcétera.

Una de las principales características de las shields, al igual que de las placas Arduino, es que se trata de elementos de hardware fáciles de conectar y con precios accesibles, de esta forma nos permiten trabajar en la realización de nuestros prototipos electrónicos de forma sencilla, rápida y económica. Pero, sin duda, lo más interesante de una shield es que proporciona funciones adicionales a una placa Arduino, y otorga nuevas posibilidades sin necesidad de adquirir otra placa.

La conexión de una shield es sencilla, pues se trata de elementos modulares que siguen la filosofía de Arduino: accesibilidad y facilidad de uso.



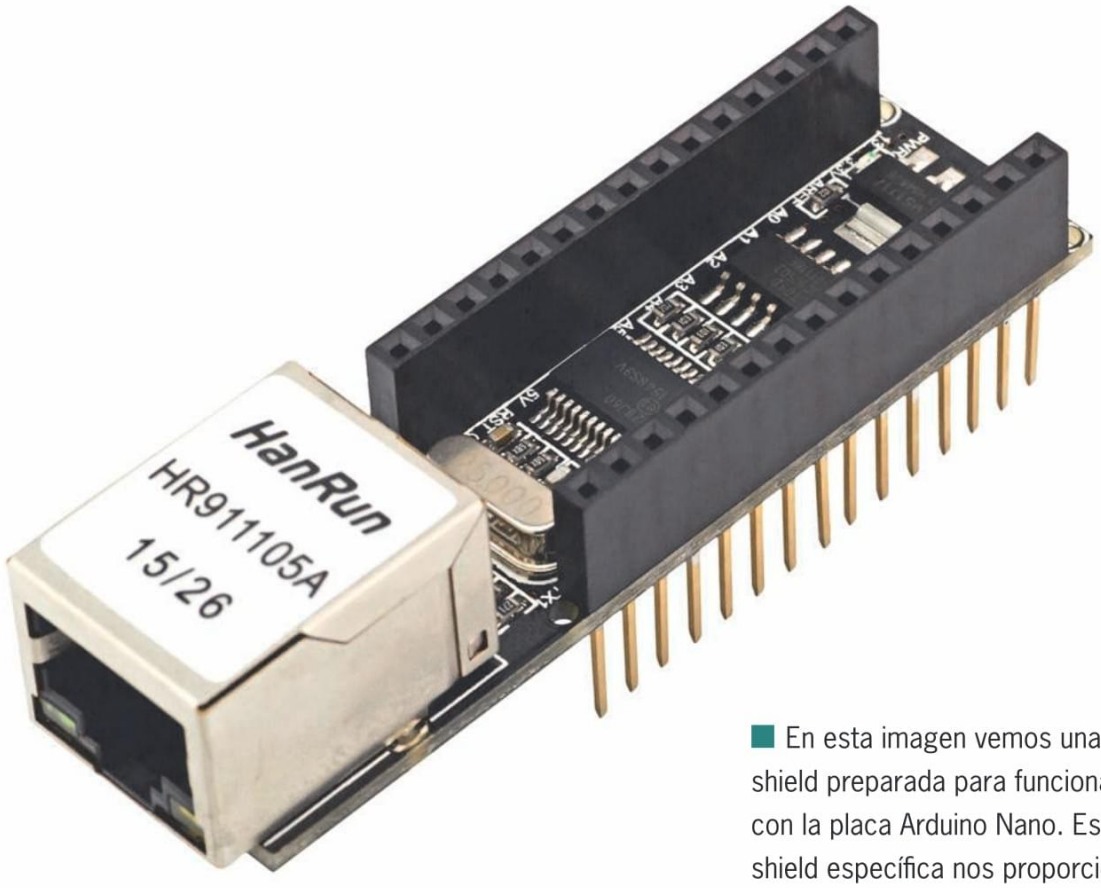
Las shields para Arduino son modulares y apilables, por lo que es posible utilizar varias de ellas en un solo proyecto, conectadas a una placa Arduino de base.

Características generales

Gracias a su modularidad, la conexión de una shield es una tarea sencilla, pues solo debemos montarla sobre la placa Arduino. Es posible montar varias shields, unas sobre otras, para obtener novedosas aplicaciones y muchas características adicionales.

Al ser apilables, una shield se comunica utilizando los pines digitales o analógicos de la placa Arduino, aunque también pueden utilizar el bus SPI o I2C, el puerto serie, o los pines de interrupción. Para alimentarse, por lo general, utilizará los pines de 5V y GND.

Las shields oficiales presentan un factor de forma que sigue las especificaciones de la placa Arduino; así, tendrán los pines con un espaciado definido y encontraremos solo una forma de conectarlo. Esto dará como resultado que la conexión y el uso de una shield serán sencillos, y se minimizarán los errores relacionados con una conexión realizada en forma incorrecta.



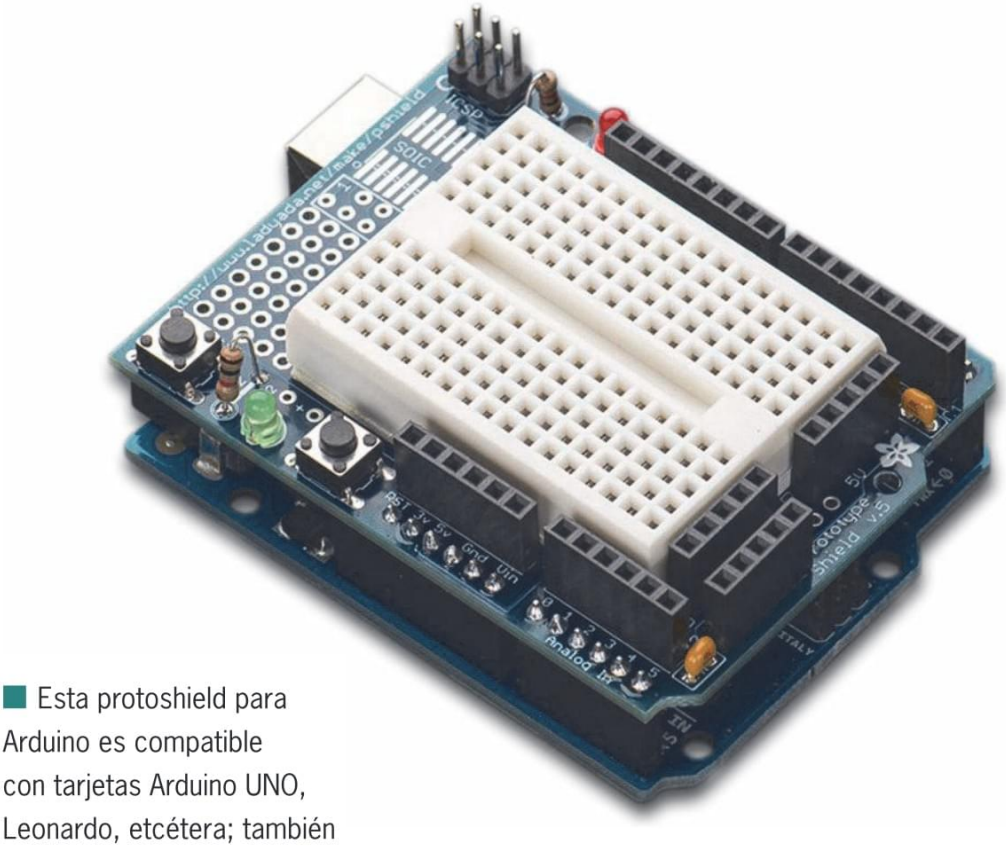
En esta imagen vemos una shield preparada para funcionar con la placa Arduino Nano. Esta shield específica nos proporciona una conexión Ethernet.

"Antes de elegir una shield específica debemos comprobar su compatibilidad."

No todas las shields son iguales, tampoco son compatibles con todas las tarjetas Arduino, por esta razón será necesario que leamos detenidamente la documentación de cada shield antes de adquirirla o de utilizarla en un proyecto específico.

Además de esto es importante tener en cuenta que algunos modelos presentan pines que no permiten la conexión de más shields por encima de ellos, estos tipos de shields deben utilizarse como elementos terminales.

La gran cantidad de placas Arduino existentes y también los diversos fabricantes de shields hacen que la tarea de clasificarlas y enumerarlas sea algo difícil. A continuación, presentaremos una **Tabla** que reúne ejemplos de shields para Arduino junto a sus respectivas descripciones.



Esta protoshield para Arduino es compatible con tarjetas Arduino UNO, Leonardo, etcétera; también puede ser utilizada con Arduino MEGA2560.

SHIELDS PARA ARDUINO

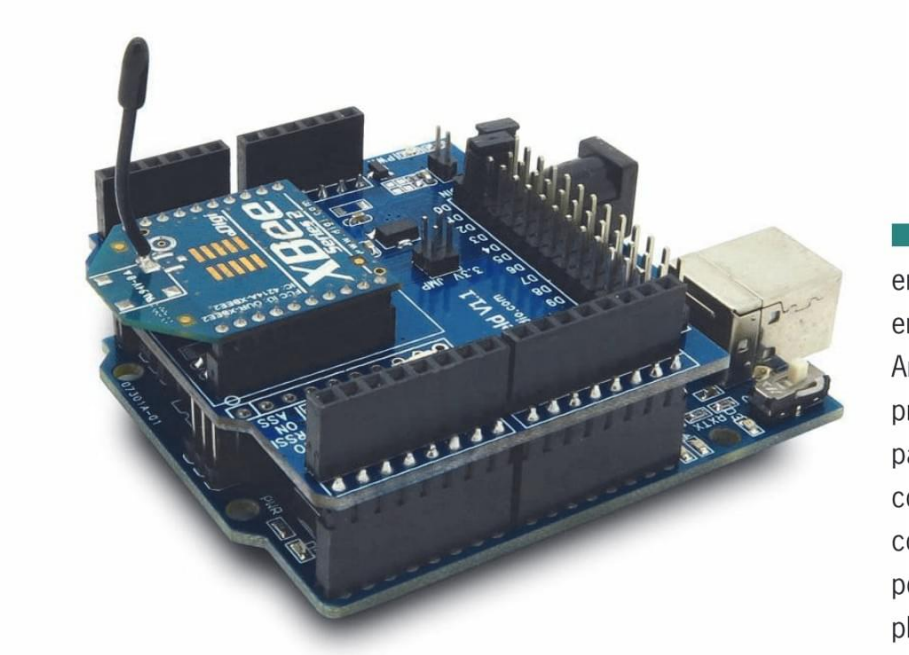
SHIELDS	DESCRIPCIÓN
Arduino Motor Shield	Se trata de una shield diseñada por el equipo de Arduino, por lo que la compatibilidad está garantizada. Se basa en el circuito L298, permite manejar cargas inductivas, como relevadores, solenoides, motores de DC y motores a pasos, por lo tanto, nos permitirá controlar la velocidad y la dirección de hasta dos motores de corriente directa o un motor a pasos.
Ethernet Shield	Esta shield es adecuada para trabajar en proyectos de domótica, como control de acceso, redes de sensores, dataloggers, conexión con servicios en la nube (web services), etcétera, así como también en aplicaciones que requieren la implementación de protocolos TCP/IP.
Monster Motor Shield VNH2SP30	Gracias a esta shield, podremos controlar motores de corriente directa que requieren hasta 15 amperes. Esta shield está pensada para funcionar en aplicaciones automotrices, es muy confiable para el manejo de motores en ambientes difíciles.
Shield con pantalla LCD	Nos ofrece una pantalla LCD táctil de 2.4 pulgadas. Se trata de una shield adecuada para proyectos con Arduino DUE o Mega. La pantalla incluida posee una resolución de 320×240 pixeles.
Shield GPS para Arduino	Esta shield proporciona la capacidad de recibir señales de GPS y almacenar las coordenadas recibidas en una tarjeta micro SD. Se basa en un módulo EB-365, y es compatible con Arduino UNO y Arduino Mega.
Shield XBee	Se trata de una shield que ofrece capacidades inalámbricas a nuestros proyectos. Gracias a XBee, es posible conectar los módulos de radiofrecuencia compatibles con XBee, y podremos seleccionar los pines utilizados para comunicarse con los módulos de radio.

Ejemplos de shields para Arduino junto a sus principales características.

La oferta de shields es variada y está muy lejos de ser estática, por lo tanto, cada cierto tiempo aparecerán nuevas propuestas o versiones renovadas de las shields que ya conocemos, esto nos permitirá avanzar cada vez más en la realización de nuestros proyectos.

Conexión de una shield

La conexión de una shield deberá hacerse con la placa Arduino apagada, porque, al conectarla bajo tensión, podemos obtener consecuencias no deseadas. Es necesario que los pines de la shield encajen suavemente; después, presionaremos en forma cuidadosa hasta que la conexión se realice por completo.



La shield debe encajar correctamente en los pines de la placa Arduino, es necesario presionar con cuidado para lograr una conexión correcta. Luego de conectar la shield, podemos encender la placa Arduino.

Listado de Shields

Como sabemos, existe una enorme cantidad de shields disponibles en el mercado. Cada fabricante nos propone un listado propio de shields que cubren diferentes necesidades, y son aplicables a distintas placas y proyectos. Para tener información sobre una gran cantidad de shields, podemos visitar el sitio web **http://shieldlist.org**, donde encontraremos un listado de fabricantes; al seleccionarlos, veremos una lista de shields y sus principales características.

SHIELDS DISPONIBLES

Ya sabemos que existen muchas shields para Arduino, algunas se califican como oficiales, y otras se agrupan como no oficiales; podemos utilizar cualquiera de ellas, aunque debemos tener en cuenta que los elementos oficiales nos garantizan la compatibilidad con ciertas tarjetas Arduino.

A lo largo de los capítulos que componen este libro, hemos aprendido a visualizar las posibilidades que nos ofrece una placa Arduino, las opciones son muchas, pero, al integrar una shield, el potencial aumenta considerablemente. Existen muchas shields, a continuación conoceremos algunas de las más importantes y sus principales características.

Ethernet Shield

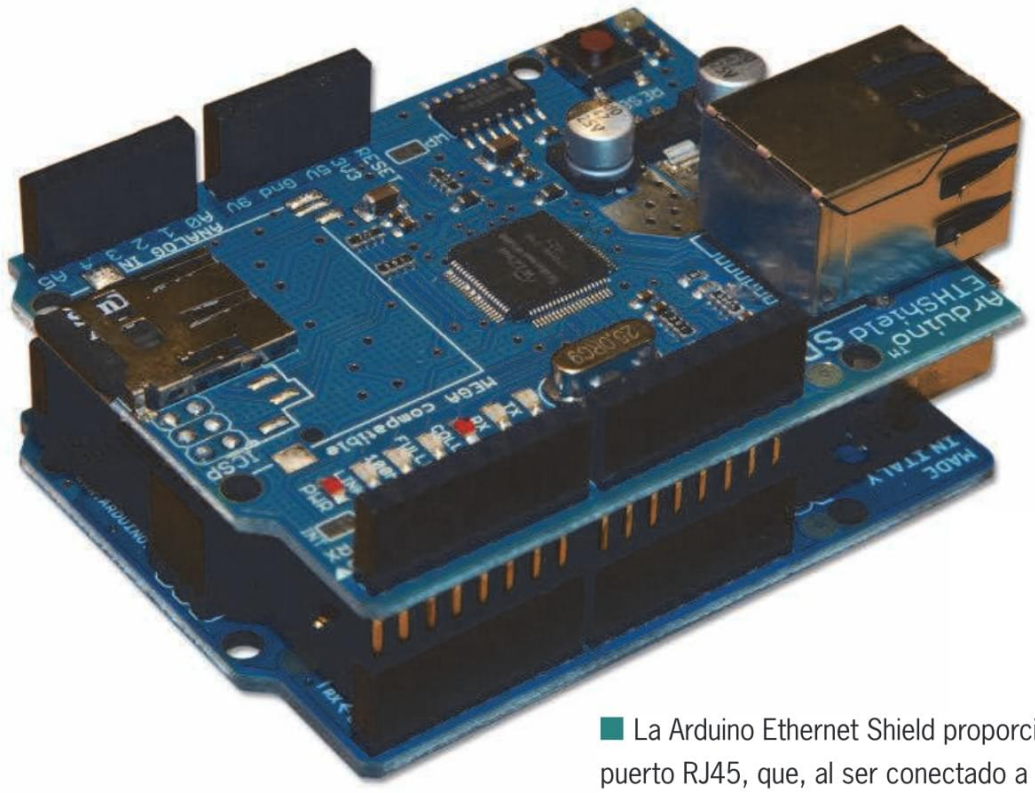
La Ethernet Shield proporciona acceso a la red local, por lo tanto, puede ser utilizada en diversos proyectos.

La conexión de una Ethernet Shield solo requiere encastrar cuidadosamente los pines adecuados sobre la placa Arduino, luego de esto podemos conectar la alimentación a dicha placa mediante un cable USB a la computadora y, también, conectar un cable de red a la shield.

Para conectar la shield a la red, es necesario contar con un cable de red RJ45, cuyos extremos deben conectarse al puerto adecuado de la shield y al router o a un punto de red, respectivamente. Una vez que la conexión se ha realizado, las posibilidades de Arduino aumentan, por ejemplo, es posible conectarse a internet o crear un servidor web con Arduino.

Comprobar que la conexión de la Ethernet Shield a la red se ha realizado en forma correcta es sencillo, pues generalmente junto al conector RJ45, incluye un LED que se encenderá cuando la conexión a la red se concrete. Asimismo, podemos verificar que se enciende la luz que corresponde al puerto conectado a la shield en el panel frontal del router.

"El shield Ethernet es una de las más utilizadas, nos otorga acceso a la red local por lo que abre interesantes posibilidades."



La Arduino Ethernet Shield proporciona un puerto RJ45, que, al ser conectado a una placa Arduino, permite unirla a una red para acceder a internet, crear un servidor web o realizar tareas relacionadas con domótica y redes.

Lo primero que debemos hacer es conseguir una dirección IP válida, y para ello podemos utilizar algunas líneas de código que se integran en los ejemplos del Arduino IDE. Para importar las librerías necesarias escribimos las siguientes líneas:

```
  #include<SPI.h>
  #include<Ethernet.h
```

Para definir una dirección MAC usamos la línea siguiente:

```
  byte mac[] = {  0x00, 0xAA, 0xBB, 0xCC, 0xDE, 0x02
} ;
```

Continuamos con la creación de una instancia del cliente Ethernet y también iniciamos el puerto serie. Debemos considerar que, al no

entregar una dirección IP, se intentará conseguir mediante DHCP en forma automática. Obtendremos **1** si se consigue la dirección y **0** en caso de que la dirección no se consiga.

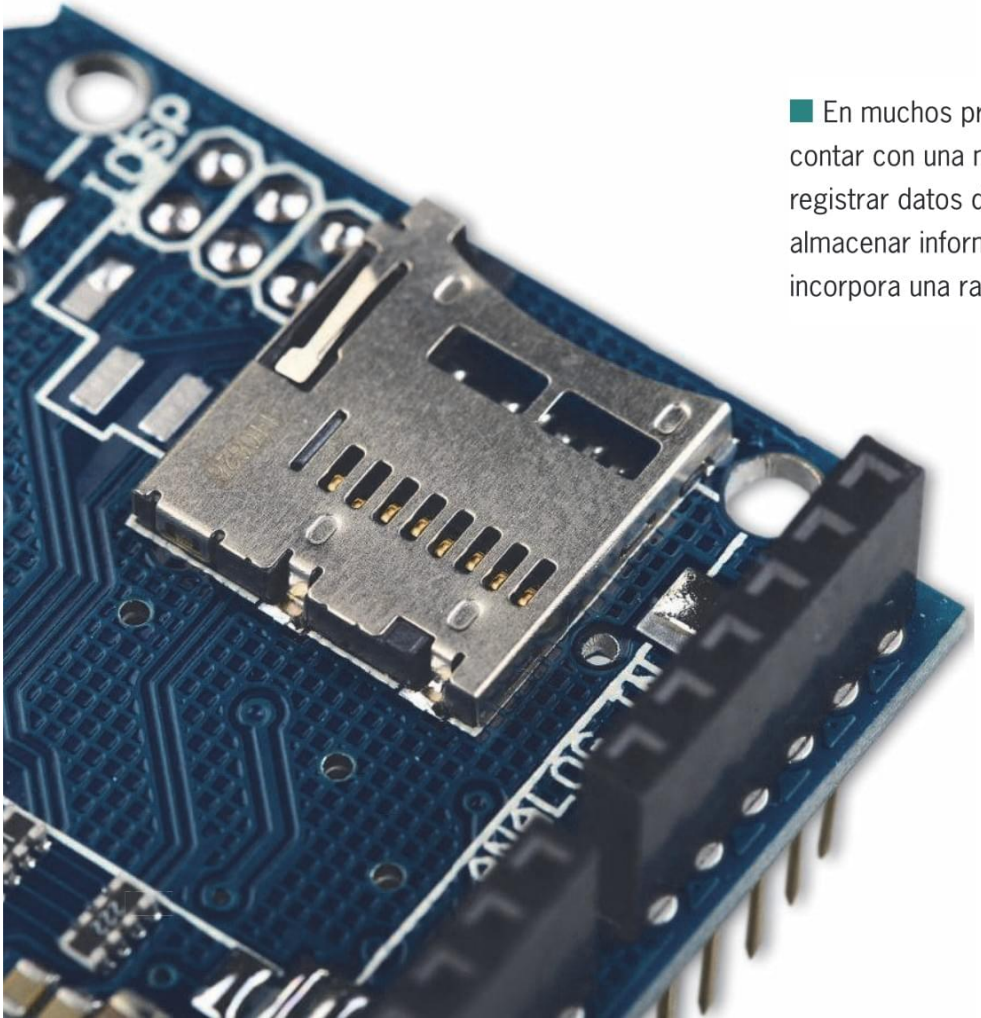
```
  if (Ethernet.begin(mac) == 0)
     {
  Serial.println("Error en configuración DHCP");
  while (true; )     ;
     }
```

Si se devuelve **1**, se rellenará un array **Ethernet.localIP()** con la dirección IP obtenida mediante DHCP. A continuación, vemos el código completo de un sketch, que permite efectuar una conexión, negociar una dirección IP y, posteriormente, mostrar la dirección IP obtenida por el puerto serie:

```
  #include<SPI.h>
  #include<Ethernet.h>

  byte mac[] = {  0x00, 0xAA, 0xBB, 0xCC, 0xDE,
0x02};
  EthernetClientclient;

  voidsetup()
    {  Serial.begin(9600);
  while (!Serial) { ; }
  Serial.println("Buscando DHCP...");

  if (Ethernet.begin(mac) == 0)
          {     Serial.println("Falla en la configura-
ción DHCP");
  for (;;)    ;
          }
  Serial.print("IP : ");
  for (byte B = 0; B < 4; B++) {
  Serial.print(Ethernet.localIP()[B], DEC);
  Serial.print(".");
```

```
  }
Serial.println();

Serial.print("IP Router: ");
for (byte B = 0; B < 4; B++) {
Serial.print(Ethernet.gatewayIP()[B], DEC);
Serial.print(".");
  }
Serial.println();
}

voidloop() {

}
```

Hasta aquí utilizamos la shield solo para conectarnos a una red local y negociar una dirección IP mediante DHCP, la que después mostramos mediante el puerto serie.



En muchos proyectos será interesante contar con una memoria micro SD. Para registrar datos desde sensores o para almacenar información, la Ethernet Shield incorpora una ranura para micro SD.

Otra de las posibilidades de la Ethernet Shield es acceder a internet y realizar navegación o consultas específicas, por ejemplo, gracias a un sketch incluido en el IDE de Arduino es posible buscar en Google y mostrar el resultado mediante el puerto serie.

Debemos tener en cuenta que esta tarea genera mucha información y llegará en formato HTML, por lo que será necesario copiarla, pegarla en un editor de texto plano asignándole la extensión .html y, luego, abrirla con un navegador web para ver su contenido. El código de ejemplo de **WebClient** es el siguiente:

```
  #include<SPI.h>
  #include<Ethernet.h>

  byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED
};
  //IPAddress server(74,125,232,128);
  char server[] = "www.google.com";

  IPAddressip(192, 168, 1, 177);
  EthernetClientclient ;

  voidsetup()
     {     Serial.begin(115200);
  while (!Serial) ;

  if (Ethernet.begin(mac) == 0)
             { Serial.println("Failedto configure Eth-
ernet using DHCP");
  Ethernet.begin(mac, ip);
             }
  delay(1000);
  Serial.println("connecting...");
  if (client.connect(server, 80))
              { Serial.println("connected");
  client.println("GET /search?q=arduino HTTP/1.1");
  client.println("Host: www.google.com");
  client.println("Connection: close");
```

```
client.println();
            }
else
Serial.println("connectionfailed");
   }
voidloop()
   {    if (client.available())
           {   char c = client.read();
Serial.print(c);
           }
if (!client.connected())
          {  Serial.println();
Serial.println("disconnecting.");
client.stop();

while (true) ;
          }
   }
```

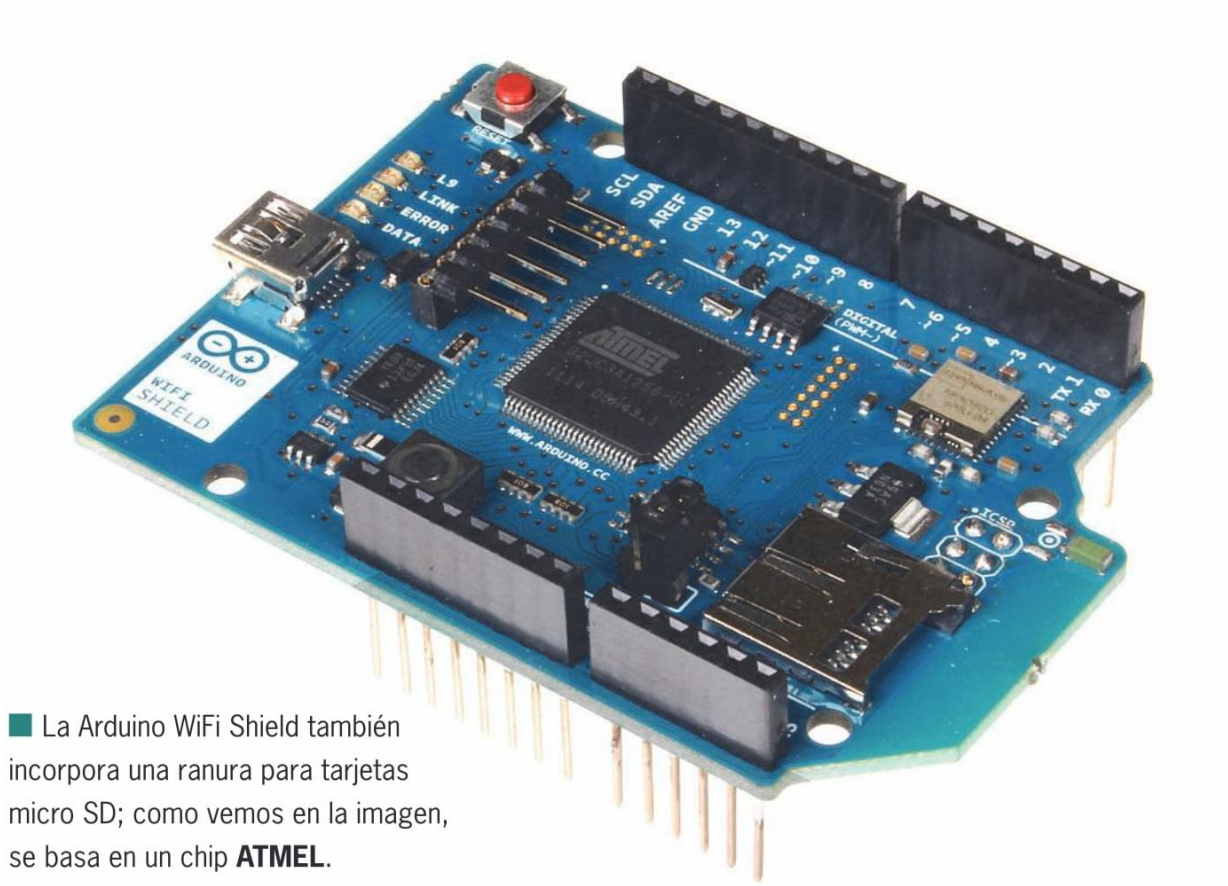
Arduino WiFi Shield

Una shield WiFi proporciona conectividad inalámbrica a nuestra placa Arduino, por lo tanto, nos permite conectarnos a internet en forma inalámbrica. Su función es similar a lo que revisamos sobre Ethernet Shield, pero en este caso se utiliza la infraestructura de comunicaciones WiFi.

Existen diversas shields con capacidad WiFi, por ejemplo, la **Arduino WiFi Shield** o también la **WiFi Shield CC3000**.

La primera de ellas, nos permite conectar una placa Arduino a internet haciendo uso de WiFi y de la WiFi Library; también posee un slot para una tarjeta micro SD. Para comunicarse con Arduino, utiliza el bus SPI mediante los pines ICSP, usa los pines 4 y 10 de igual forma que la Ethernet Shield. Debemos tener en cuenta que también utilizará el pin 7, por lo que no podremos contar con él para nuestros proyectos.

Otro punto que debe considerarse es que las tarjetas micro SD y WiFi usarán el bus SPI, por lo tanto, no es posible que trabajen en forma simultánea.

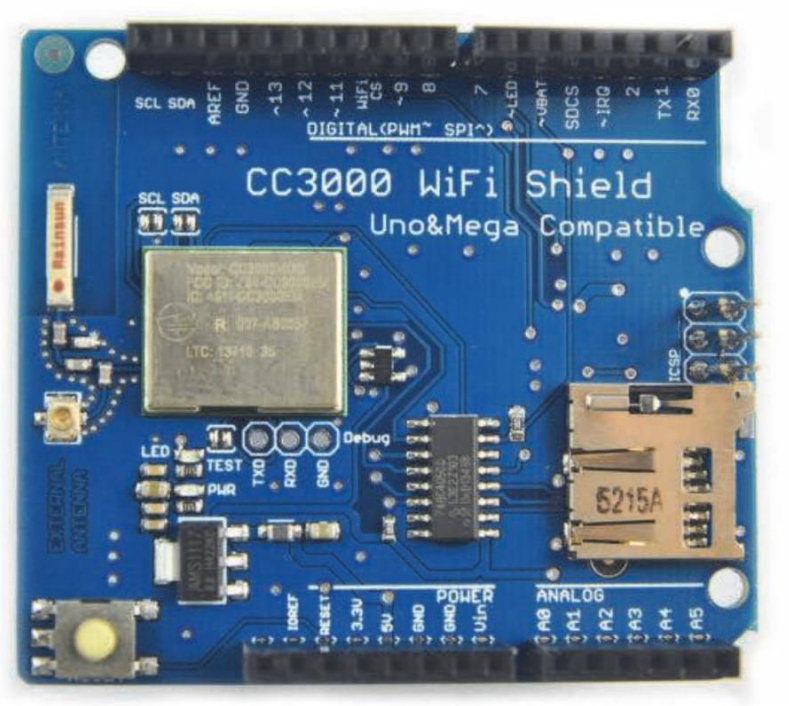


La Arduino WiFi Shield también incorpora una ranura para tarjetas micro SD; como vemos en la imagen, se basa en un chip **ATMEL**.

Otra opción que fue bastante usada es la WiFi Shield que se fundamenta en el chip CC3000 de Texas Instruments. Este chip está en la base de muchas aplicaciones de IOT o *Internet Of Things* y resuelve las comunicaciones WiFi en forma accesible. Por esta razón, es el pilar de algunas shields que buscan ofrecer conectividad inalámbrica sin que sea necesario desembolsar las sumas de dinero que se asocian a la shield WiFi oficial de Arduino.

Existen diversas shields WiFi que basan su funcionamiento en el chip CC3000, por ejemplo, variantes de las placas de **Adafruit**, que también incorporan un lector de tarjetas micro SD y, además, permiten trabajar con las librerías CC3000.

"Gracias a la biblioteca WiFi es posible escribir programas que conecten Arduino a Internet, mediante la WiFi Shield"



En esta imagen vemos una shield WiFi que se basa en el chip CC3000.

En términos generales, el chip CC3000 usa SPI para comunicarse con el exterior, por esta razón, la velocidad puede ser alterada dependiendo de las necesidades. Soporta 802.11b/g, Open/WEP/WPA/WPA2 Security, TKIP, y también AES. Incorpora un stack TCP/IP completo y permite efectuar un máximo de 4 conexiones concurrentes.

Para utilizar la WiFi Shield CC3000, será necesario descargar la librería adecuada, podemos buscarla bajo el nombre **Adafruit_CC3000_Library-master**. Una vez que hayamos instalado la librería, comenzaremos a trabajar con los primeros sketchs. Aunque existen muchos procedimientos y métodos internos que son nuevos, podemos iniciar de la siguiente forma:

```
#include<Adafruit_CC3000.h>
#include<SPI.h>

#define ADAFRUIT_CC3000_IRQ   3
#define ADAFRUIT_CC3000_VBAT  5
#define ADAFRUIT_CC3000_CS    10
```

Este código incluye las librerías que necesitamos y, también, se encarga de establecer los pines de control.

Otro punto importante es definir los datos que utilizaremos para efectuar una conexión mediante WiFi:

```
#define WLAN_SSID       "mi-red"
#define WLAN_PASS       "password"
#define WLAN_SECURITY   WLAN_SEC_WPA2
```

Para inicializar el chip WiFi, utilizamos **cc3000.begin()**, también es necesario comprobar que esta tarea ha resultado correcta:

```
Serial.begin(115200);
Serial.println("Buscando chip CC3000!");
Serial.print("Inicializando chip CC3000 ...");
if (!cc3000.begin())
   { Serial.println("\nError al inicializar" );
Serial.flush();
exit(0);
   }
```

Es recomendable limpiar los datos que corresponden a conexiones anteriores, podemos hacerlo de la siguiente forma:

```
Serial.println("Eliminar datos de conexión ");
cc3000.deleteProfiles();
```

Modulo WiFi

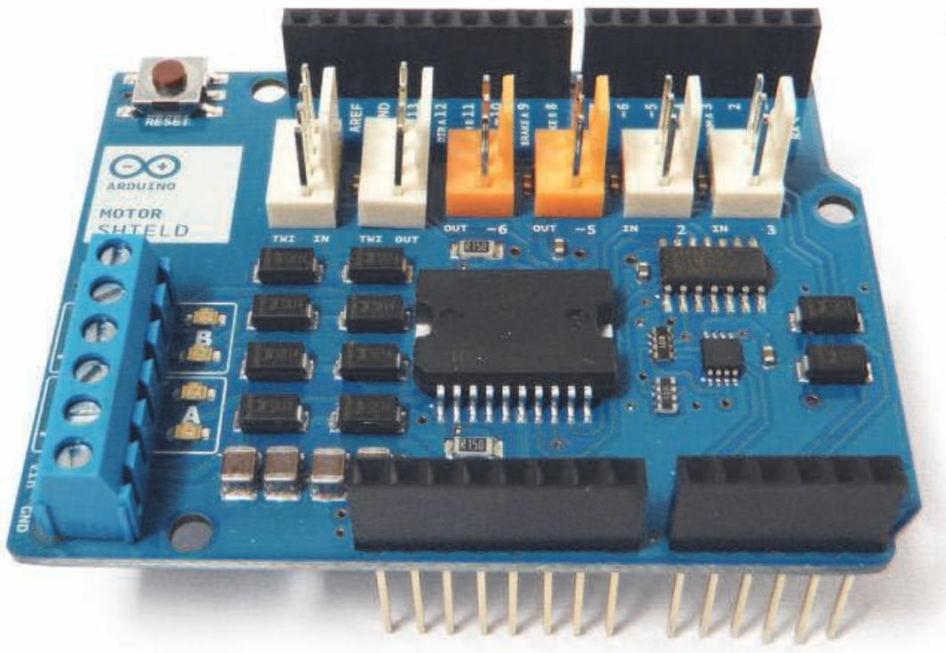
Aunque las shields WiFi nos proporcionan una forma eficiente de otorgar conectividad WiFi a nuestra placa Arduino, su principal dificultad es que los valores que encontramos en el mercado no resultan muy accesibles. Podemos buscar tanto una shield no oficial como algunas que se basan en un chip CC3000, o también decidir que utilizaremos un módulo WiFi. Una de estas opciones es el módulo **WiFi ESP8266**, que incluye todo lo que necesitamos para lograr una comunicación en la banda WiFi. De esta forma tenemos, en un pequeño componente de bajo costo, un procesador y conectividad WiFi.

Para conectar, utilizamos la función **cc3000.connectToAP**, con los datos de conexión que especificamos antes:

```
  char *ssid = WLAN_SSID;
  Serial.println("Conectando a "); Serial.
print(ssid);
  if (!cc3000.connectToAP(WLAN_SSID, WLAN_PASS, WLAN_
SECURITY))
  {    Serial.println("Error al conectar..");
  Serial.flush(); exit(0);
  }
  Serial.print("Conectado, esperando DHCP");
```

Arduino Motor Shield

Esta shield oficial hace posible que la tarea de manejar dos motores DC, controlando su dirección y su velocidad, pueda realizarse en forma sencilla y sin demasiadas complicaciones. Su construcción se basa en la presencia de un chip de doble puente H ST L298.



La Arduino Motor Shield entrega la posibilidad de conectar y controlar motores en forma sencilla.

Como hemos aprendido, una de las bondades del hardware libre es que hace posible la aparición de alternativas y modificaciones, y esto se convierte en un punto a favor del entusiasta de la electrónica, pues no es necesario conformarse con una sola shield. El control de motores no es la excepción pues la shield oficial de Arduino no es la única disponible, existen muchas opciones en el mercado, por ejemplo la Motor Shield V1 de Adafruit, que nos permite trabajar con cuatro motores de corriente continua más dos servos o, también, dos motores paso a paso.

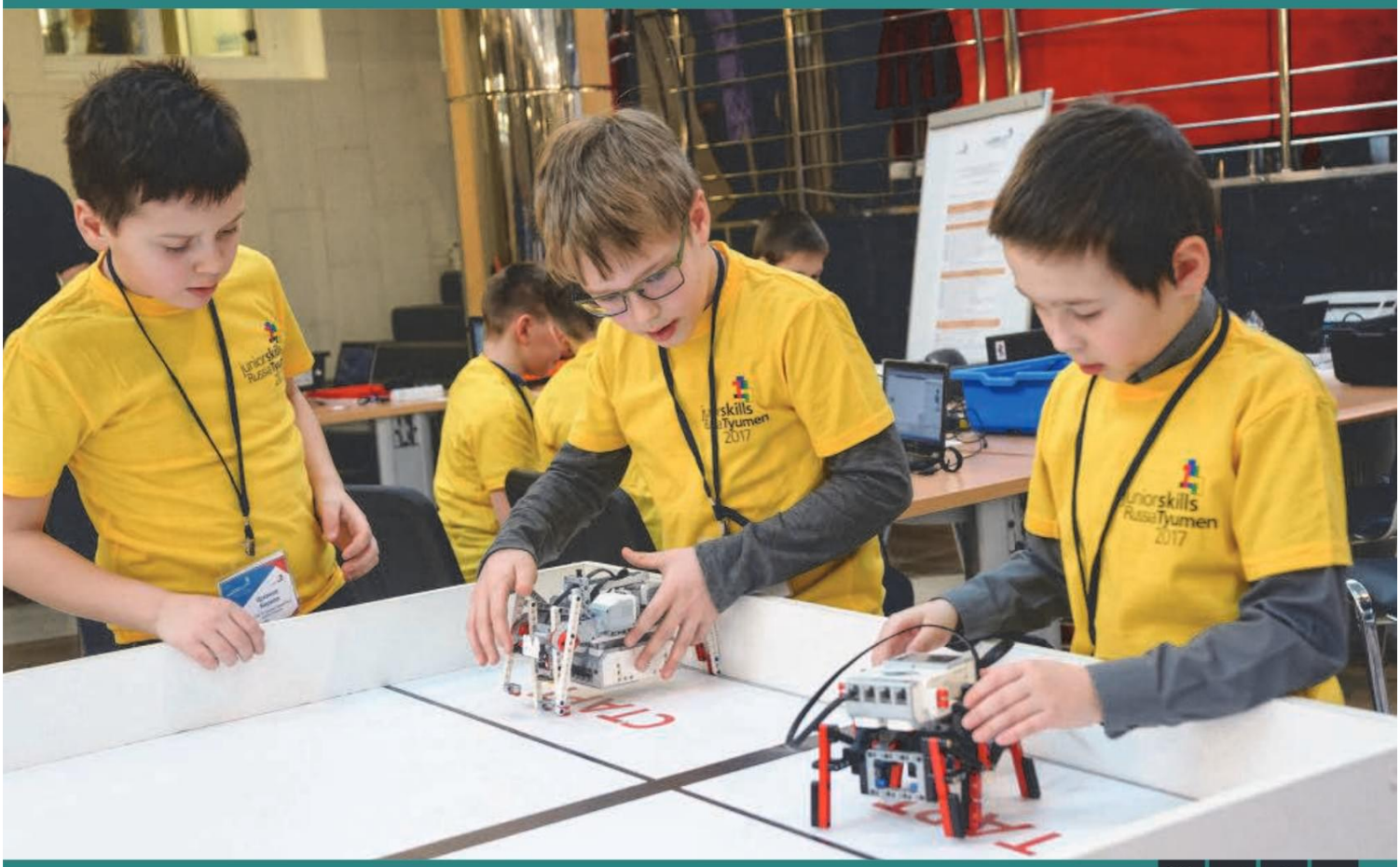
Si estamos trabajando con cargas pequeñas o recién empezamos a experimentar con la conexión de motores, el Motor Shield V1 es una excelente opción, pero, si necesitamos fuerza, podemos pensar en la versión V2.

Por otra parte, también está disponible **RAMPS**, se trata de una shield diseñada para la placa Arduino MEGA, que permite controlar cinco motores paso a paso; esta, sin duda, se trata de una opción más profesional, adecuada para casos específicos, por ejemplo, las impresoras 3D de RepRap.

Resumen Apéndice

En este apéndice hemos dado un paso más en la tarea de profundizar en el mundo de Arduino. Revisamos las shields y aprendimos que se trata de elementos modulares que se pueden conectar en forma sencilla a ciertas placas Arduino, otorgando nuevas funciones y características que las placas no ofrecen en forma predeterminada. Vimos los pasos necesarios para conectar una shield y también conocimos algunas shields específicas. Analizamos las características y algunos ejemplos de uso de Ethernet Shield y WiFi Shield, y caracterizamos a una Motor Shield.

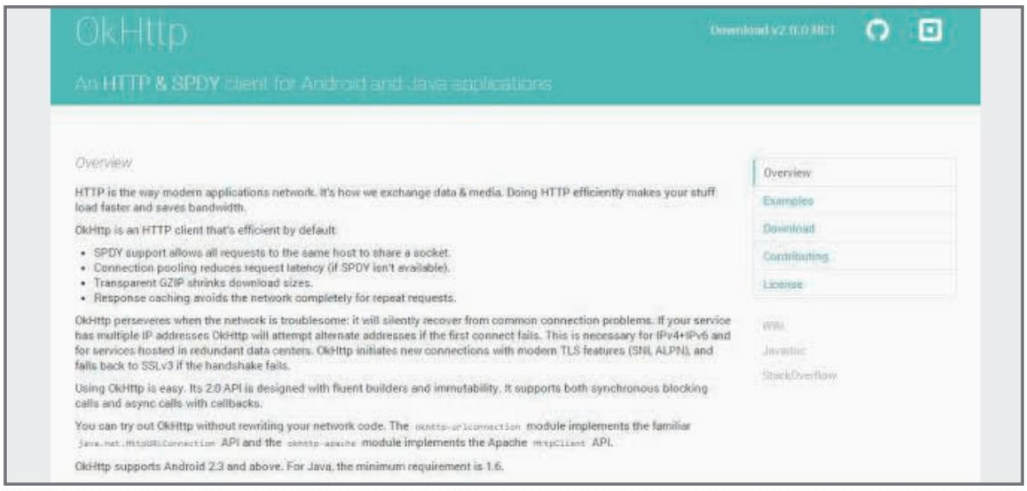
Servicios al lector



En este apéndice hemos incluido recursos para seguir aprendiendo; sitios web recomendados y programas relacionados con la obra.

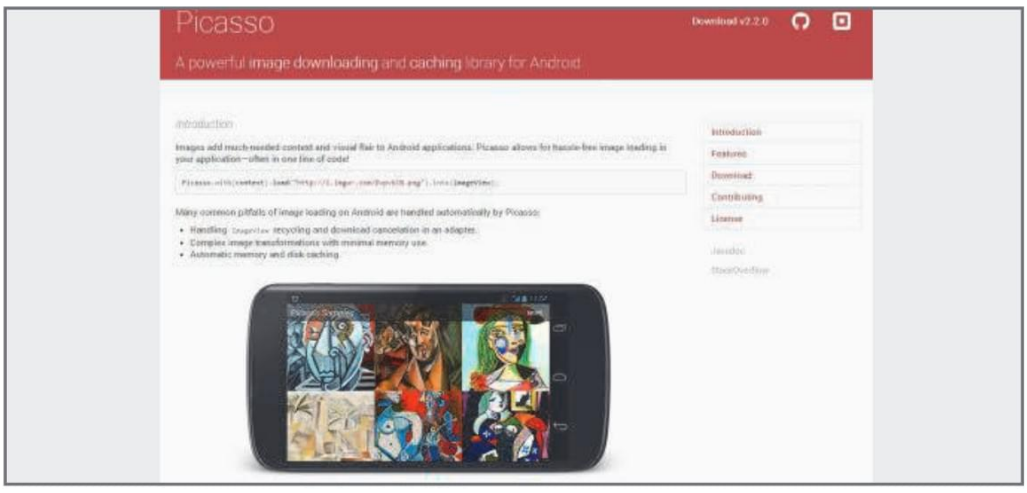
GUÍAS DE ARDUINO
https://playground.arduino.cc/Es/Guias

Completa colección de guías de inicio que nos permitirán trabajar con Arduino y realizar interesantes proyectos. Se trata de una wiki de recursos, por lo que puede ser ampliada por los usuarios.



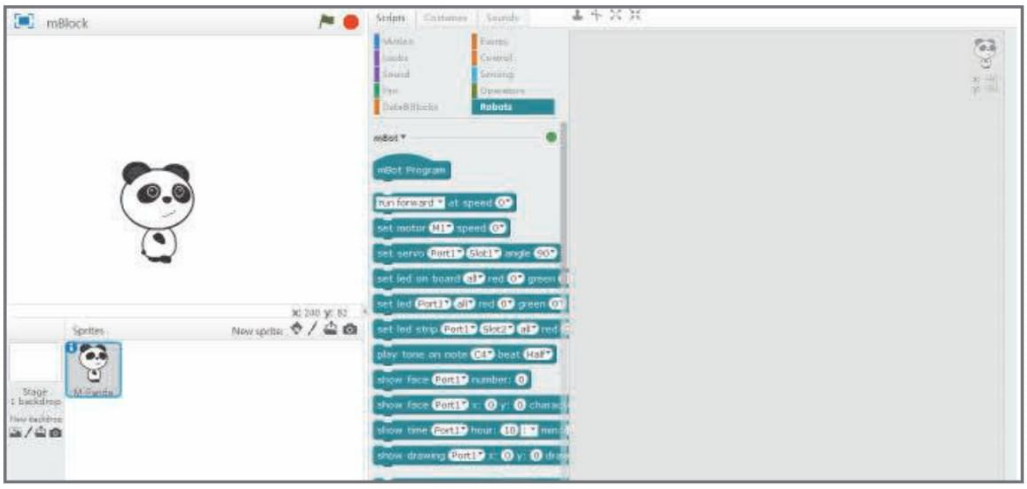
SCRATCH 4 ARDUINO
http://s4a.cat

Plataforma para programar Arduino mediante bloques, ofrece la posibilidad de admitir la conexión con una placa Arduino y añadir los bloques correspondientes que permiten interactuar con ella en forma sencilla.



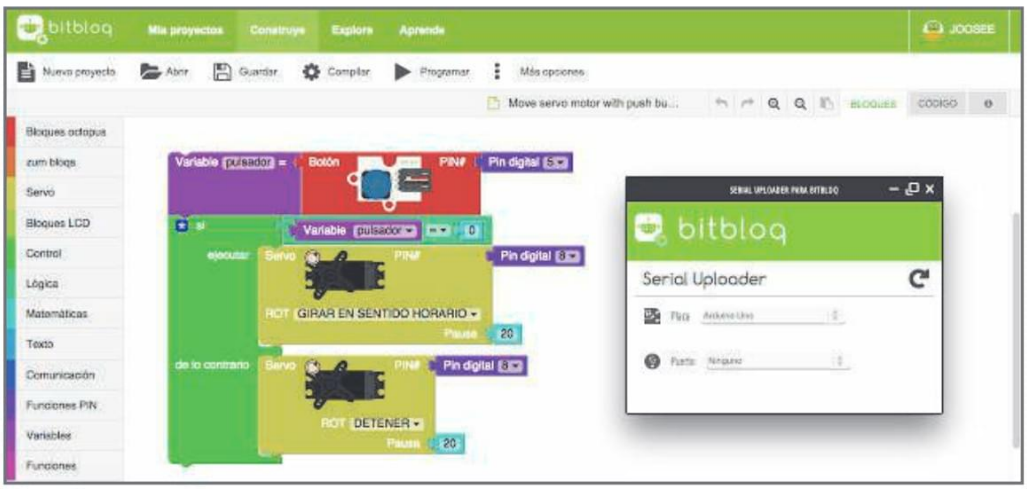
MBLOCK
http://www.mblock.cc

Plataforma de Makeblock, desarrollador del robot mBot, se trata de un scratch modificado, que permite programar sus dispositivos. Como los robots de esta empresa se basan en Arduino, esta plataforma es una excelente opción para programarlos.



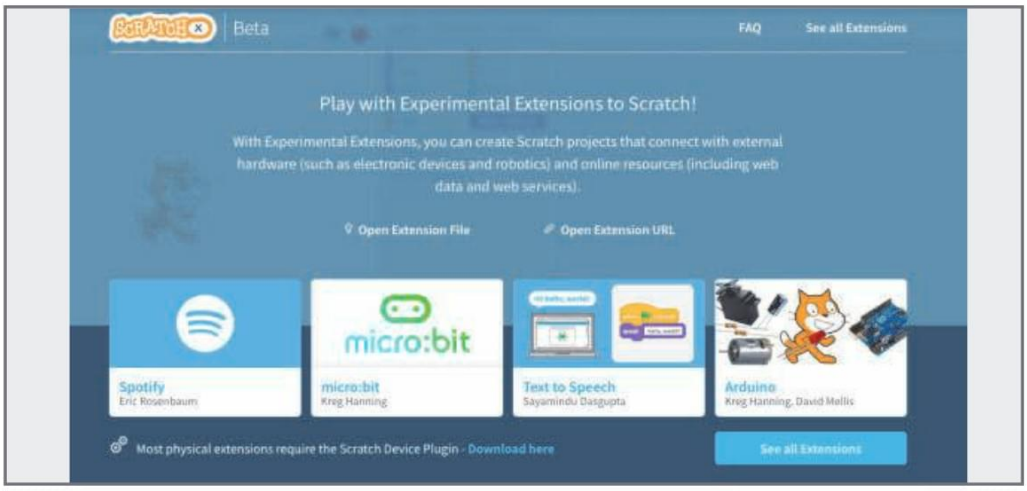
BITLOQ
http://bitbloq.bq.com/

Bitbloq es el software que permite programar y controlar los dispositivos que se encuentran basados en Arduino, enfocados tanto al hogar como al aula, comercializados por la empresa BQ.



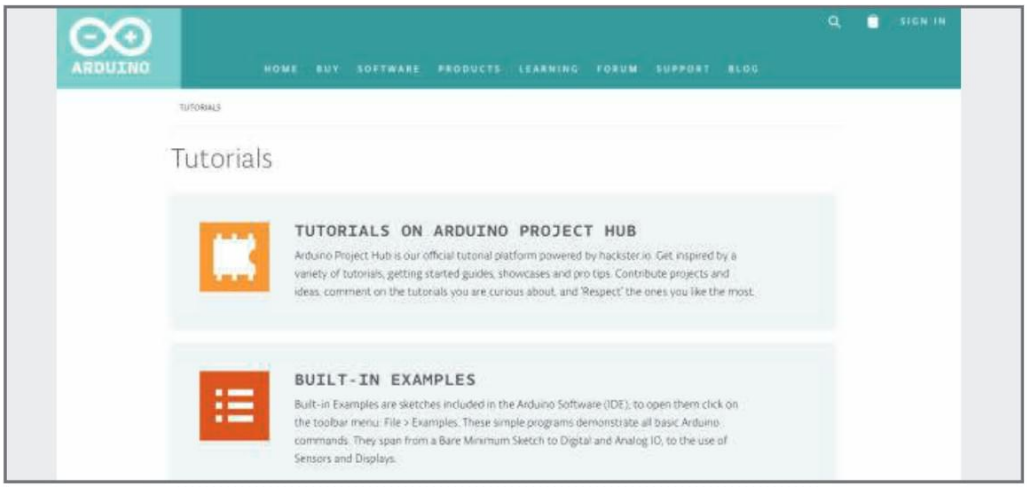
SCRATCHX
http://scratchx.org

Plataforma coordinada por el MIT que nos ofrece extensiones experimentales posibles de ser probadas sobre Scratch. ScratchX hace posible agregar código para programar sistemas dedicados a LEGO Mindstorms, LittleBits y Arduino.



TUTORIALES ARDUINO.CC
https://www.arduino.cc/en/Tutorial/HomePage

Sitio web que reúne una gran cantidad de tutoriales de la comunidad Arduino.cc.
En esta dirección encontraremos guías detalladas y, también, instrucciones paso a paso para generar diversos proyectos.



MINIBLOQ

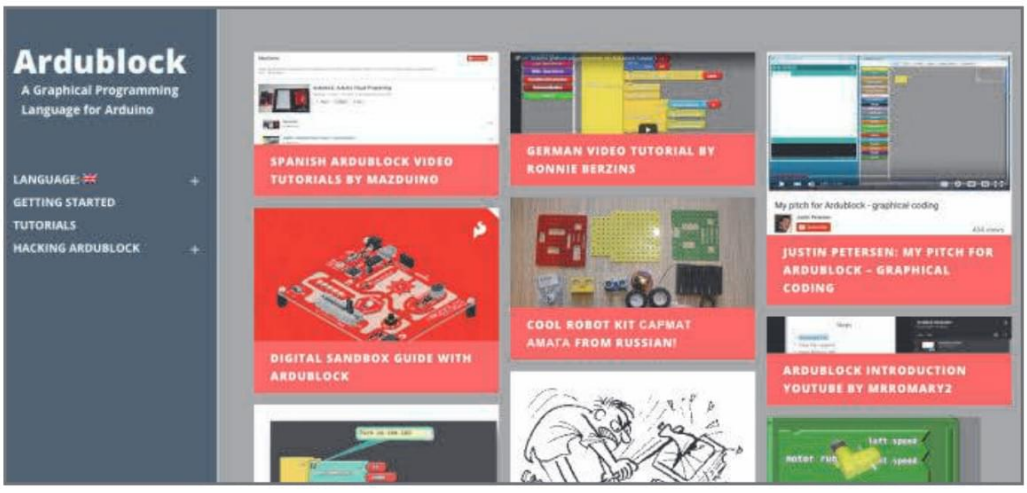
http://blog.minibloq.org/2016/10/root-un-robot-para-ensenar-programacion.html

Plataforma que nos permite crear programas mediante la mezcla de estos bloques gráficos con instrucciones de la programación tradicional. Se trata de una opción para usuarios con más experiencia.

ARDUBLOCK

http://blog.ardublock.com

Lenguaje de programación compatible con Arduino, nos permite crear programas con solo arrastrar y soltar bloques para programar una placa Arduino.

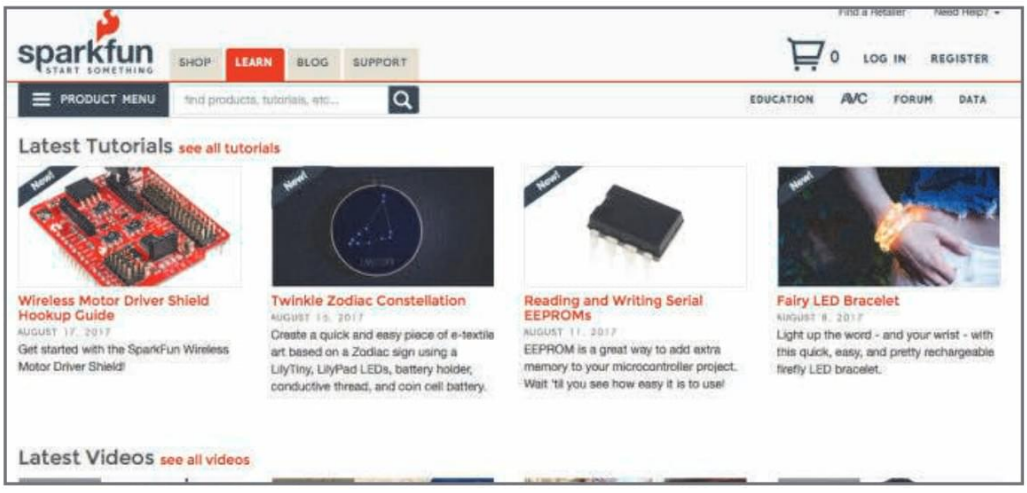


FOREFRONT
http://forefront.io/a/beginners-guide-to-arduino/

Guía de inicio para dar los primeros pasos en Arduino. Nos proporciona información relevante, y enseña cómo realizar unos sencillos programas iniciales en la IDE oficial.

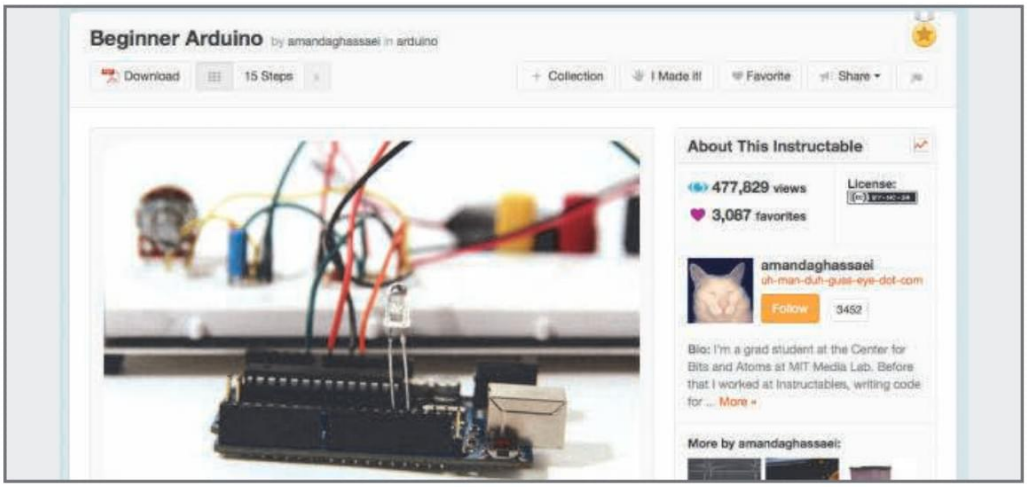
SPARKFUN
https://learn.sparkfun.com

Se trata de una tienda online de electrónica que también nos ofrece acceso a contenido interesante sobre el desarrollo Arduino. Su activa comunidad la ha convertido en una referencia en cuanto a los contenidos para fomentar el autoaprendizaje de Arduino.



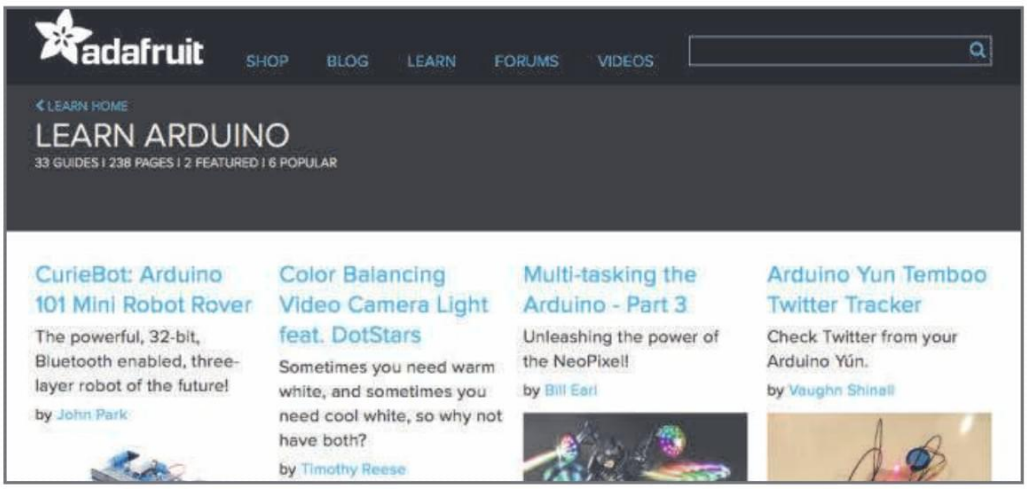
INSTRUCTABLES ARDUINO
http://www.instructables.com/id/Beginner-Arduino/

Instructables es una de las webs de referencia para makers. También posee una sección dedicada a Arduino, donde encontramos excelentes tutoriales para iniciarnos en esta plataforma de electrónica.



ADAFRUIT
https://learn.adafruit.com/category/learn-arduino

Adafruit es una de las tiendas de electrónica online más reconocidas de los últimos años, lo interesante es que también dedica esfuerzos a la formación. En su sitio web, encontramos diferentes cursos y tutoriales disponibles en forma gratuita.



INDICE TEMÁTICO

A

B

C

D

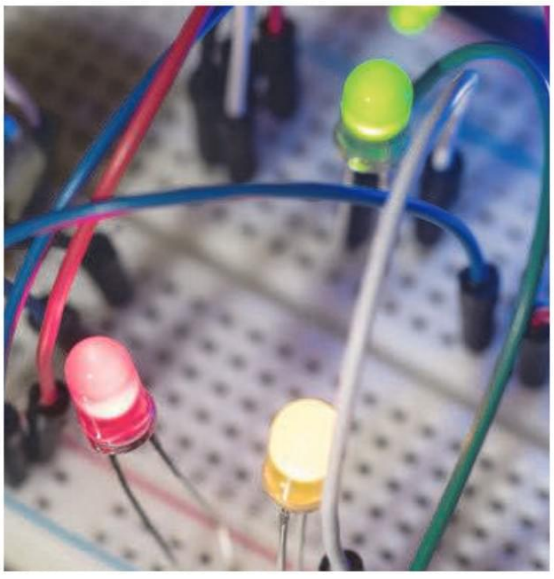
E

F

I

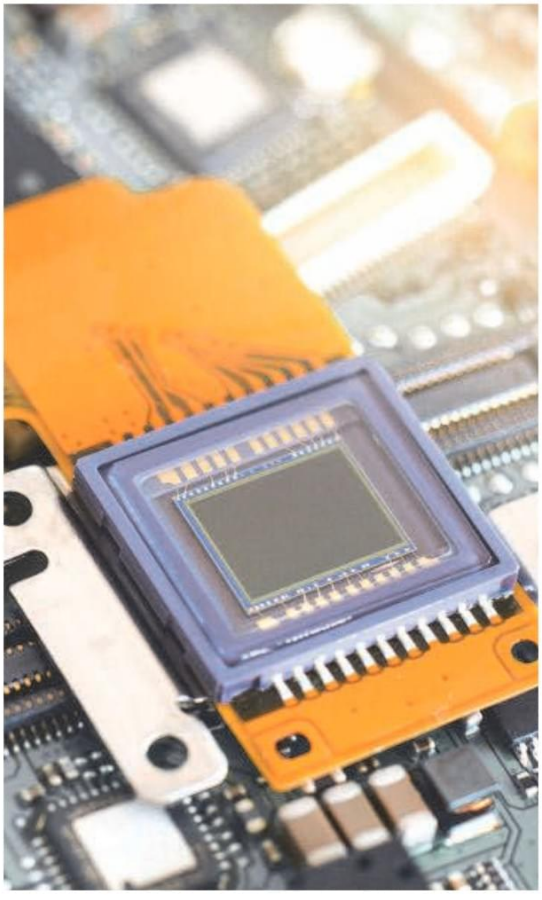
L

M



O

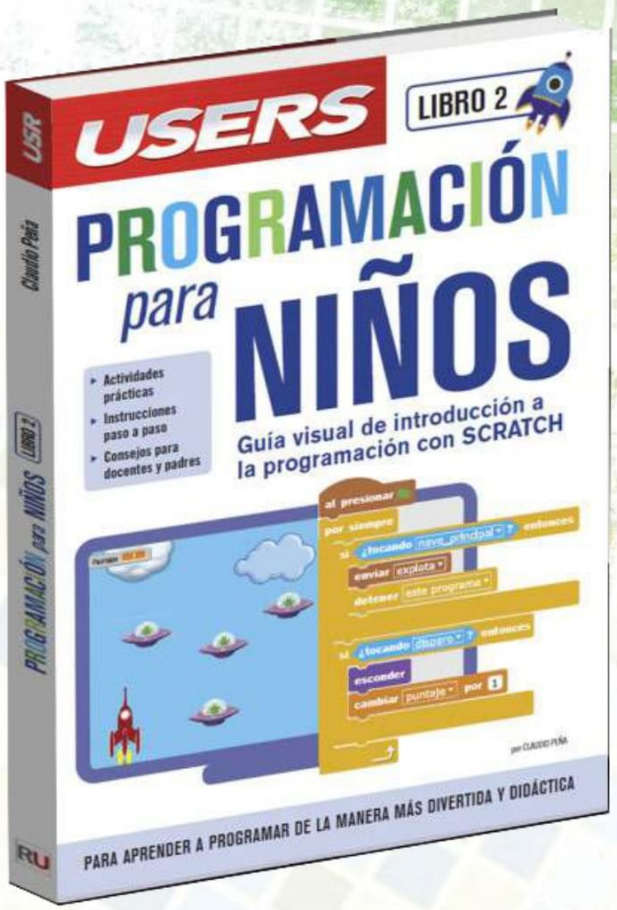
P



R

S

T



Scratch es el lenguaje de programación visual para niños más utilizado en el Mundo. Es de distribución gratuita y está orientado a desarrollar proyectos por medio de coloridos bloques funcionales. Esta obra en dos volúmenes es una valiosa herramienta para docentes y padres que quieren enseñar programación y pensamiento lógico a los niños. El abordaje es muy visual, con abundantes actividades prácticas e instrucciones paso a paso.

- **EDUCACIÓN - DESARROLLO**
- **NIVEL: BÁSICO | 336 PÁG**

COMPRANDO LOS LIBROS EN NUESTRO SITIO DE VENTAS **USERSHOP**, OBTENDRÁ GRATUITAMENTE LA VERSIÓN DIGITAL PARA LEER ONLINE O DESCARGAR EN SU DISPOSITIVO.

LEA EL SUMARIO COMPLETO Y UNA MUESTRA DEL LIBRO EN USERSHOP.REDUSERS.COM

usershop.redusers.com

usershop@redusers.com

54 (011) 4110-8700

USERS

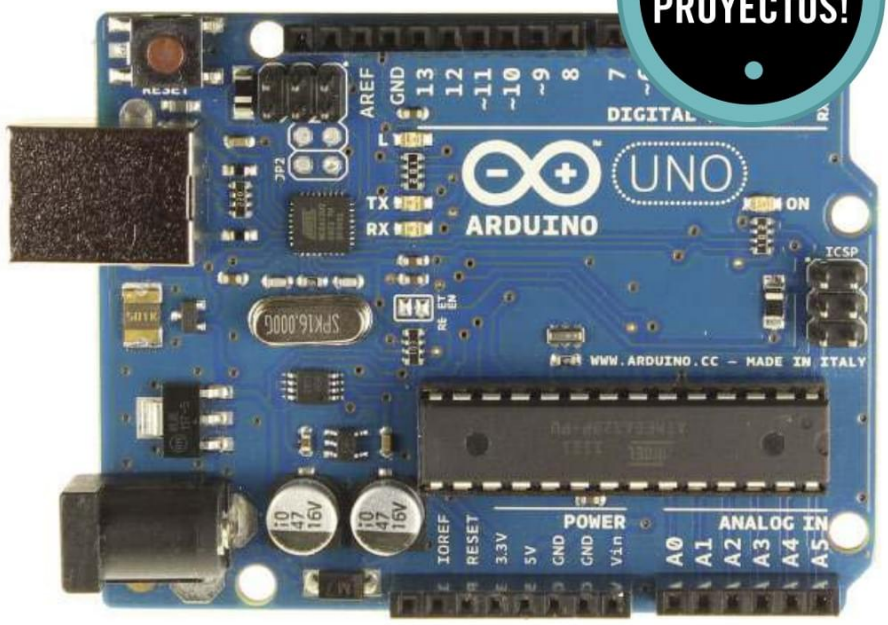
ARDUINO

Este libro ofrece al lector los conceptos necesarios para desarrollar sus proyectos en Arduino: la electrónica básica, el hardware y el entorno de programación. De manera didáctica y paso a paso, se explican distintos proyectos prácticos para que el lector obtenga sus primeros prototipos.

¡CREA TUS PROYECTOS!

CONTENIDO

- Electrónica general
- Placas Arduino
- Shields
- Programar Arduino
- LEDs y sensores
- Proyectos: detección de luz, sonidos
- Robótica, drones y más.




```
sketch_apr12b
void setup()
    {
        int Led = 0 ;
        for ( Led = 4 ; Led < 10 ; Led++)
            pinMode( Led , OUTPUT) ;
    }
void loop()
 {
     int paso = 2;
     for (int Led = 4; Led < 7; Led++)
     {
       digitalWrite(Led, HIGH);
       digitalWrite(Led + 2 * paso + 1, HIGH);
       delay(200);
       digitalWrite(Led, LOW);
       digitalWrite(Led + 2 * paso + 1, LOW);
       delay(200);
       paso--;
      }
       paso = 0;
       for (int Led = 6; Led > 3; Led--)
           {
       digitalWrite(Led, HIGH);
       digitalWrite(Led + 2 * paso + 1, HIGH);
       delay(200);
Compilado
```

SOBRE EL AUTOR

Claudio Peña Millahual es un prolífico autor y docente de tecnología. Dirige talleres de Arduino, donde enseña conceptos básicos de programación y electrónica para niños y jóvenes.

NIVEL DE USUARIO

Intermedio / Avanzado

CATEGORÍA

Hardware / Electrónica / Programación

ISBN 978-987-46518-7-7

RU

USERSHOP.REDUSERS.COM
Conozca los mejores libros sobre tecnología.

PROFESOR EN LÍNEA
Ante cualquier consulta técnica relacionada con el libro, puede contactarse con nuestros expertos: profesor@redusers.com.