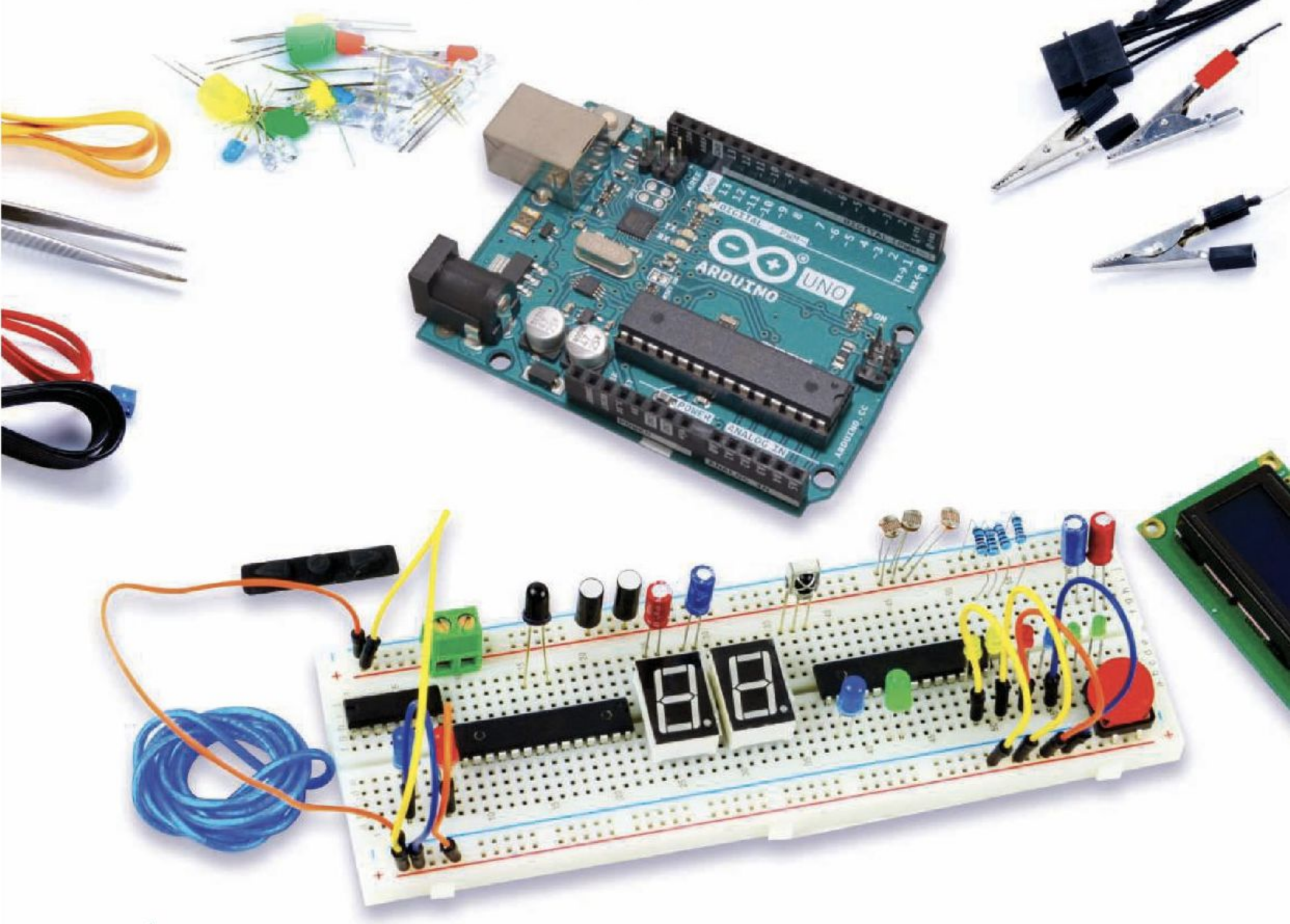
USERS

Descubriendo Arduino



- Introducción a la electrónica y sus componentes
- La interfase de programación
- Sensores y controladores

UNA GUIA PARA REALIZAR TUS PRIMEROS PROYECTOS

Descubriendo **Arduino**

El fenomenal éxito de la plataforma Arduino tiene varios motivos.

Por un lado, el hardware y el software son **open-source**: las especificaciones del circuito y el código fuente son de libre distribución.

Esto generó a su vez el principal motivo de éxito: la **enorme comunidad** de usuarios de la plataforma, a cuya ayuda tarde o temprano tendremos que recurrir.

Por otro lado, es una plataforma **económica**, se trata de una avanzada placa de desarrollo por algunas decenas de dólares.

Pero el mayor encanto de la plataforma Arduino son sus **posibilidades**, nos permite agregar "inteligencia" a cualquier dispositivo, obtener información del mundo exterior mediante sensores y realizar acciones controlando interruptores, luces y motores.

Adelante entonces, ¡el límite son sólo nuestras ganas de aprender y nuestra imaginación!

Miguel Lederkremer (@leder)
Director Editorial

TÍTULO
Descubriendo Arduino

AUTOR
Claudio Alejandro Peña Millahual

DISEÑO Y PRODUCCIÓN
Gustavo De Matteo

COLECCIÓN	Users Guías Prácticas
FORMATO	28 x 20 cm
PÁGINAS	144
ISBN	978-987-4958-24-2

facebook.com/redusers twitter.com/redusers instagram.com/reduserscom/ redaccion@redusers.com

Peña Millahual, Claudio Alejandro
Descubriendo Arduino / Claudio Alejandro Peña Millahual. - 1a ed. - Ciudad Autónoma de Buenos Aires: Six Ediciones, 2020.
144 p.; 28 x 20 cm. - (Guías users; 13)

ISBN 978-987-4958-24-2

1. Hardware. 2. Programación. I. Título.
CDD 005.4

Descubriendo **Arduino**

Esta guía te abrirá el Mundo de la plataforma para proyectos electrónicos más utilizada en el mundo. Aprenderás elementos básicos de electrónica, los componentes necesarios, las distintas placas disponibles y la plataforma IDE para programar las placas. Veremos también algunos proyectos básicos para dar tus primeros pasos en esta fascinante plataforma.

INTRODUCCIÓN

Electricidad y electrónica 4
Componentes electrónicos 8
Circuitos electrónicos 10
Microcontroladores 12
Placas de desarrollo 13

PRINCIPIOS BÁSICOS

¿Qué es Arduino? 14
Comunidad 15
Hardware 16
Software 20
Placas Arduino 22
Placas no oficiales 28
Usos de Arduino 30

COMPONENTES NECESARIOS

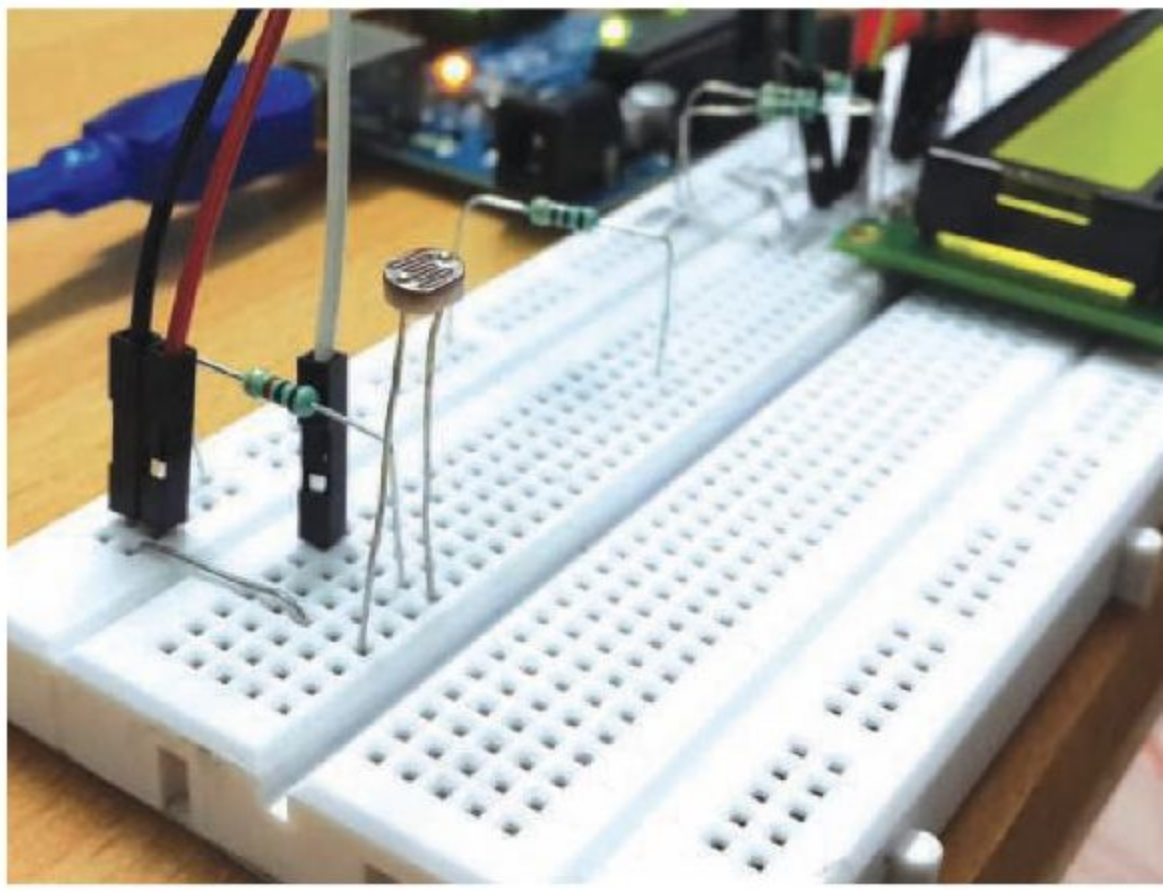
¿Qué necesitamos? 32
Prueba de conexión 34
Comparación con otras placas 37
Protoboard 39
Cables de puente 40
Condensador 42
Diodo 44
Diodo emisor de luz (LED) 45
Puente H 46
Broche de presión de pila 47
Potenciómetro 48
Pantalla de cristal líquido 49
Motor de corriente continua 50
Optoacoplador 51
Pulsador 52
Resistencias 54
Fotorresistencia 55
Transistor 56
Zumbador piezoeléctrico 57
Sensores 58
Servomotor 59

ARDUINO IDE

¿Qué es Arduino IDE? 60
Instalación del IDE 62
Entorno de trabajo 64
Configuración inicial 66
Librerías 68

PROGRAMACIÓN

Estructura de un sketch 76
Aspectos importantes 78
Funciones y parámetros 80
Variables 82
Datos y operadores 86
Tipos de operadores 88
Estructuras de control 90
Bucles 92



LEDS

LEDs y Arduino 94
Conexión básica de un LED 96
¿Cómo controlar el LED? 100
Agregar iteraciones 102
Conectar el circuito 104
Seis LEDs en secuencia 107
Secuencia de 8 LEDs 111

DETECCIÓN DE LUZ

¿Qué es una fotorresistencia? 114
¿Cómo aprovechar una fotorresistencia? 116
Armar el circuito 119
Primera versión del código 122
Segunda versión del código 124
Tercera versión del código 126

SONIDO

Buzzer 128
Proyecto básico 130
Reproducir una escala musical y una melodía 134

SENSORES

¿Qué es un sensor? 136
Entradas en Arduino 138
Sensores para Arduino 142

Electricidad y electrónica

Para comenzar, conoceremos la electricidad como fenómeno físico cuyo origen son las cargas eléctricas, y la electrónica como la encargada de controlar y conducir el flujo de electrones.

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.

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.

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 los 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.

Carga eléctrica

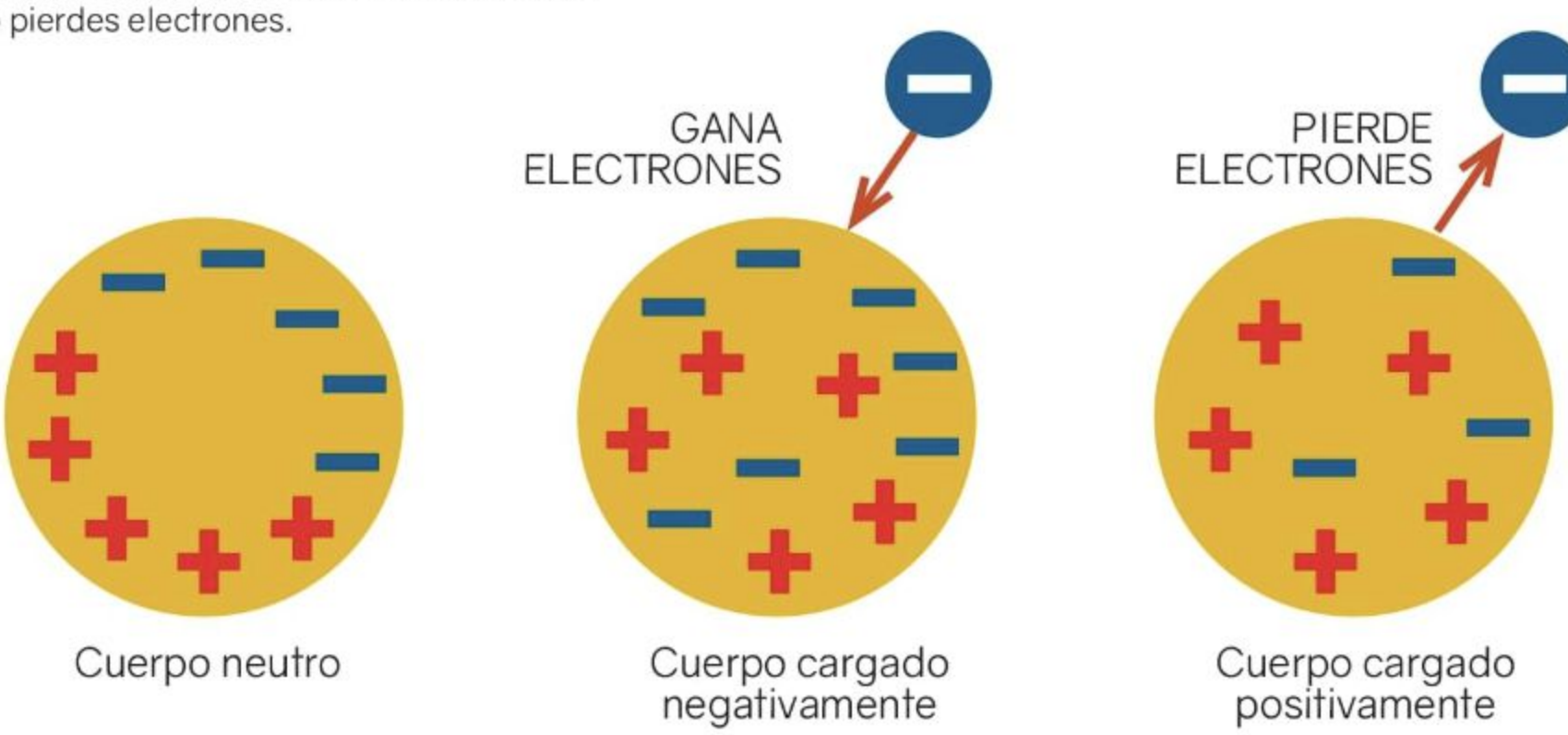
La **carga eléctrica** es una capacidad que tienen las partículas de atraer o repeler otras. Es la cantidad de energía que poseen las partículas que componen el átomo; este puede quedar cargado positivamente (si pierde electrones de sus órbitas) o negativamente (si gana electrones). Juntas, generarán fuerzas de atracción y de repulsión, tal como se puede observar cuando utilizamos un magneto y un trozo de metal.

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.

Propiedades de la carga eléctrica

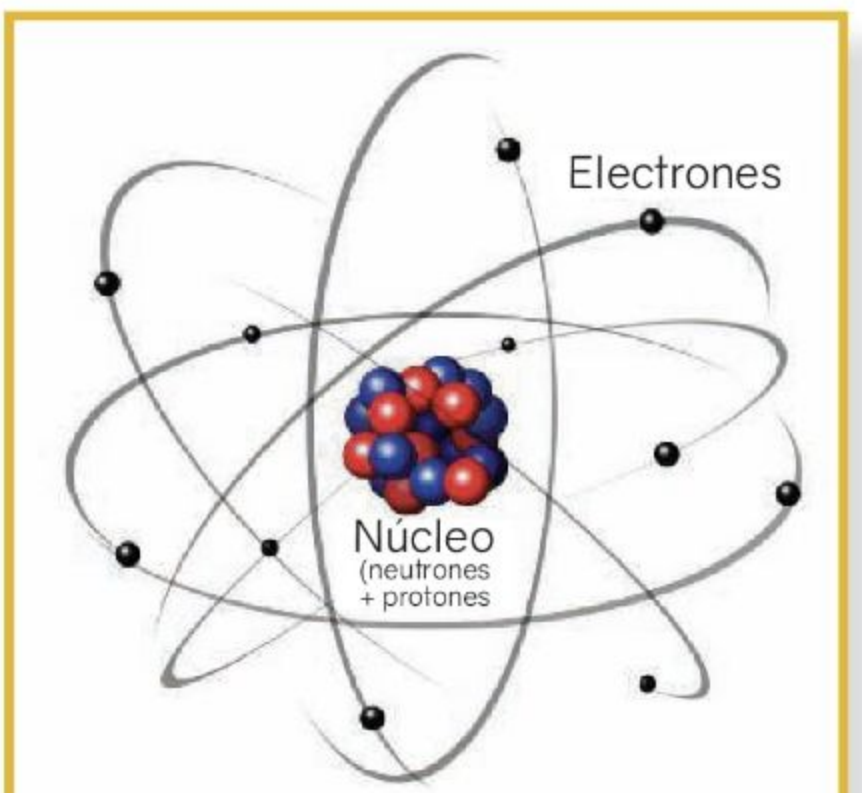
La mayoría de los cuerpos tiene el mismo número de electrones que de protones, por lo que su carga neta es cero (son neutros). Se electrizan cuando ganan o pierdes electrones.



El 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 de los electrones se ocupa de crear un campo magnético que, finalmente, origina movimiento.



Electrónica

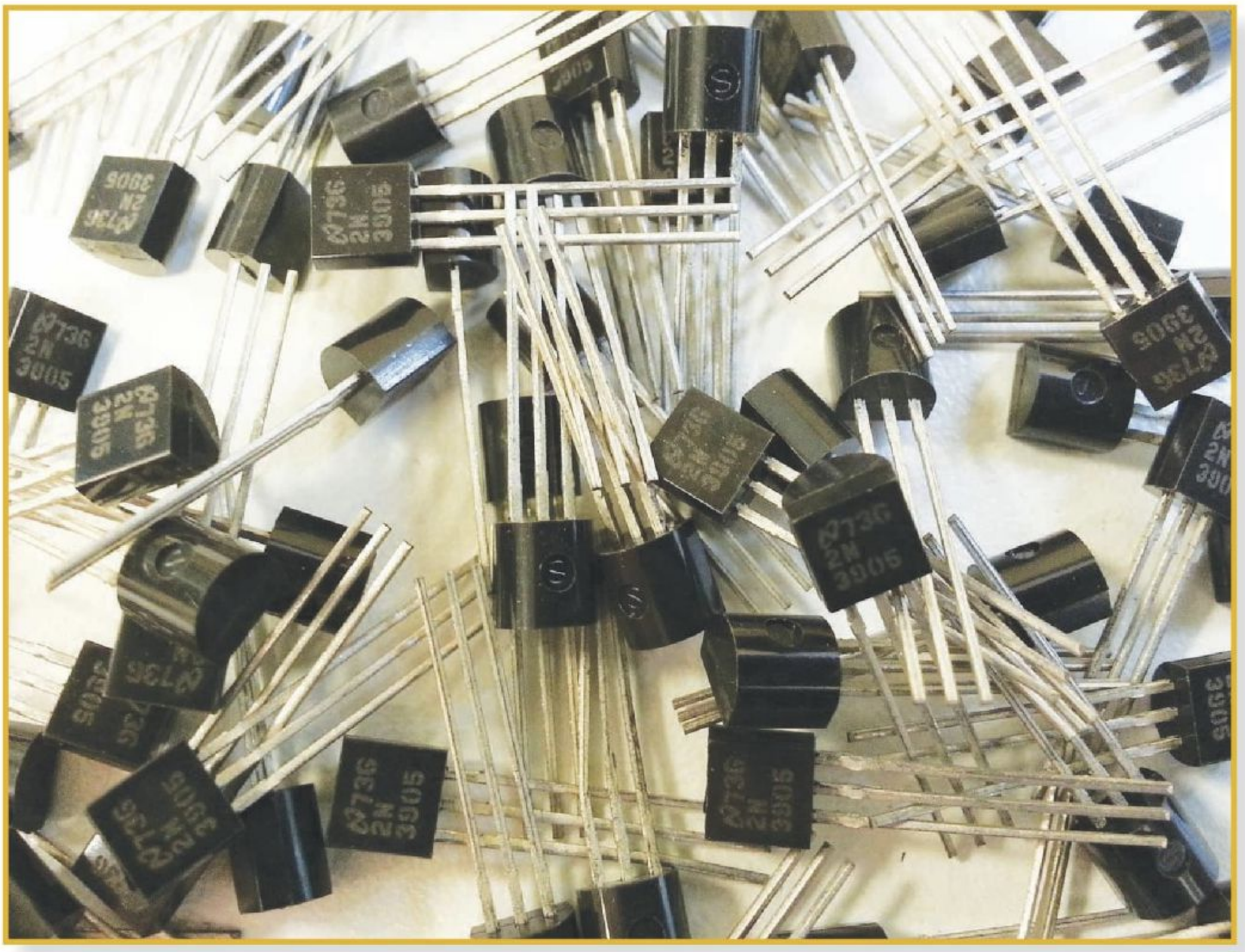
La **electrónica** es una rama de la física cuya finalidad es encargarse del control, la conducción y el flujo de los electrones o de cualquier partícula cargada eléctricamente.

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 de los átomos.

Los **circuitos electrónicos** hacen posible la conversión y la distribución de la energía eléctrica, razón por la cual 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 convierten 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 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.

Un circuito integrado (CI), también conocido como chip o microchip, es una estructura pequeña, de material semiconductor, de algunos milímetros cuadrados de superficie, sobre la que se fabrican circuitos electrónicos.

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.

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:

$$V = R \times I$$

Si conocemos la tensión y la corriente, calculamos la resistencia como el cociente entre la tensión y la corriente:

$$R = V / I$$

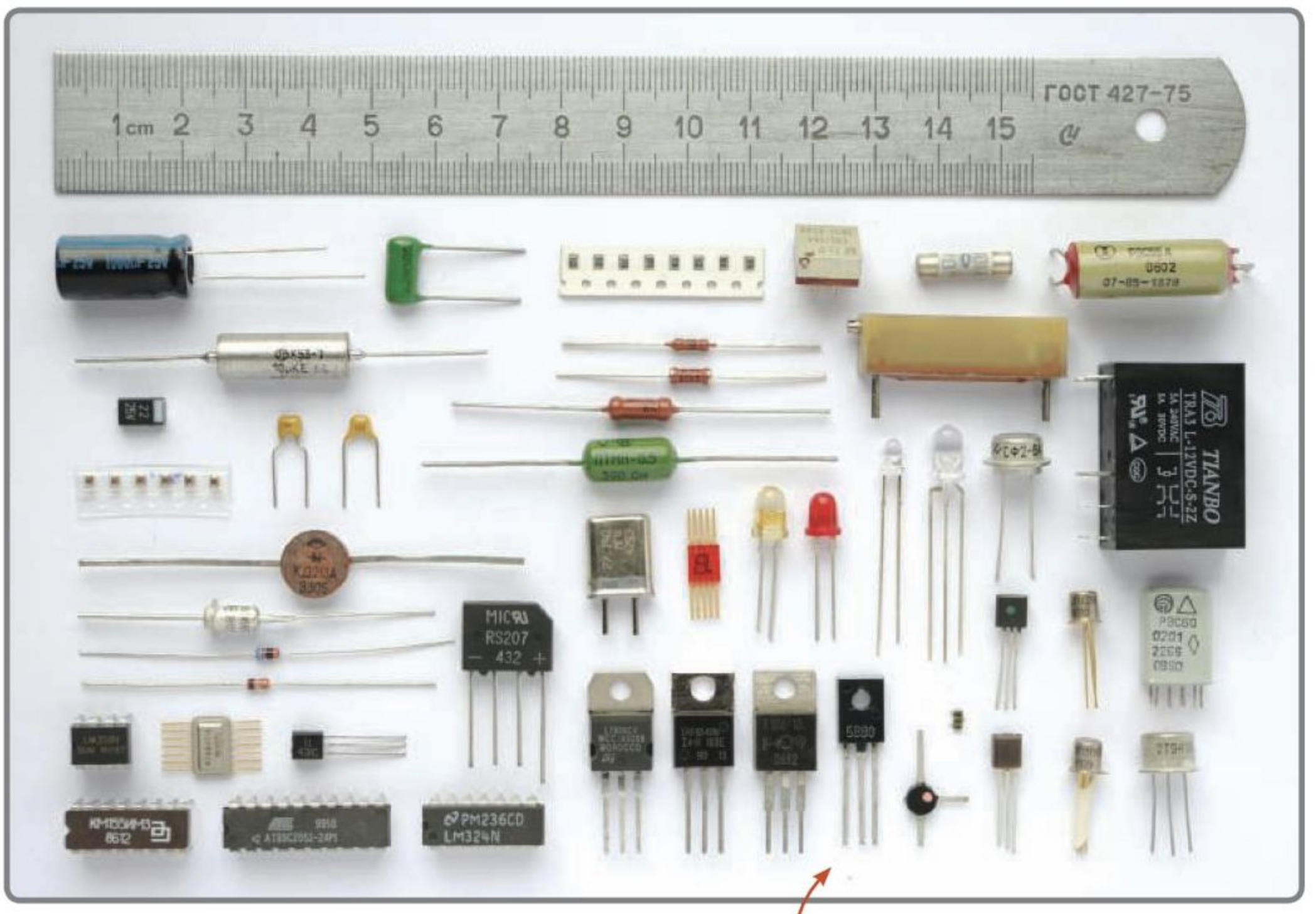
Si conocemos la tensión y la resistencia, calculamos la corriente como el cociente entre la tensión y la resistencia:

$$I = V / R$$

Componentes electrónicos

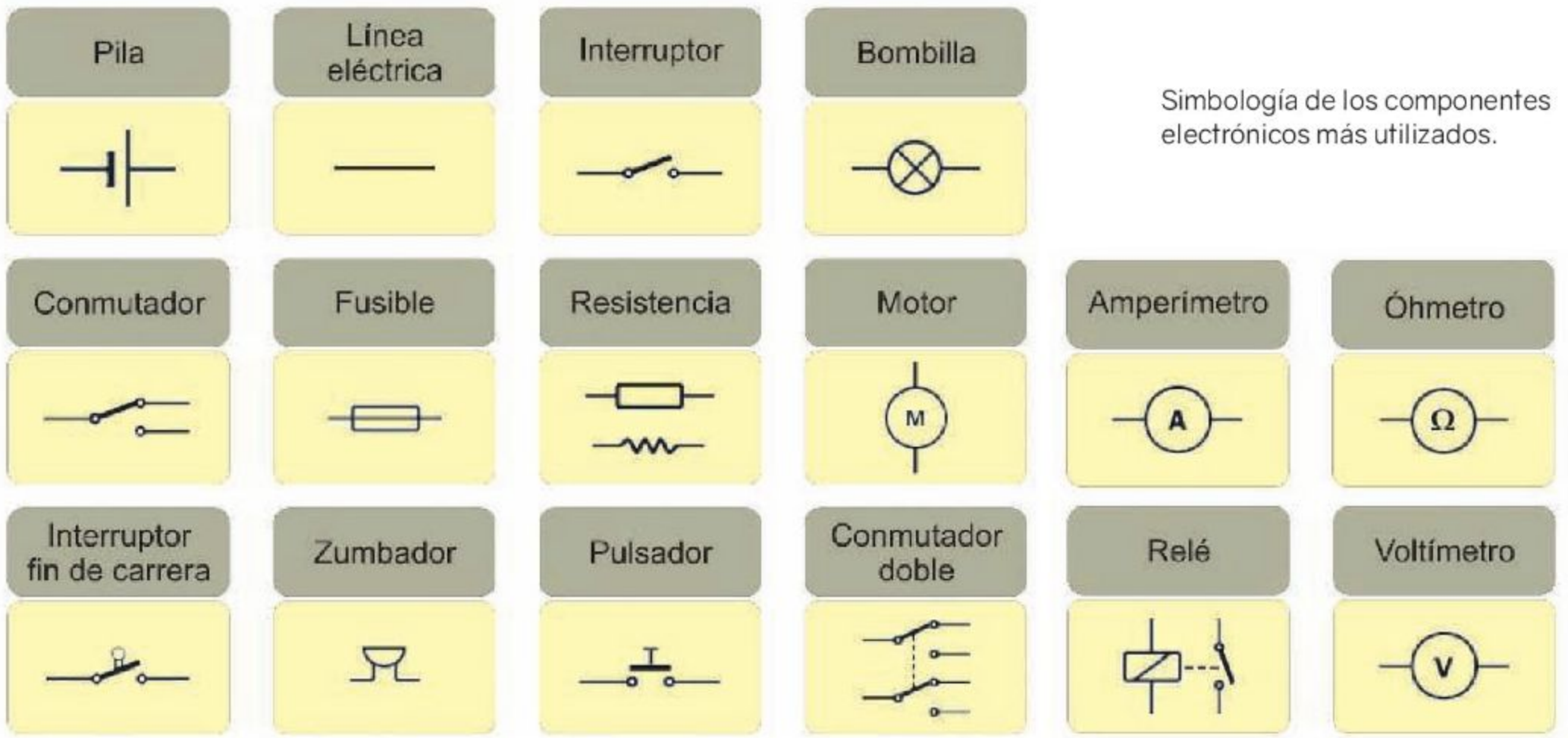
Los componentes electrónicos son aquellos que pueden formar parte de un circuito electrónico; por lo general, se encuentran conectados mediante soldadura al circuito impreso.

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) y según el **tipo de energía** (electromagnéticos, electroacústicos, optoelectrónicos).
En esta ocasión, los clasificaremos según su funcionamiento: **activos y pasivos**.



Los componentes electrónicos se suelen encapsular, generalmente, en un material cerámico, metálico o plástico.

Existe una gran cantidad de componentes activos, siendo usual que un sistema electrónico se diseñe a partir de uno o varios de ellos, cuyas características propias se encargan de condicionar su comportamiento.



Simbología de los componentes electrónicos más utilizados.

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.

COMPONENTE ELECTRÓNICO ACTIVO	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 y, de esta forma, aseguran que las señales eléctricas puedan transmitirse o que se modifique su nivel.

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

COMPONENTE ELECTRÓNICO PASIVO	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.

Circuitos electrónicos

Se trata de circuitos eléctricos que contienen dispositivos tales como transistores y válvulas, entre otros.

Un circuito electrónico es una asociación de componentes que, funcionando en conjunto, realizan un determinado tratamiento de las señales eléctricas. Por ejemplo: generan ondas de radio, aumentan la potencia de la señal, recuperan la imagen o el sonido que transporta una onda, etcétera.

Las fronteras entre los circuitos eléctricos y electrónicos no son muy nítidas, aunque los últimos están asociados a semiconductores.

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:

1

El origen de la electricidad tendrá dos terminales: positivo y negativo.

2

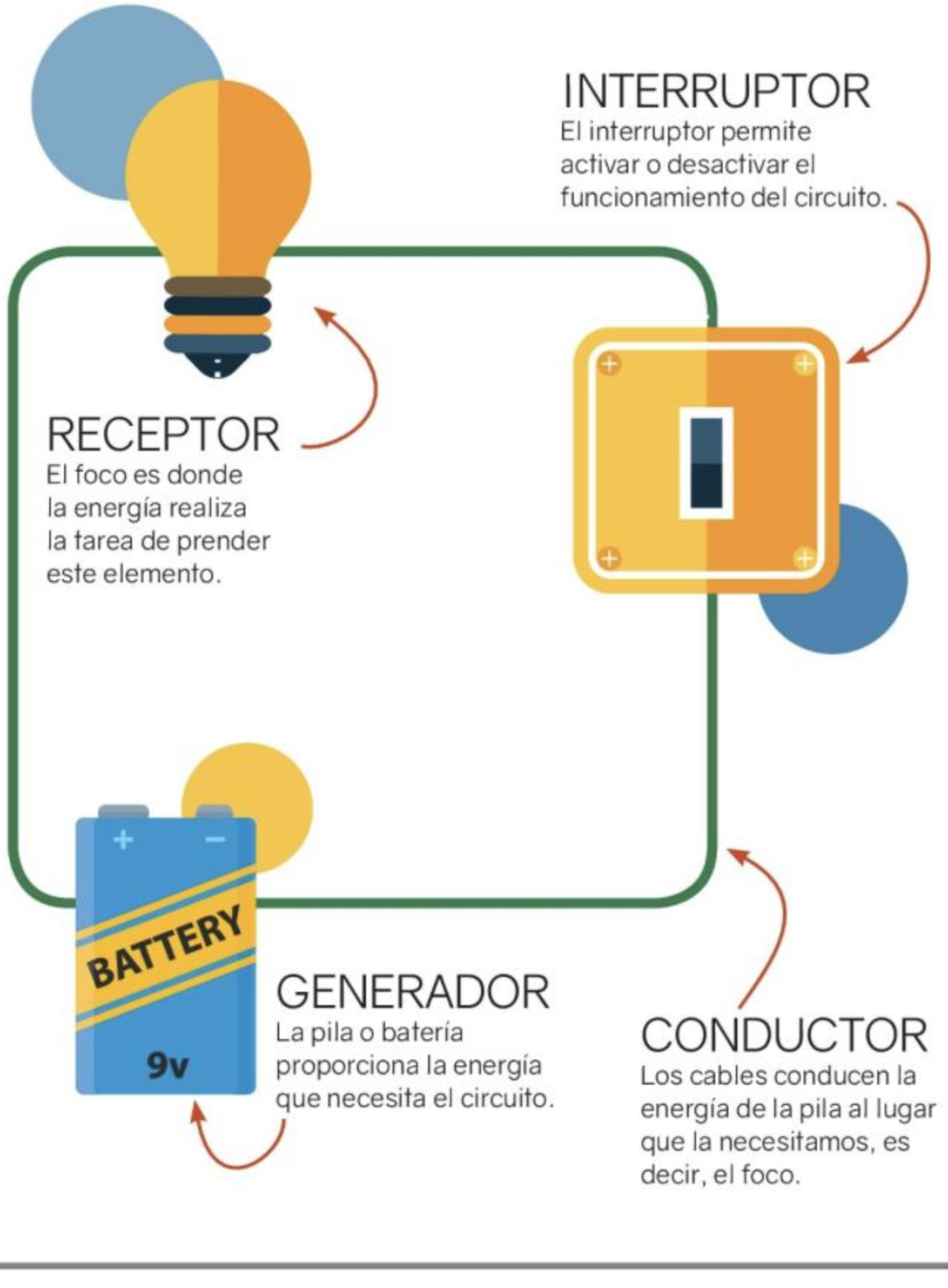
El origen del flujo eléctrico, por ejemplo, un generador o una batería, buscará empujar los electrones fuera de su terminal negativo, utilizando un cierto voltaje. Para ejemplificarlo, pensemos en una pila, que se encarga de empujar los electrones a 1,5 voltios.

3

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.

Circuito eléctrico básico

¿Qué función cumple cada elemento?



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.

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:

1 Circuitos analógicos

En este tipo de circuitos, las señales eléctricas varían de manera 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.

2 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, PDAs y microprocesadores.

3 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 LOS CIRCUITOS ELÉCTRICOS Y LOS CIRCUITOS 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.

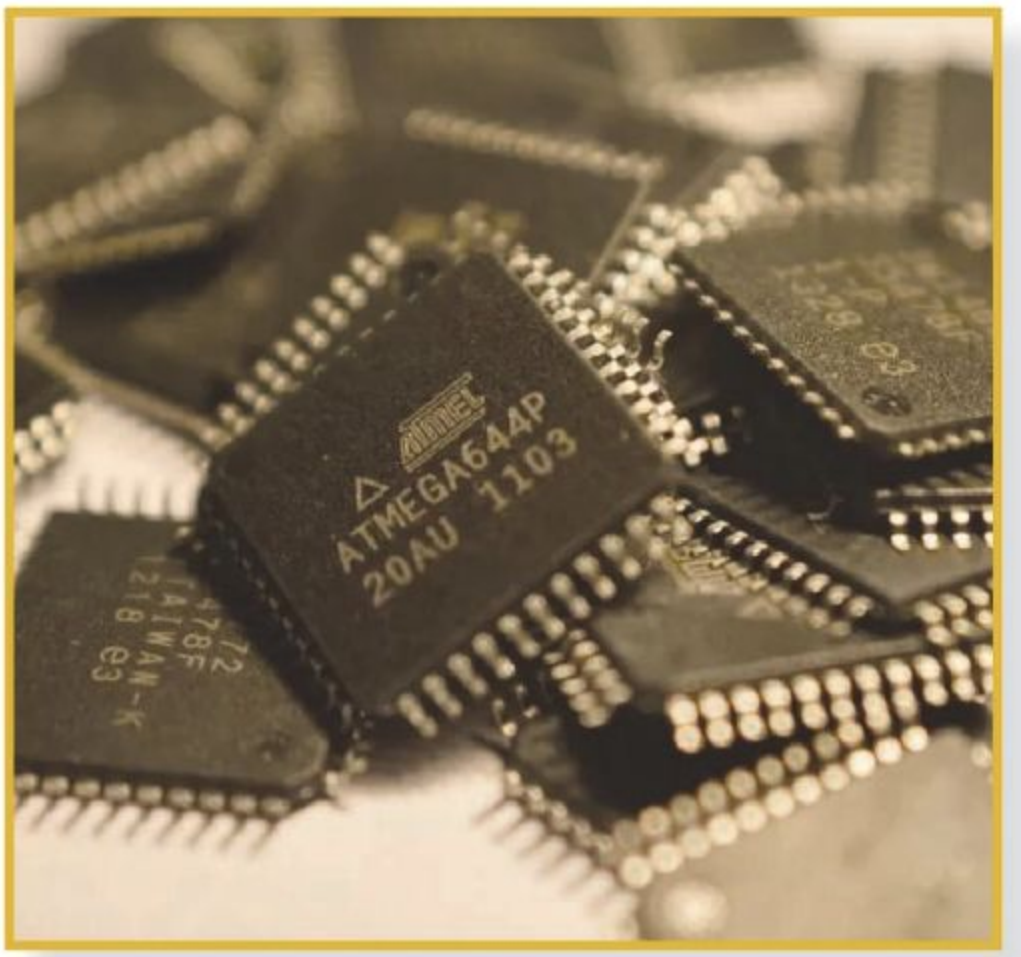
Microcontroladores

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, en la industria, etcétera. Si adaptamos la idea del microcontrolador al contenido de esta obra, diremos que es posible utilizarlo para aplicaciones tales como manejo de sensores, calculadoras, avisos lumínicos, secuenciador de luces, cerrojos electrónicos, control de motores y 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 modo que presenta un período de arranque breve y, por lo tanto, es capaz de ejecutar el código más rápido. Si bien esto parece una ventaja, debemos considerar que conlleva una 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 explicado, un MPU no estará en funcionamiento con tanta rapidez como un MCU, pero podrá disponer de una cantidad de memoria mucho mayor, gracias al uso de recursos externos.
Otra diferencia 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 distintos.



Microcontrolador Atmega 644P.

Arquitecturas de construcción

Por un lado, los MPU suelen diseñarse teniendo en cuenta la arquitectura Von Neumann; por otro, 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 mayor seguridad.

Placas de desarrollo

En este apartado conoceremos, a grandes rasgos, la evolución de los MCU hasta llegar a las actuales placas de desarrollo.

Si profundizamos en la historia, encontramos 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 eso 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.

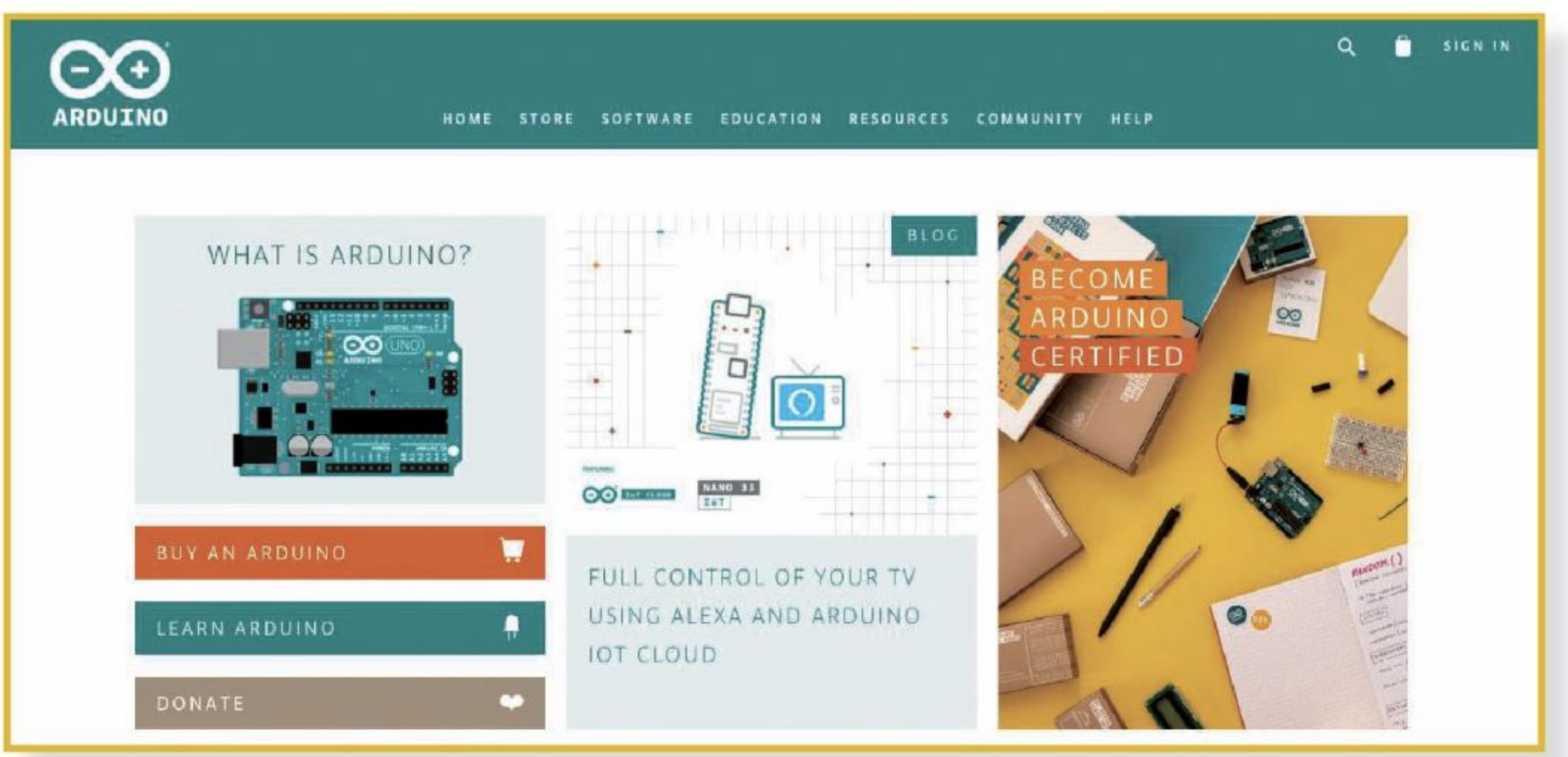
Una de las placas más populares es **Arduino**. La idea principal fue brindar acceso a MCU embebidos, pensando en proyectos de diseño interactivo. Gracias a esto, Arduino permite crear toda clase de prototipos electrónicos en forma rápida y económica.

Su importancia es tal, que todo principiante, entusiasta y experto en el mundo de la electrónica lo utiliza para realizar sus proyectos.

¿Qué es Arduino?

Definir 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.

En términos formales, **Arduino** es una plataforma de hardware libre –creada por David Cuartielles y Massimo Banzi–, que está basada en una placa con un microcontrolador y un entorno de desarrollo, y fue creada para facilitar el uso de la electrónica en proyectos multidisciplinarios, 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 y una placa; también integra un entorno de desarrollo y es una plataforma de hardware open source.
Por otra parte, de manera 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 necesario 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.



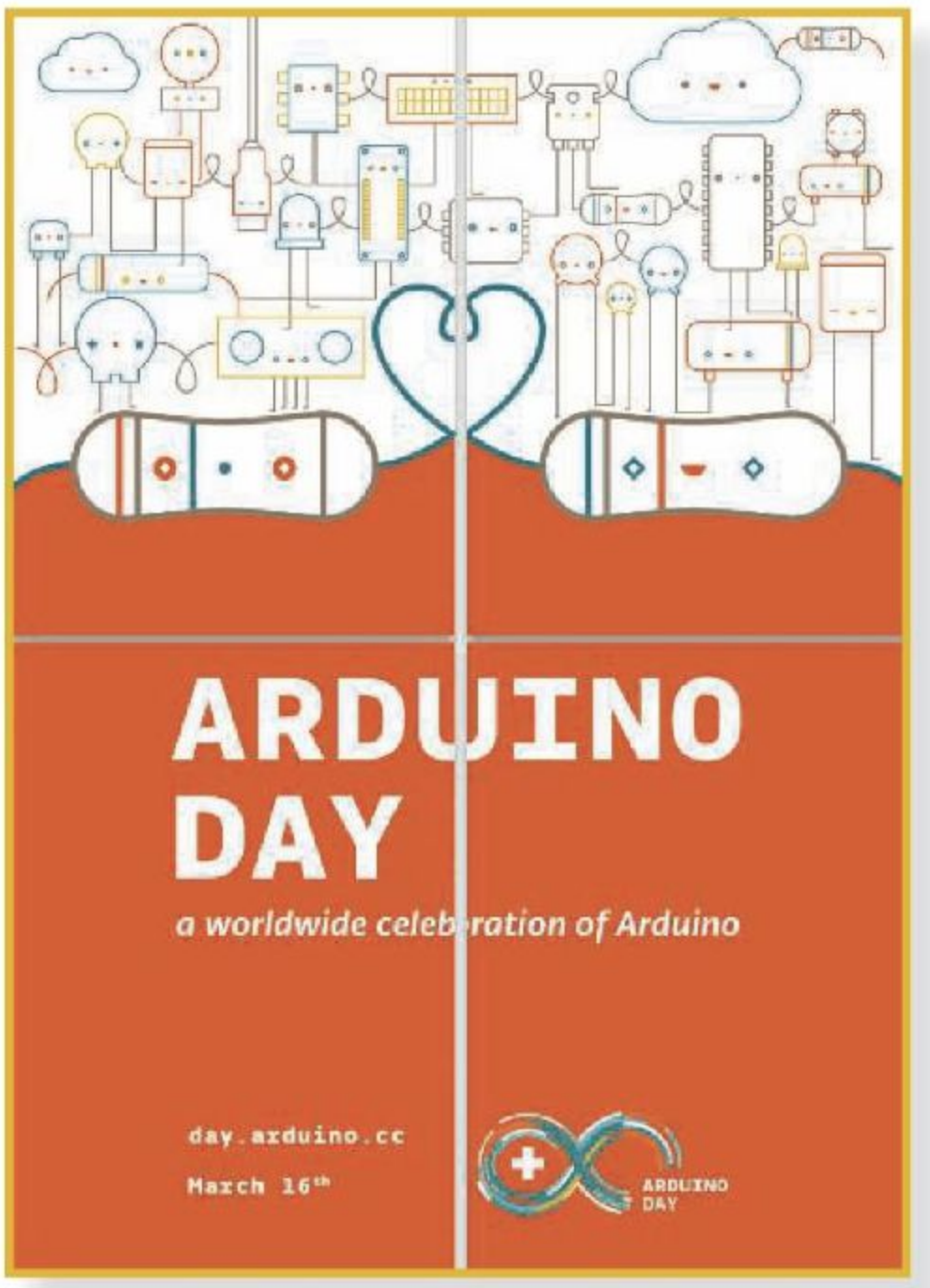
Para definir Arduino, debemos tener en cuenta el hardware, el software y también la comunidad que lo mantiene.

Comunidad

La gran comunidad detrás de Arduino es la que ha logrado darle la importancia que esta placa posee en todo el movimiento Maker, como la mejor y más versátil placa de desarrollo.

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, 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.



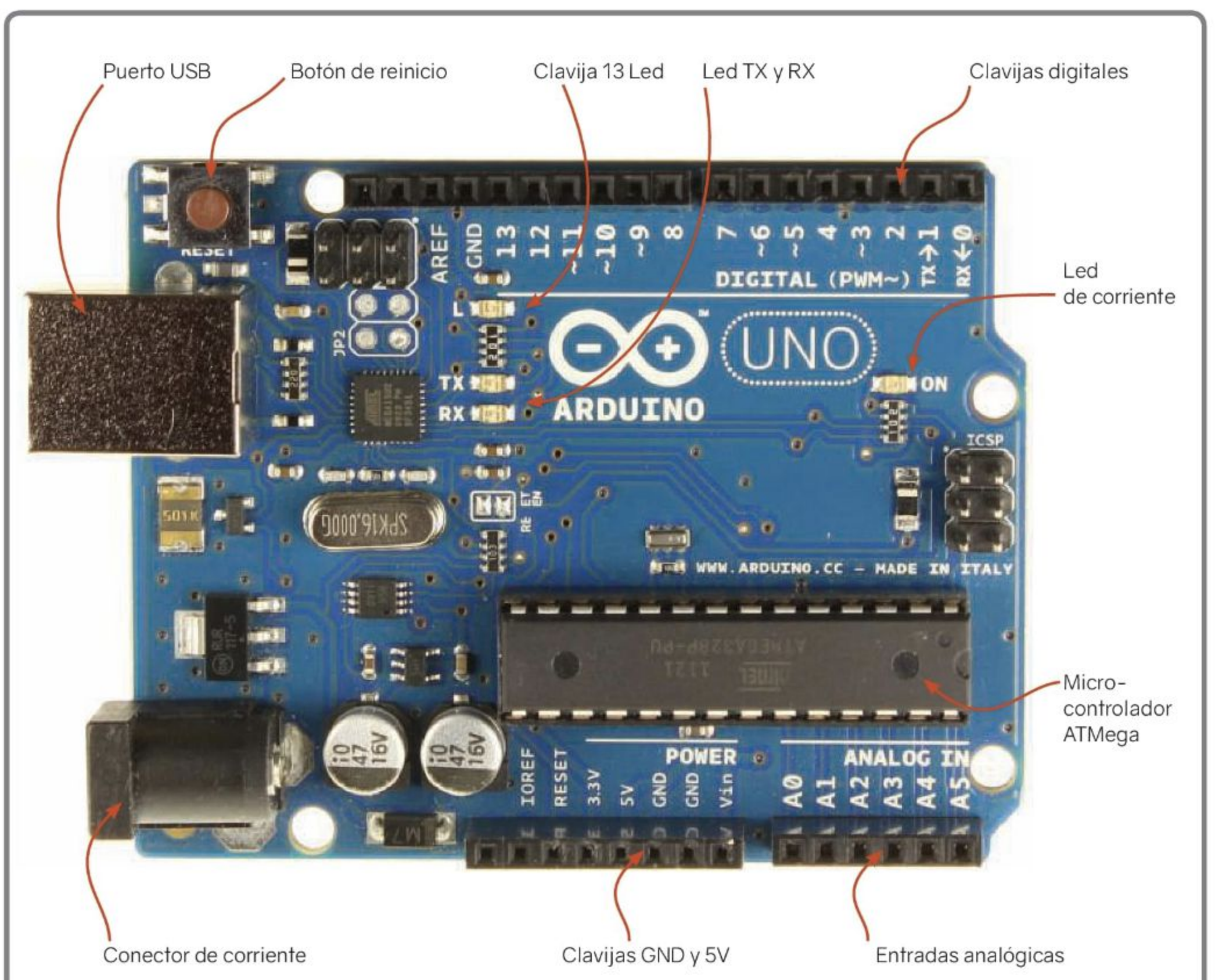
Movimiento Maker

Hace algún tiempo, surgió 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.

Hardware

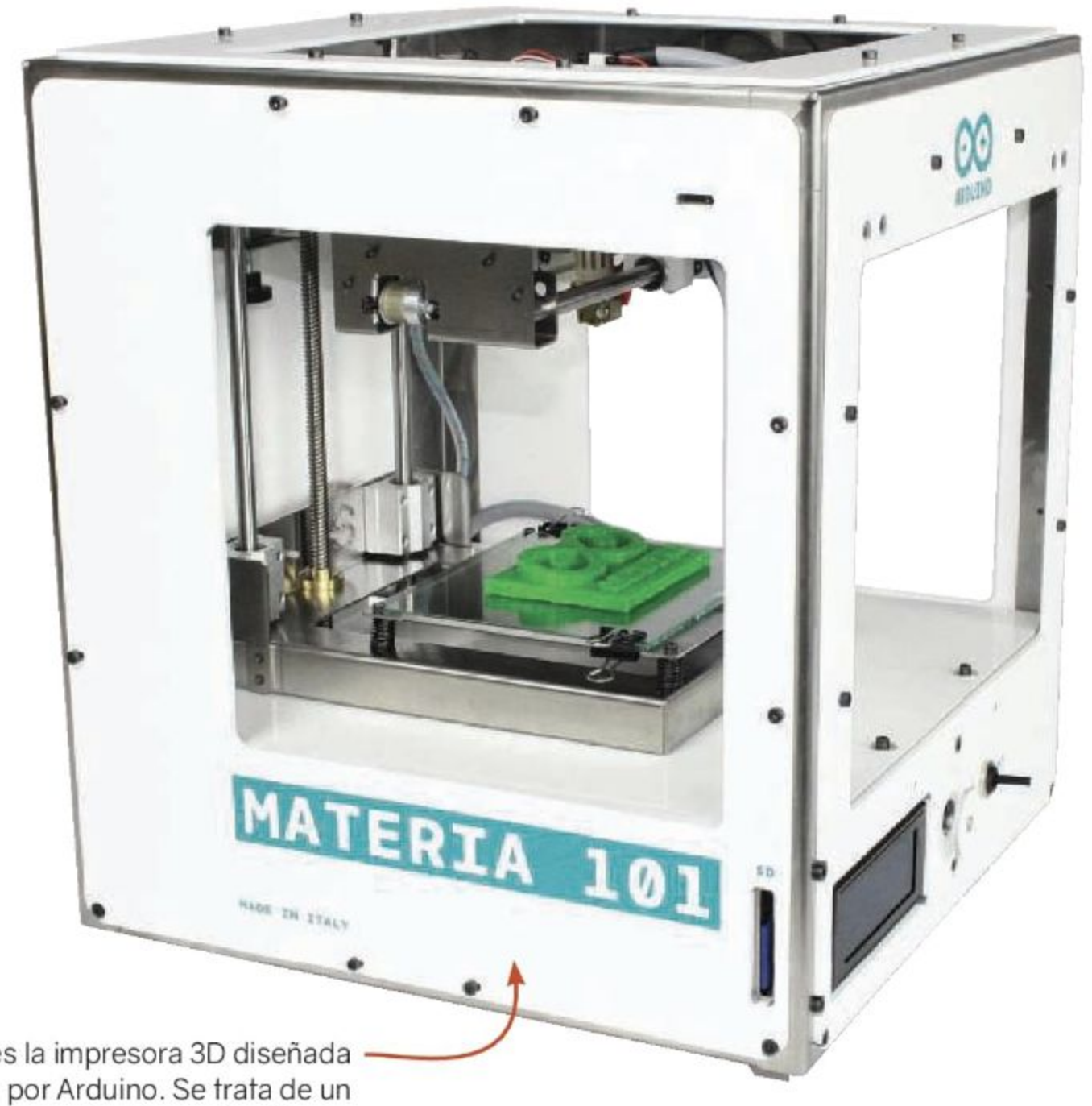
Si analizamos el hardware de Arduino, encontraremos una placa de circuito impreso con un microcontrolador (Atmel AVR), y un conjunto de puertos digitales y analógicos de entrada/salida. Además, posee un puerto USB mediante el que se alimenta y se comunica con la PC.

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 conectarla a la computadora para integrar el trabajo con el software, tal como veremos en la siguiente sección.



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, está 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.



Materia 101 es la impresora 3D diseñada y fabricada por Arduino. Se trata de un sistema open source, tanto en el software como en el hardware.

Los 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.

DIVERSOS MODELOS DE PLACAS ARDUINO Y SUS PRINCIPALES CARACTERÍSTICAS

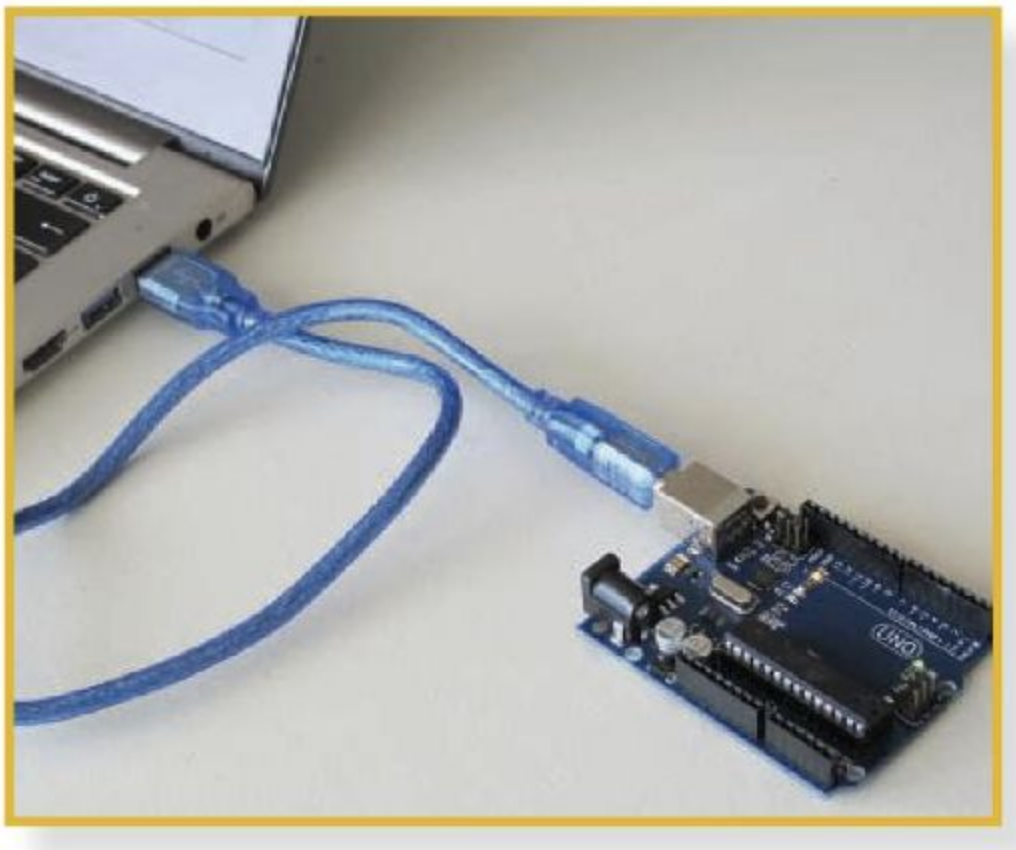
MODELO	MICROCONTROLADOR	ENTRADAS/SALIDAS DIGITALES	ENTRADAS/SALIDAS ANALÓGICAS	MEMORIA FLASH
Arduino Leonardo	ATmega32U4	20	12	32 Kb
Arduino UNO R3	ATmega328	14	6	32 Kb
Arduino Mega 2560 R3	ATmega2560	54	16	256 Kb
Mega pro 3.3V	ATmega2560	54	16	256 Kb
Arduino mini 05	ATmega328	14	6	32 Kb
Arduino Fio	ATmega328P	14	8	32 Kb
Mega Pro Mini 3.3V	ATmega2560	54	16	56 Kb
Arduino DUE	AT91SAM3X8E	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 u otra placa; más adelante conoceremos las características específicas de algunos de los modelos de Arduino más utilizados.

Aunque en las diferentes placas Arduino encontramos diversas especificaciones, en general, sus características básicas son: 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 5 V, EEPROM 512 bytes.

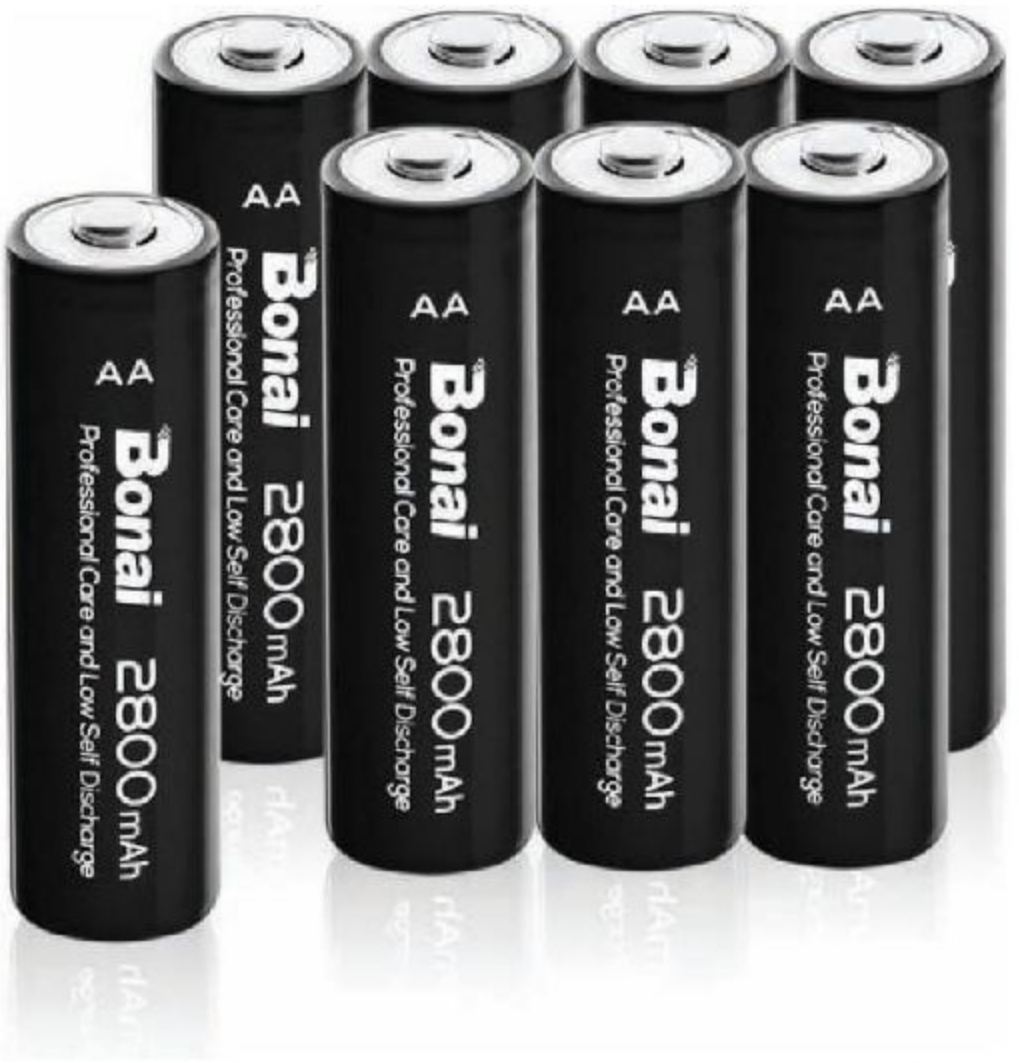
Fuente de alimentación

Un aspecto 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.



Adaptador de corriente

Es una opción similar a un cargador para teléfono móvil; resulta 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 1,5 V. Aunque se trata de una opción recomendable para proyectos que requieren movilidad, debemos considerar que su energía se consume, por lo que tendremos que cambiarlas a menudo.

Baterías LiPo

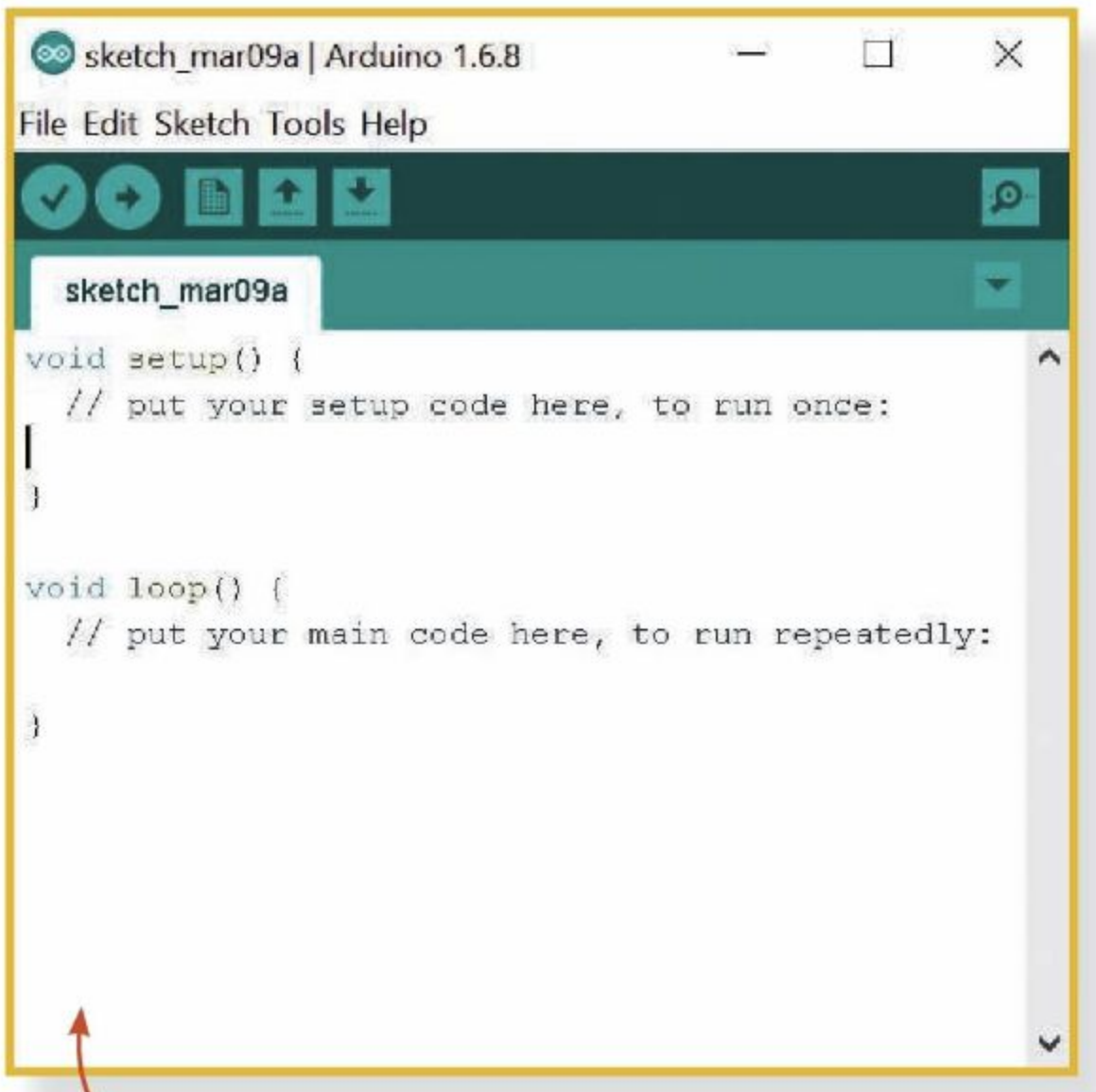
Es una opción más eficiente pues proporciona energía por bastante tiempo, aunque en comparación con las tradicionales pilas AA, estas baterías tienen un costo mayor. Son recargables, por lo que también precisaremos un módulo cargador.

Las baterías LiPo (polímero de litio) se componen de celdas de 3,7 V cada una. Son recargables, por lo que es necesario cargarlas adecuadamente para alargar su vida.

Software

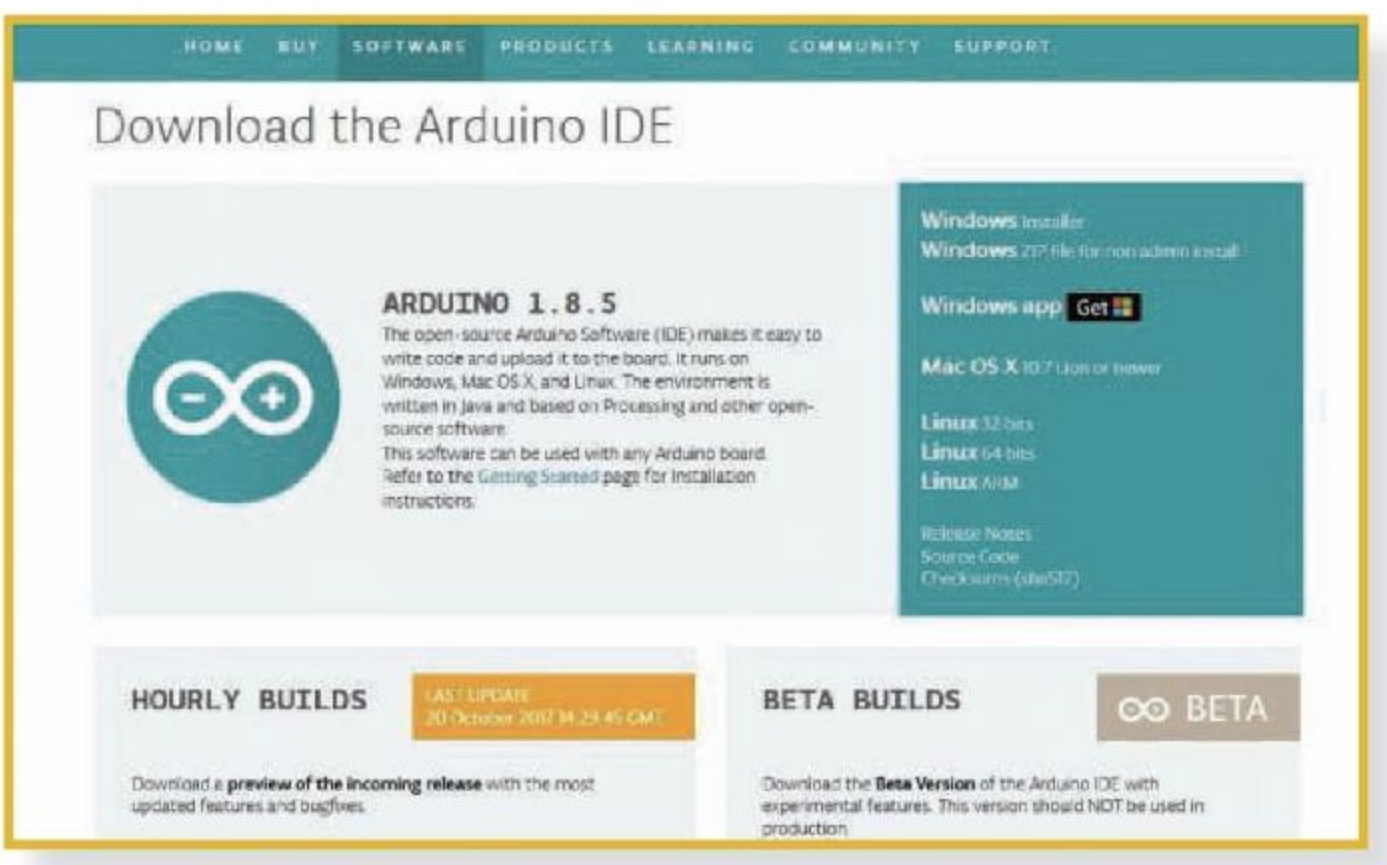
Gracias al software que integra Arduino, es posible establecer las instrucciones y los parámetros para controlar su funcionamiento y, de esta forma, generar nuestros propios proyectos.

Aunque lo que más nos llama la atención de Arduino es el hardware, la verdad es que es mucho más que placas 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.
Al igual que el hardware de Arduino, el software necesario para programarlo se distribuye libremente, por eso, solo tenemos que descargarlo desde su web.
Como vemos, uno de los componentes importantes de Arduino es el 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.

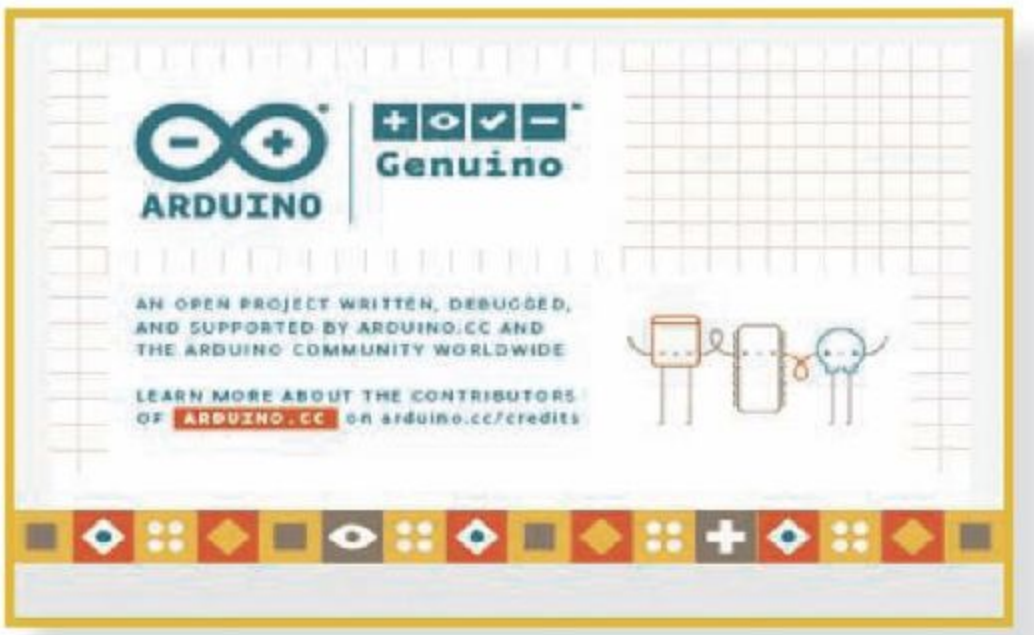


El IDE de Arduino incorpora todo lo que necesitamos para crear el código que controlará el funcionamiento de los sensores conectados a la placa Arduino, y dar vida a nuestros proyectos.

El Entorno Integrado de Desarrollo (IDE) de Arduino es una aplicación multiplataforma escrita en Java. Podemos utilizarlo para escribir y cargar programas en placas compatibles con Arduino, pero también, con la ayuda de núcleos de terceros, es posible que lo utilicemos con placas de desarrollo fabricadas por otros proveedores.



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



1 Editor de código

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

2 Compilador

Es un programa informático que se ocupa 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.

3 Depurador

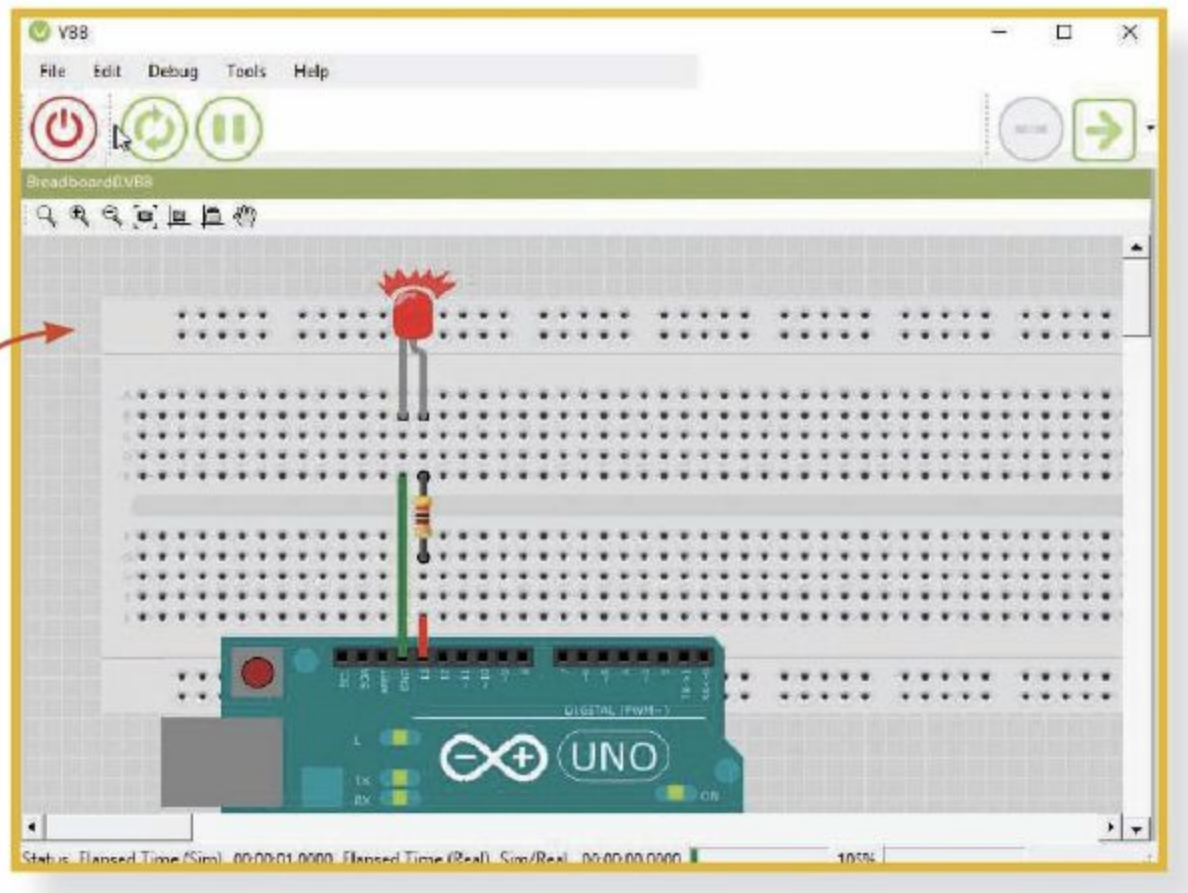
Este tipo de programa está diseñado para probar y eliminar los errores que puedan existir en el código que desarrollamos. En otras palabras, se trata de un programa que ejecuta el código para detectar posibles errores lógicos.

4 Otras opciones

Por si esto fuera poco, también dispondremos de las herramientas y opciones necesarias para cargar los programas que realicemos a la memoria flash del hardware. Es decir, podremos 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 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.

Placas Arduino

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?

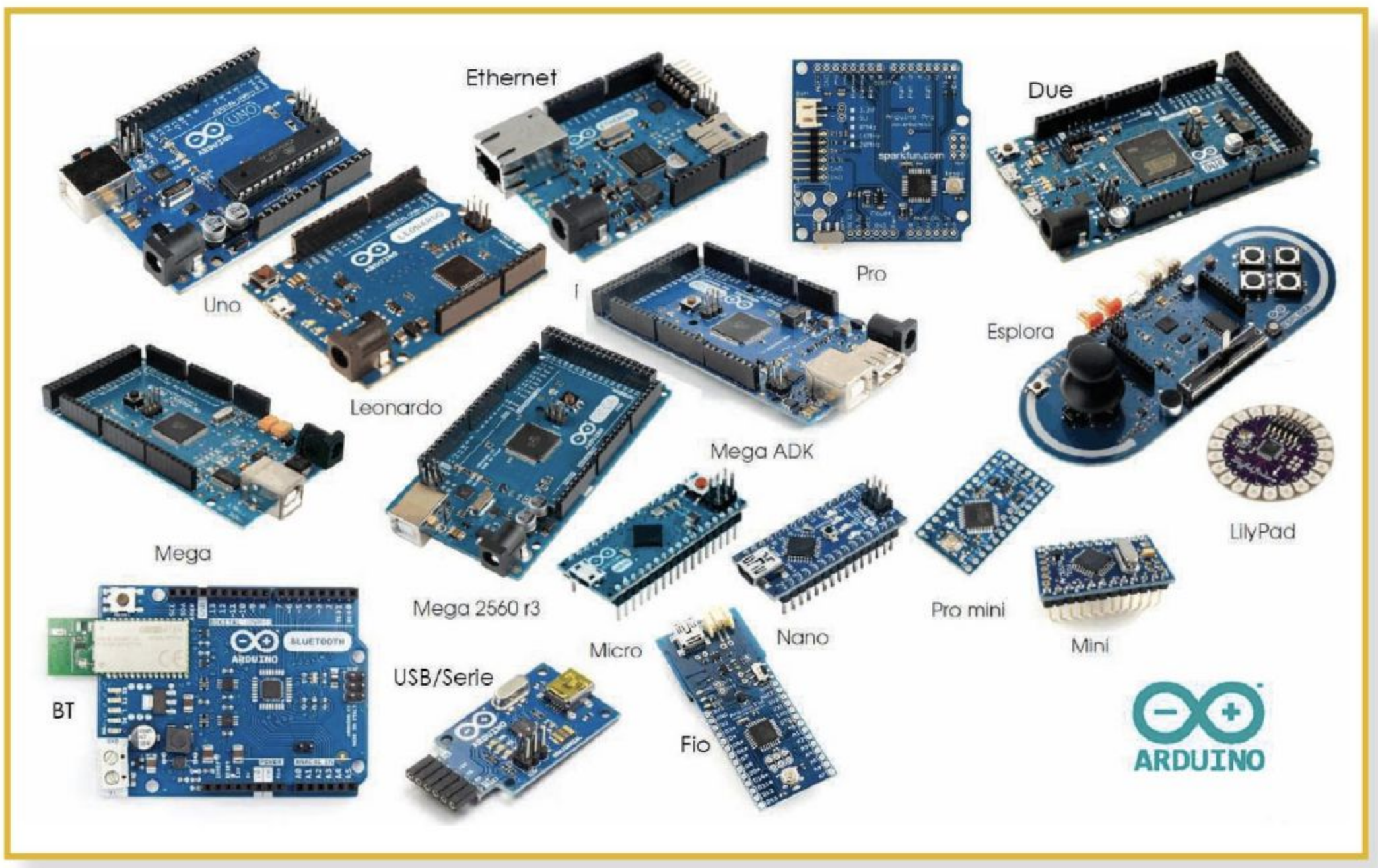
Por un lado, las placas oficiales son aquellas construidas por la empresa **Smart Projects**, por **SpartFun Electronics** o por **Gravitech**, las únicas que llevan la marca registrada Arduino y que incluyen su logo.

Por otra parte, 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 en que las placas oficiales no han llegado. En realidad, como hemos comentado en secciones anteriores, cualquiera puede crear su propia placa Arduino; en este 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 antes mencionadas.

En cualquier caso, una de las más utilizadas es la **Arduino UNO**, sobre todo, por quienes recién se inician 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 MHz a 5 V. 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

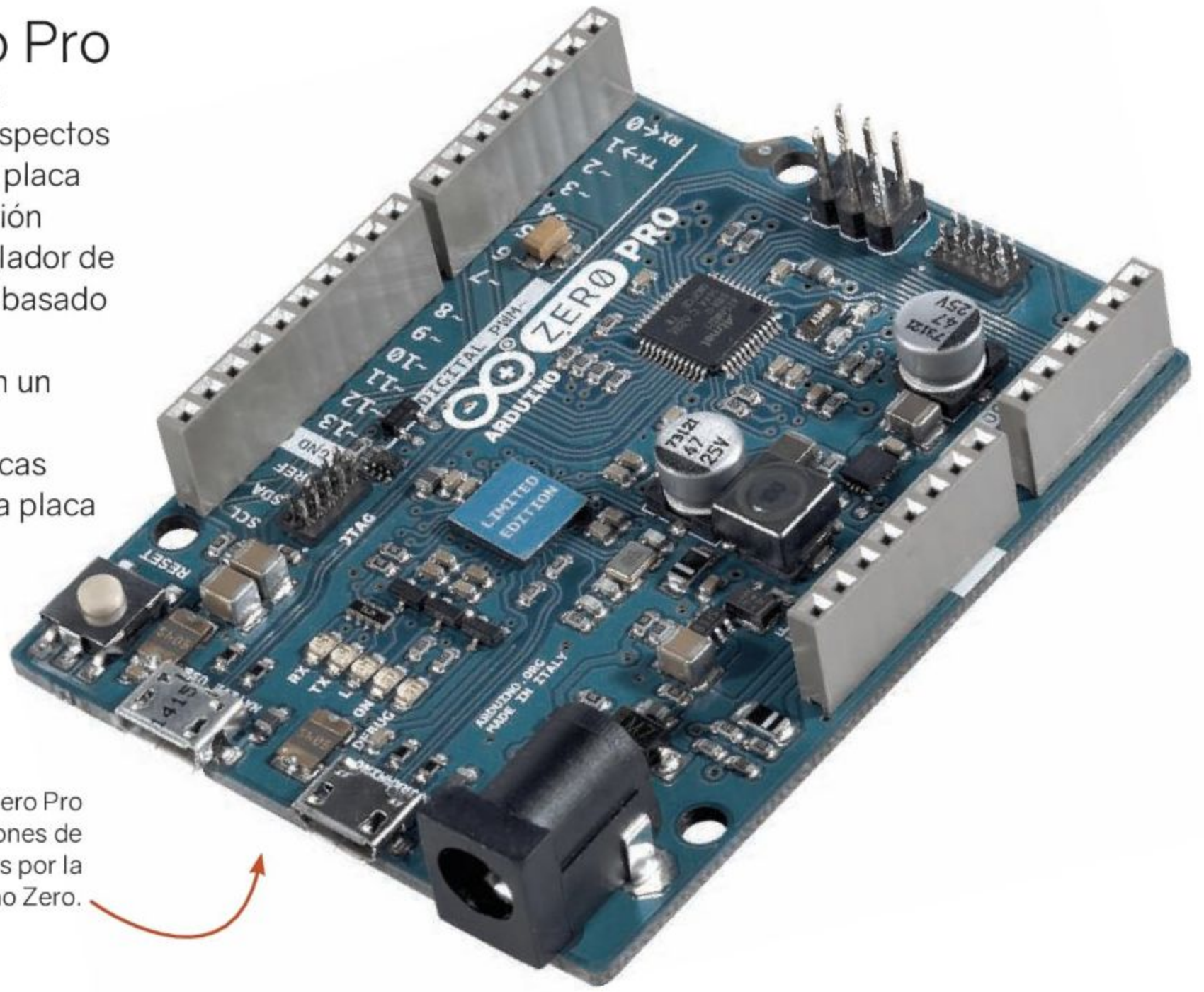
Esta placa es similar a la Arduino UNO, pero, en su arquitectura, utiliza un microcontrolador Atmel SAMD21 MCU de 48 MHz e integra un core ARM Cortex M0 de 32 bits.
En ella encontraremos 256 Kb de memoria flash, 32 Kb de SRAM y una EEPROM de más de 16 Kb por emulación. Ofrece 14 pines E/S digitales, 6 entradas analógicas para un canal ADC de 12 bits y una salida analógica para DAC de 10 bits.

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

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.

Arduino Zero Pro

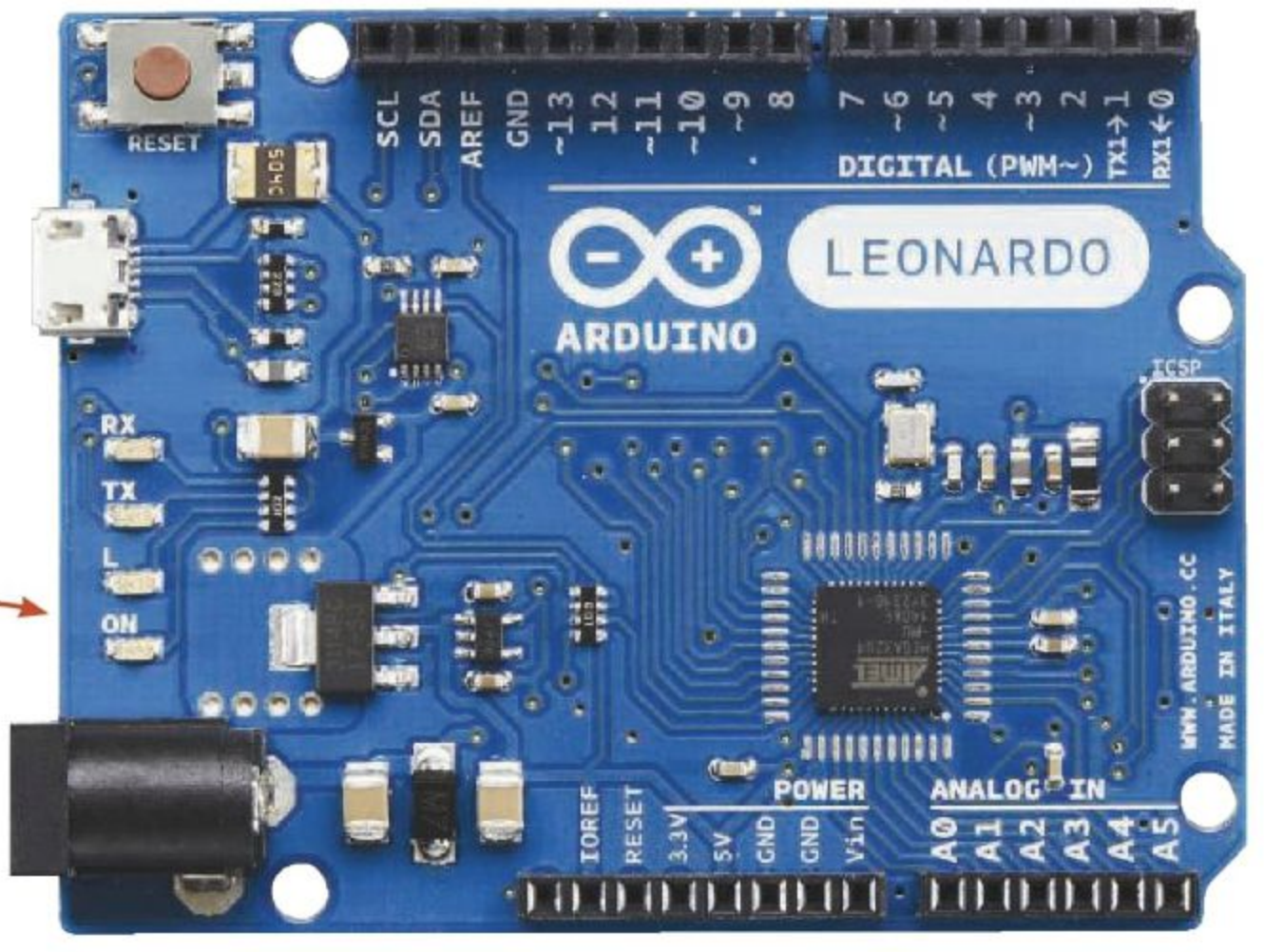
Se trata de una versión mejorada en muchos aspectos en comparación con la placa Arduino Zero. Esta opción integra un microcontrolador de 32 bits, el Cortex M0+ basado en ARM, que corre a 48 MHz, y se integra en un Atmel SAMD21 MCU.
Sus demás características son similares a las de la placa Arduino Zero.



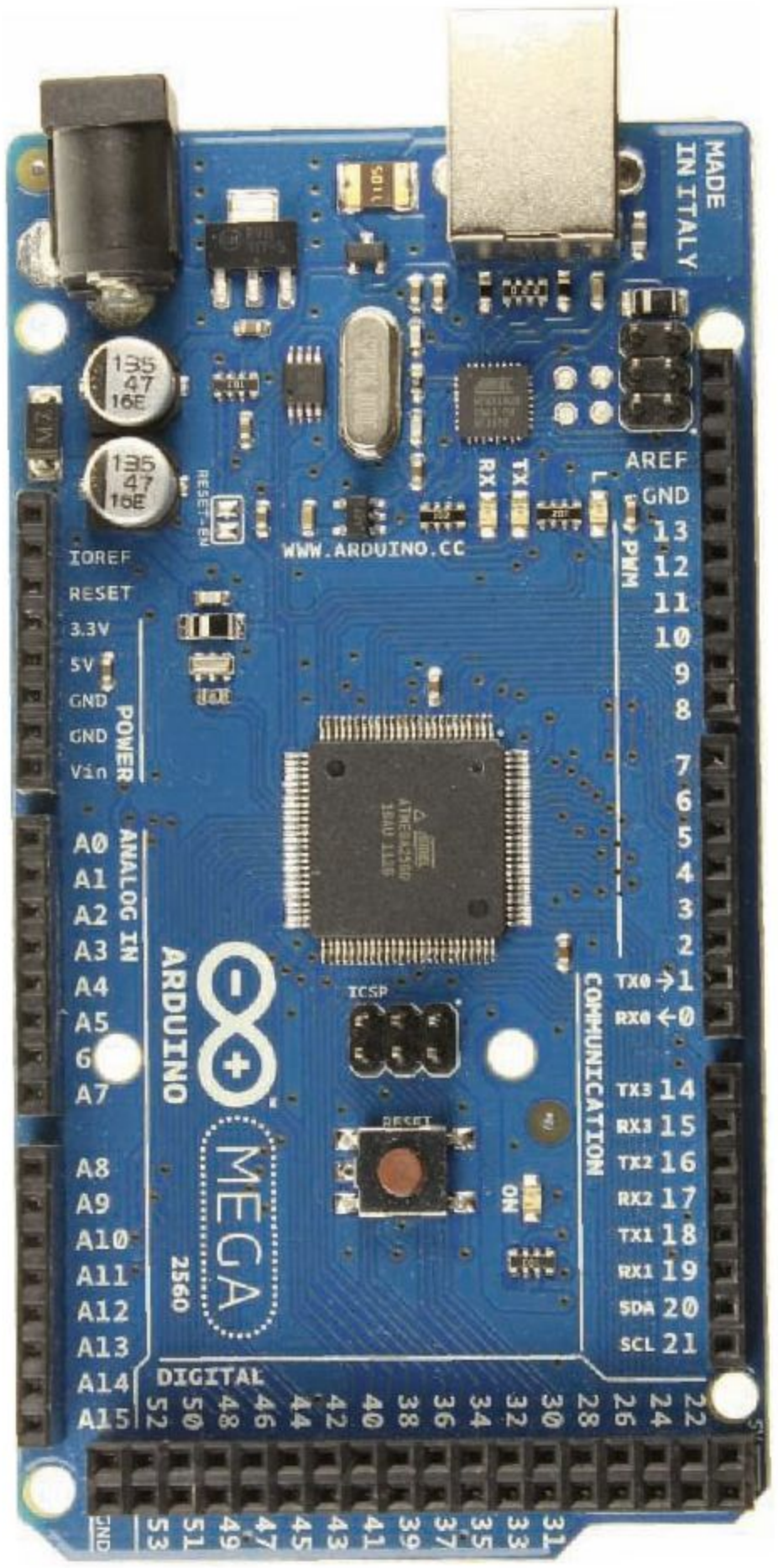
La Arduino Zero Pro mejora las prestaciones de cómputo ofrecidas por la placa Arduino Zero.

Arduino Leonardo

Esta placa puede considerarse como una mezcla entre la Arduino UNO y la Arduino Yún. Por un lado, posee las capacidades de almacenamiento de la Arduino UNO y, por otro, 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, y la EEPROM es de 1 Kb. Si la analizamos a nivel de voltajes, encontraremos que es igual a la Arduino UNO, pero nos entrega 20 pines digitales y 12 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 Mega

Sus prestaciones son similares a las de Arduino Due, pero se basa en una arquitectura AVR. Posee un microcontrolador ATmega2560 que trabaja a 16 MHz, con 5 V. 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 ofrece un buen número de entradas digitales, gracias a las que podremos incorporar diferentes módulos para potenciar nuestros proyectos.

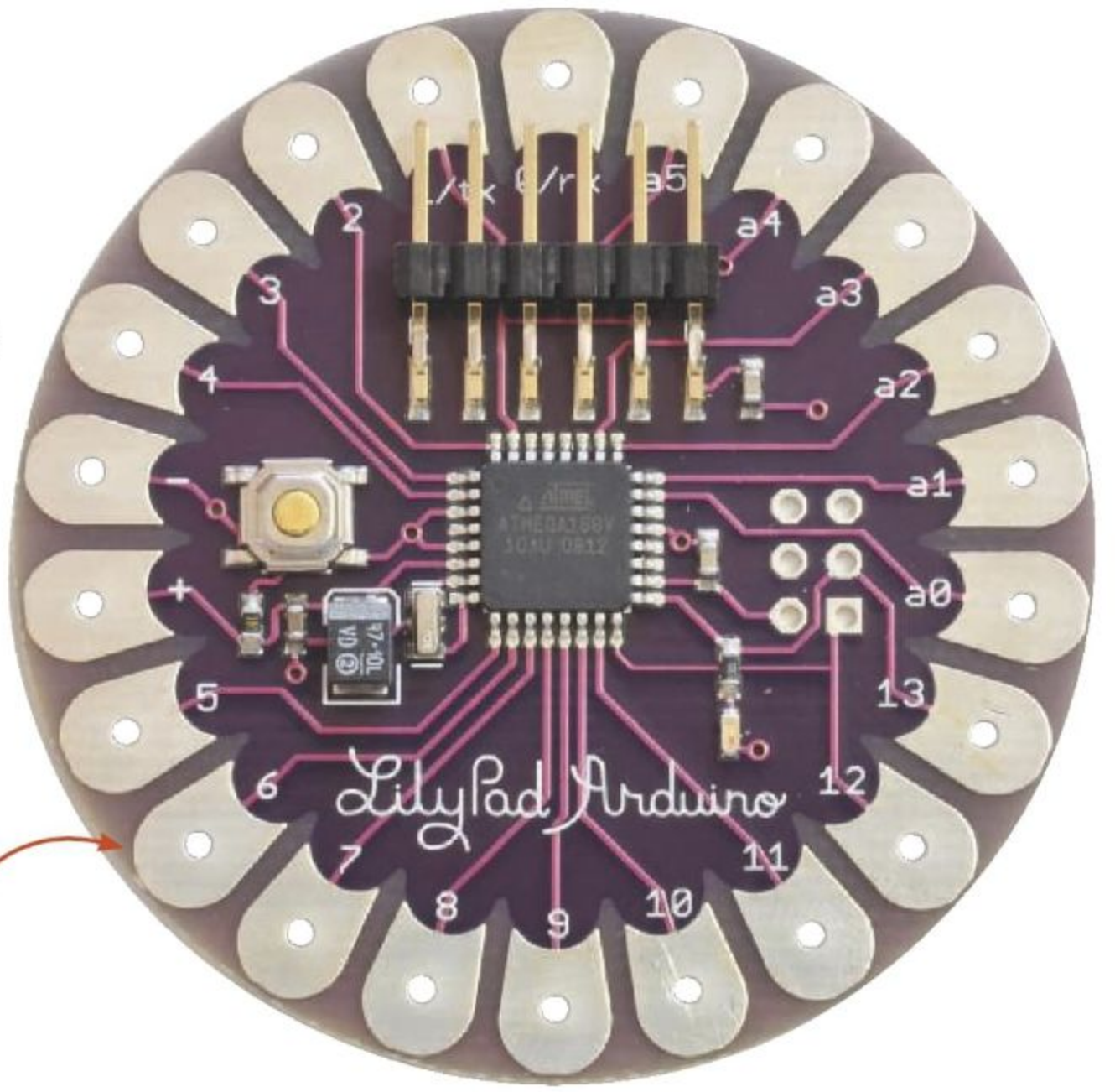
Arduino LilyPad

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. Trabajan a 8 MHz, a 2,7 V y 5,5 V, respectivamente.

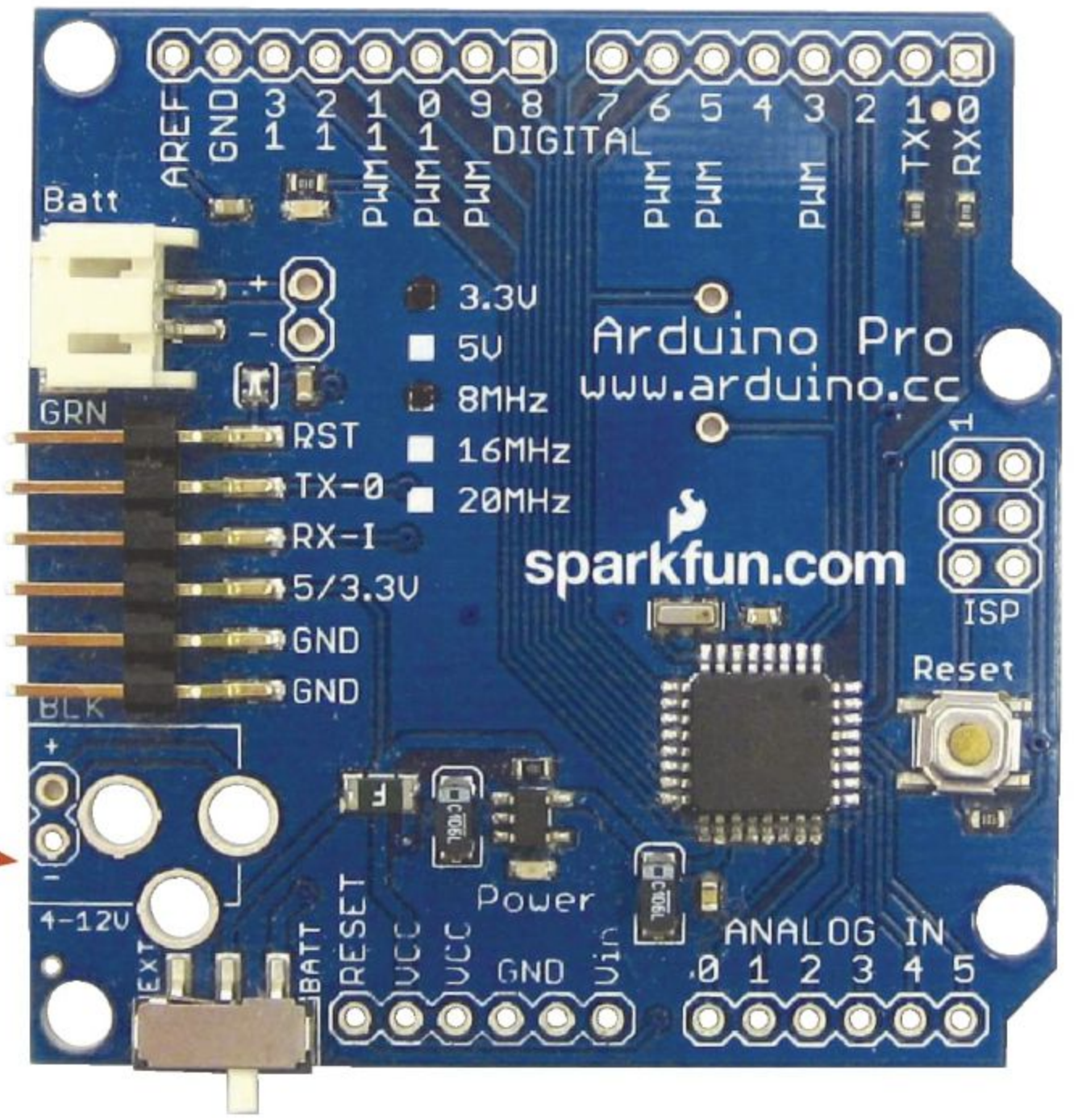
Posee 14 pines digitales y 6 analógicos, una memoria flash de 16 Kb, 1 Kb de SRAM y 512 bytes de EEPROM.



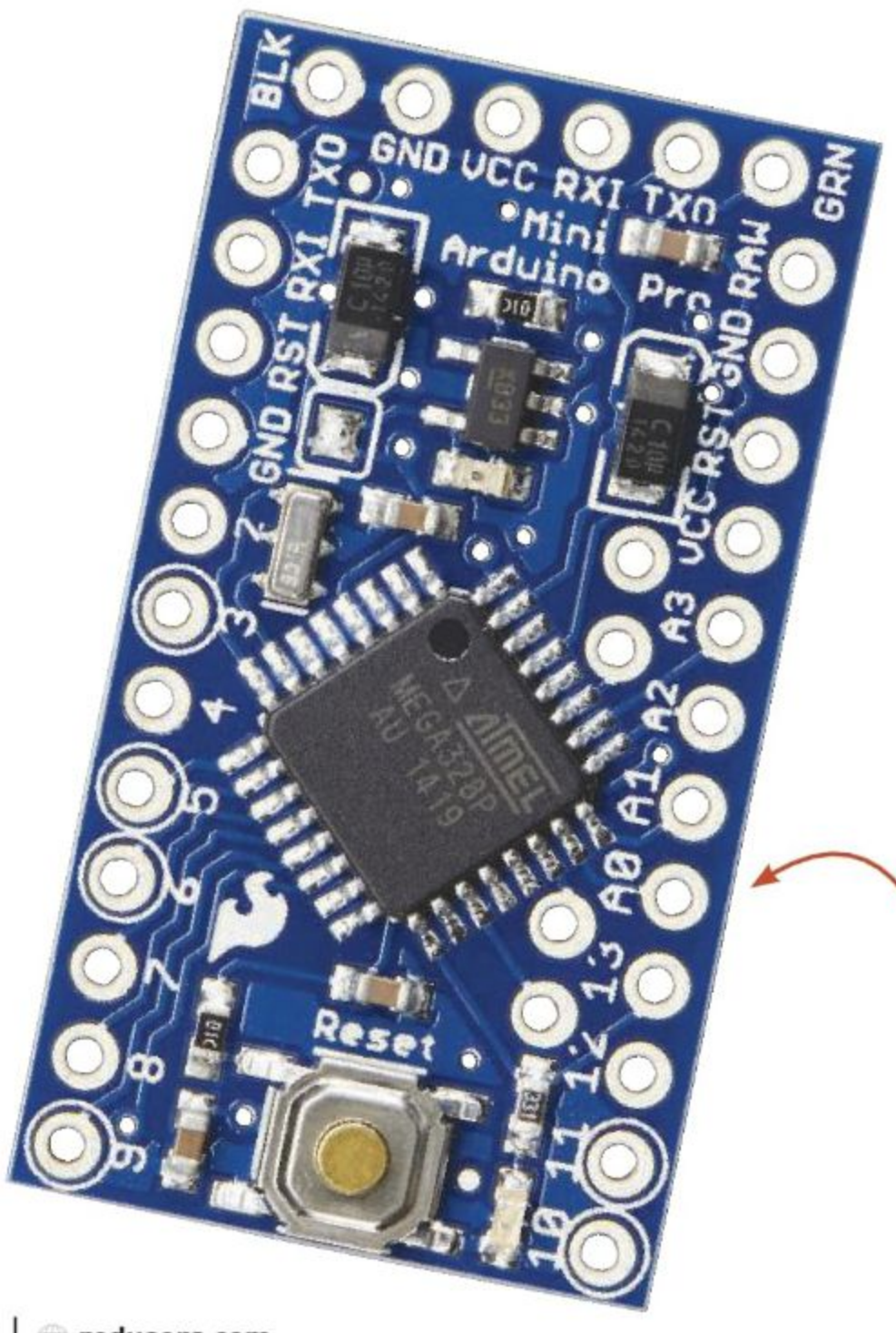
Arduino LilyPad posee una forma distinta, especial para ser incorporada en prendas o artículos que podemos utilizar a diario. Se usa en proyectos de tejidos inteligentes.

Arduino Pro

La placa Arduino Pro utiliza un microcontrolador ATmega168 o un Atmega328, con 8 MHz o 16 MHz. Ofrece 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.
No es una de las placas más potentes de 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 de 5 V incorporado, y se encuentra protegida contra inversión de polaridad.



Arduino Pro Mini

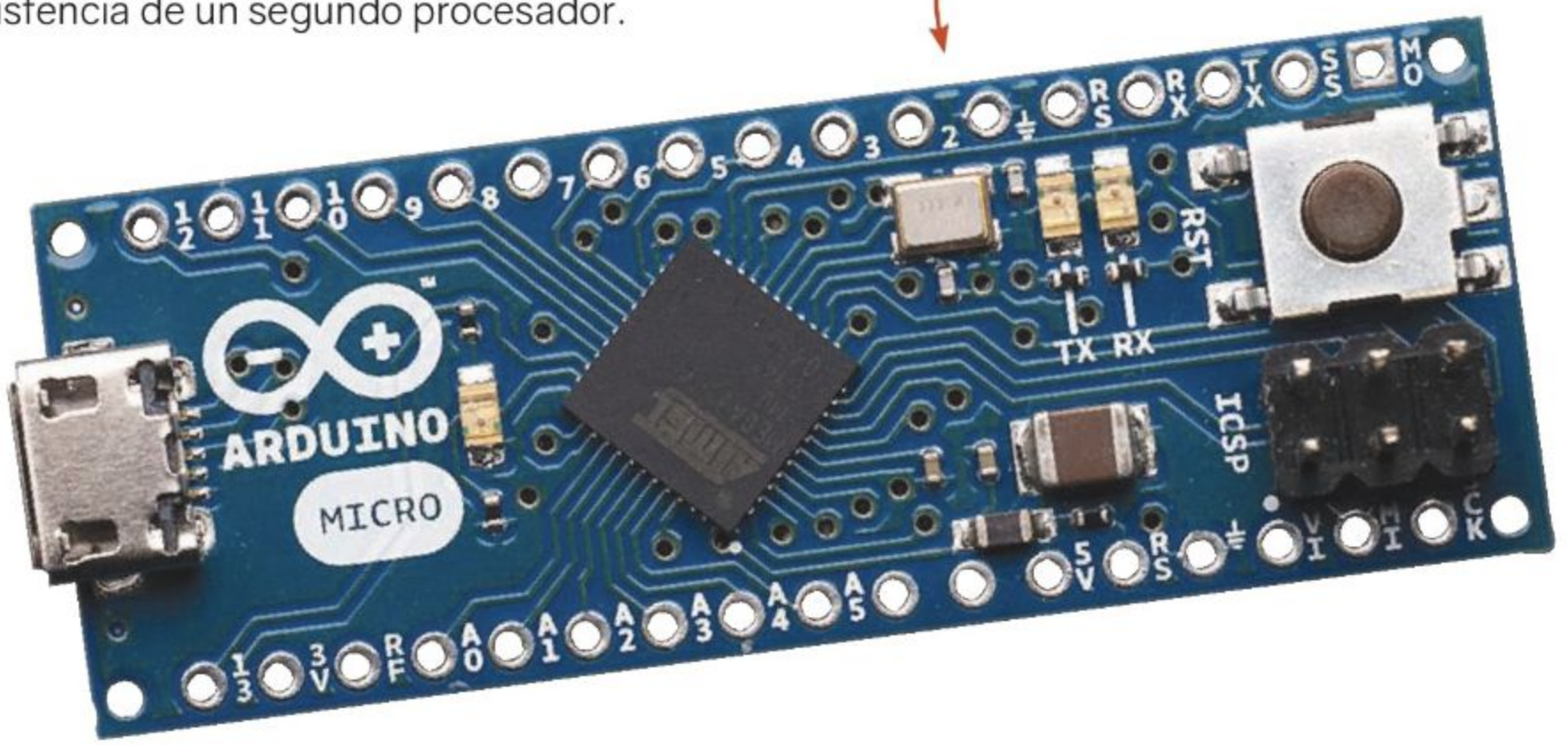
Se trata de la versión mínima de la placa Arduino Pro. Con un precio menor y un tamaño bastante reducido, aporta flexibilidad y, por lo tanto, portabilidad para proyectos específicos.
Debemos tener en cuenta que, para reducir su tamaño, se ha prescindido de características tales como el conector USB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

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

Arduino Micro

Ofrece una elevada autonomía junto con 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.

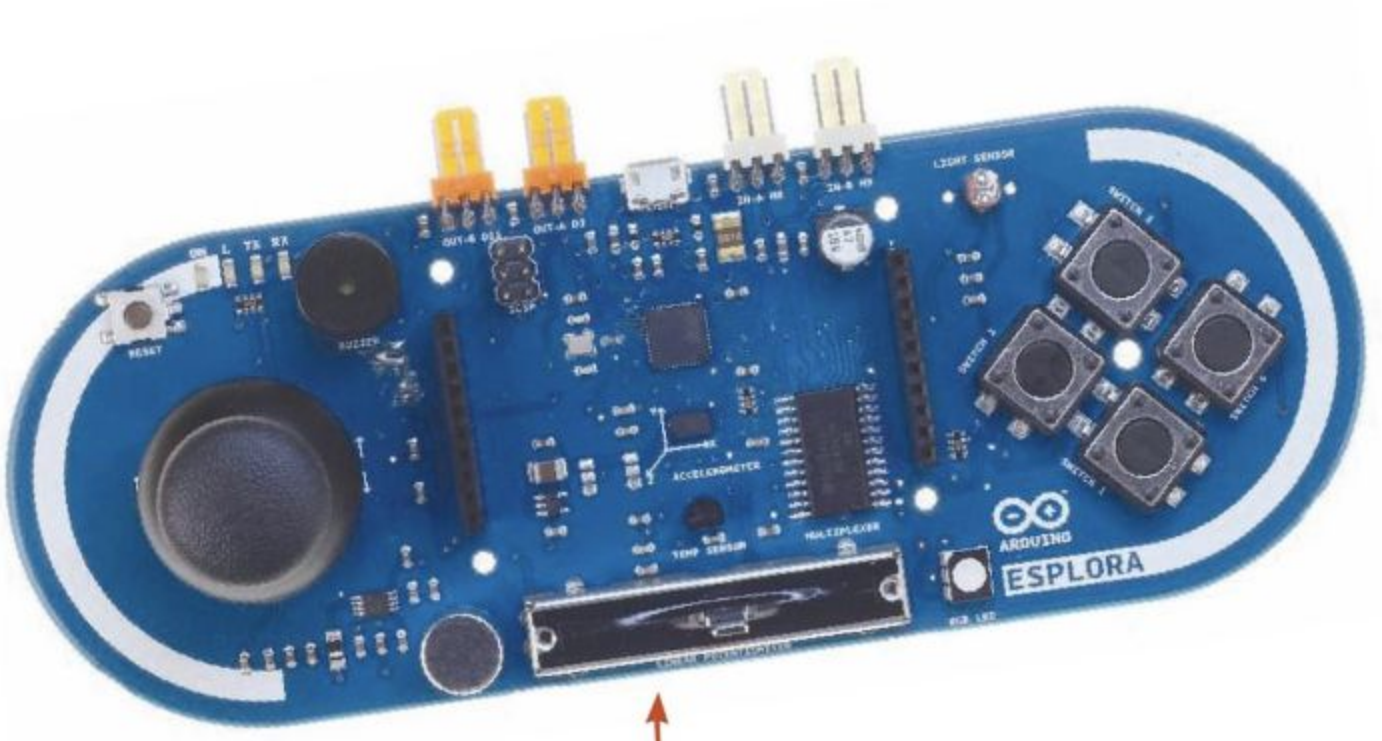
Arduino Micro ofrece una potencia similar a la placa Arduino Leonardo, pero en un formato bastante compacto, de tan solo 48 x 18 mm.



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. Por ejemplo, posee un microcontrolador ATmega32u4 que trabaja a 16 MHz, posee una SRAM de 2,5 Kb, 1 Kb de EEPROM y 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 y luz; también ofrece un zumbador, botones, joystick, micrófono y, por si todo esto fuera poco, se agrega un socket adecuado para conectar una pantalla TFT LCD.

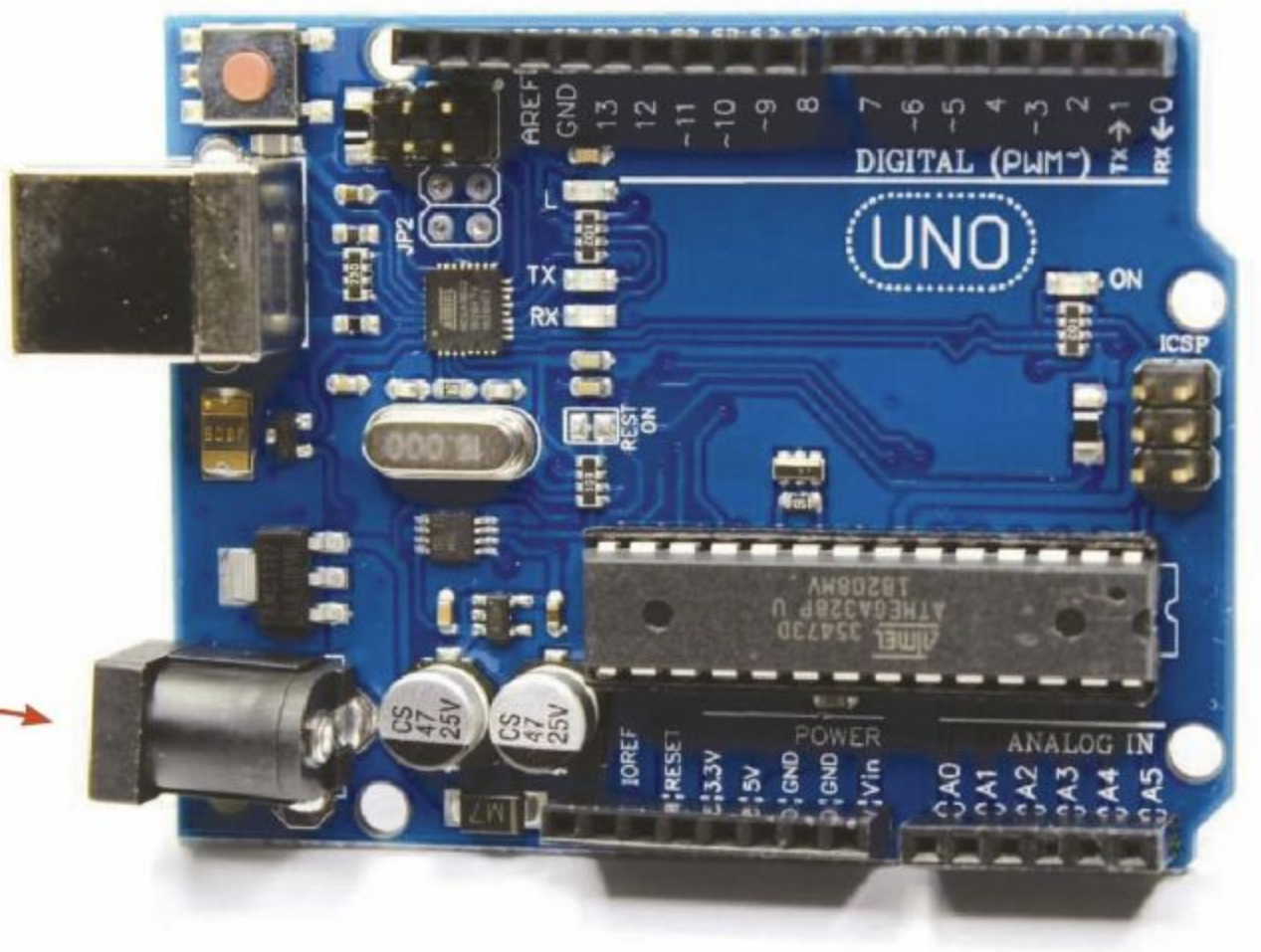


Arduino Esplora incorpora una serie de sensores y también una forma distinta de las demás placas Arduino.

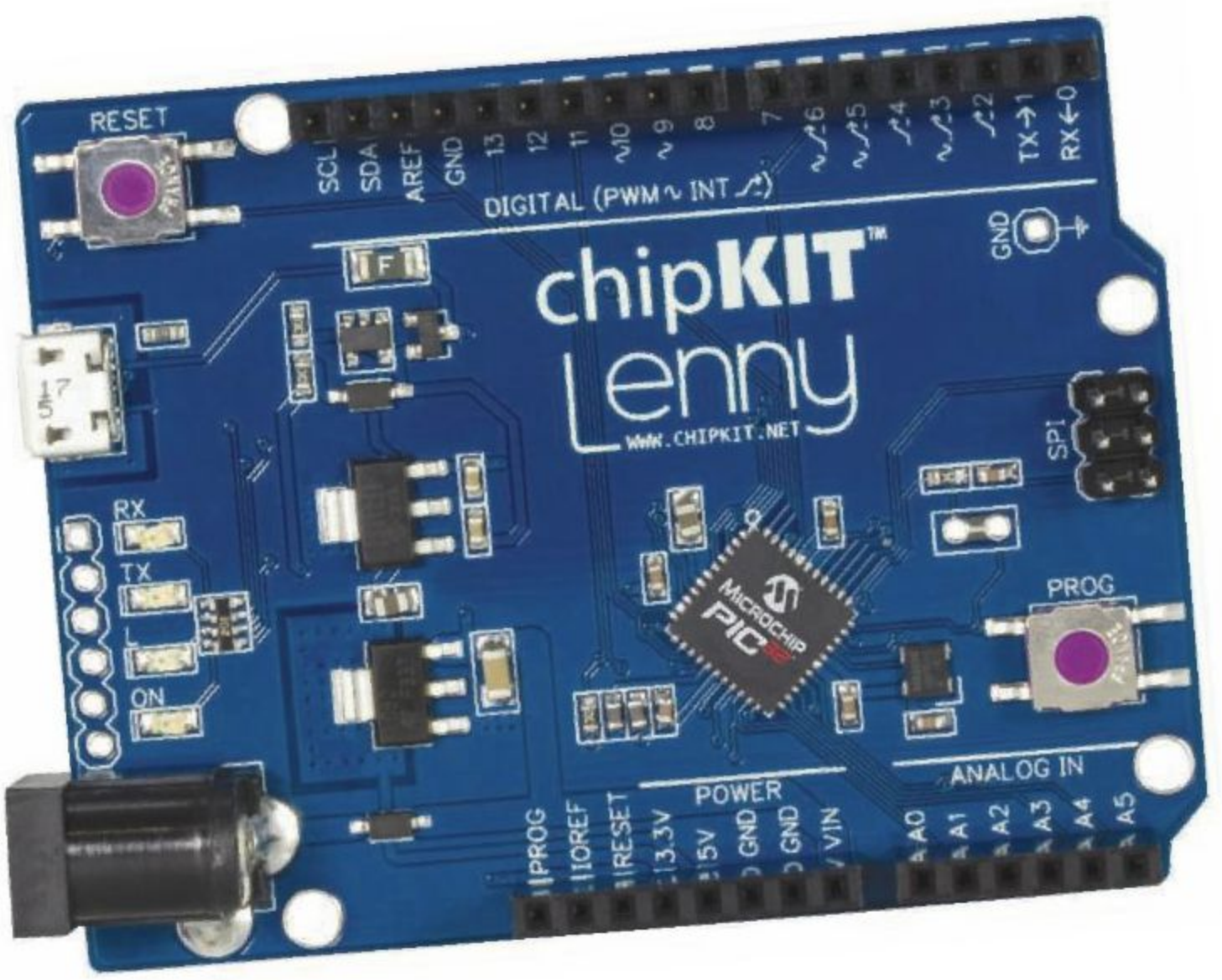
Placas no oficiales

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, de modo que únicamente podremos utilizar el IDE para programarlas. La elección de una u otra dependerá de las necesidades de nuestros proyectos.



Placa no oficial, compatible con Arduino UNO.



Las placas no oficiales o compatibles no pueden estar registradas bajo el nombre de Arduino. Son diseñadas y fabricadas por otras compañías y se trata de derivados que han salido para cubrir necesidades específicas.

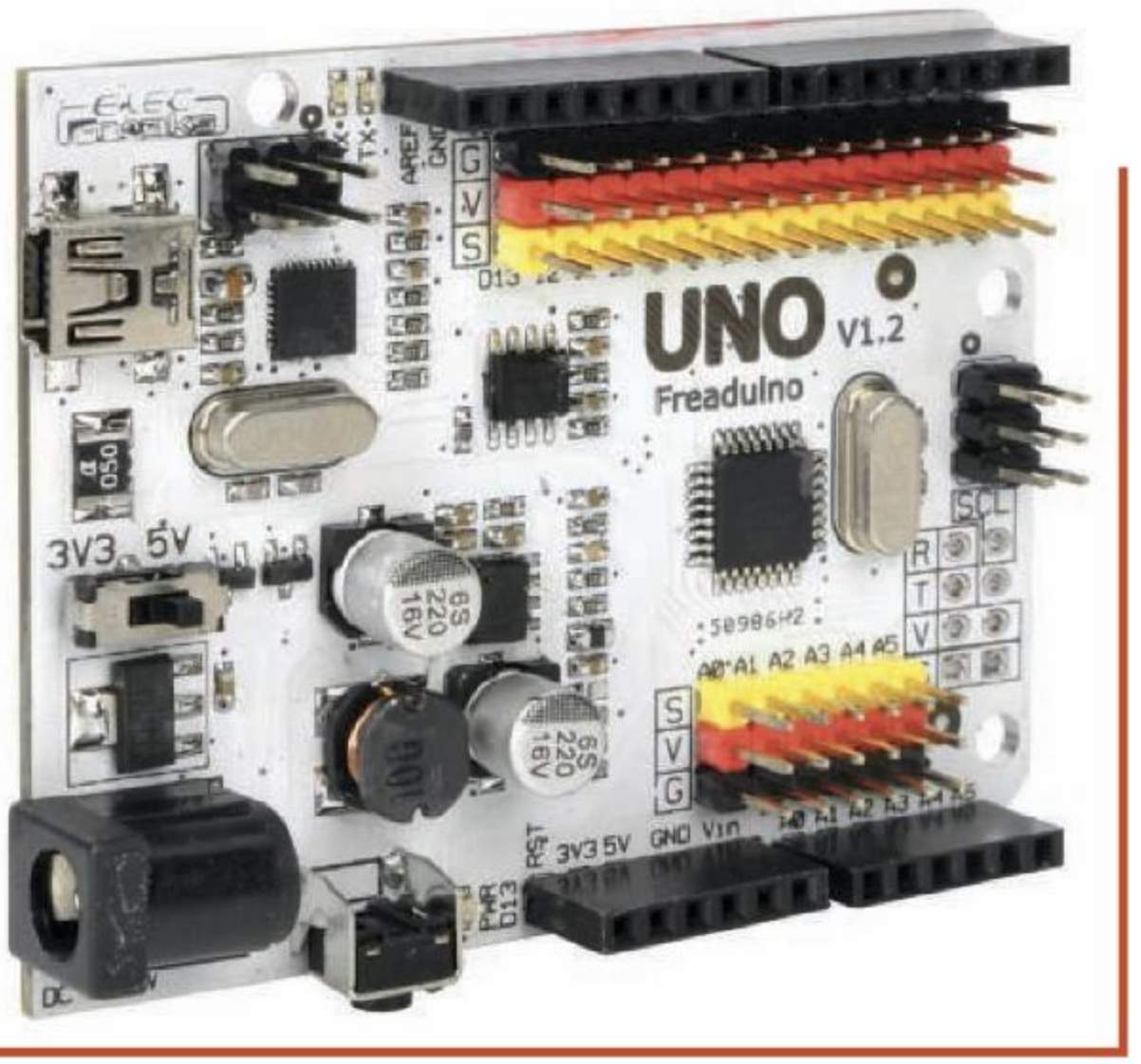
A continuación, mencionaremos algunas de las placas no oficiales existentes.

EJEMPLOS DE ALGUNAS 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 reducidos.
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; 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 y 64 Kb de RAM.
Freaduino	**Elecfreaks**	Su creación se basa en la placa Arduino UNO, y es compatible en un 100% a nivel de hardware y de software.

Freaduino UNO

Esta placa tiene el tamaño de un Arduino UNO, y es compatible al 100% con el IDE de Arduino, de modo que se reconoce como un Arduino UNO. Tiene el microcontrolador del Arduino Nano pero en un encapsulado más pequeño, por lo que ocupa mucho menos espacio. Cuenta con una gran cantidad de pines listos para conectar sensores, servos o cualquier otro tipo de módulo que use conectores de tres pines: VCC, GND y DATOS.



Usos de Arduino

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.

Ya conocimos algunas de las placas Arduino oficiales disponibles en el mercado y sus características generales, y también realizamos un repaso por algunas placas no oficiales. 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 de proyectos de automatización en industrias, gracias a que funciona como controlador lógico programable.

Entrenamiento electrónico

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

Domótica

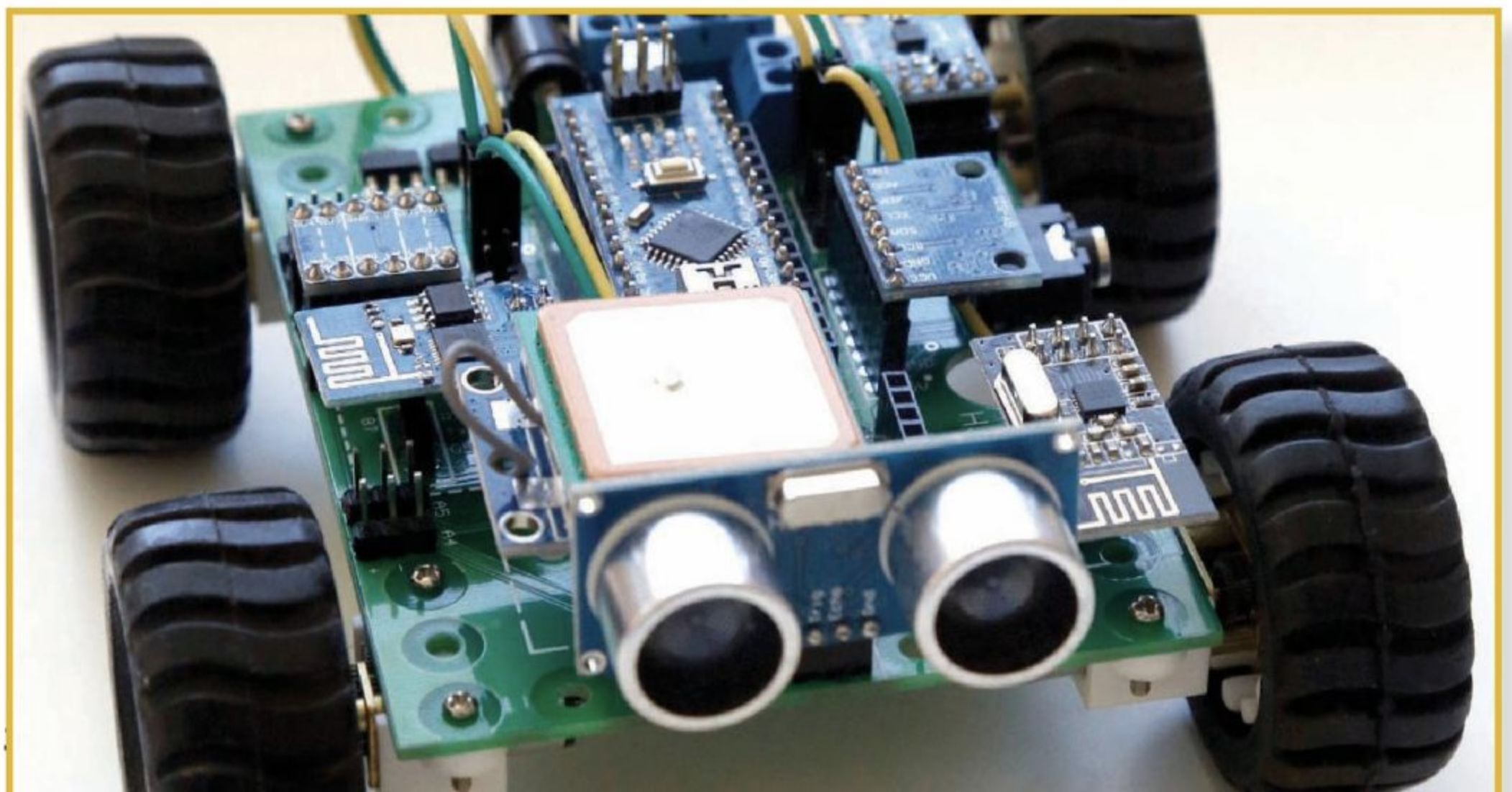
Los sistemas de control de domótica son más accesibles y pueden ser creados por entusiastas, gracias a las placas Arduino.

Arte

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

Prototipado

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



Eficiencia energética

Desde el control del consumo energético en nuestro hogar hasta medidores de consumo, todo puede crearse 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 lo cual podemos agregar a nuestros proyectos algunos de los sensores disponibles en el mercado.

Robótica

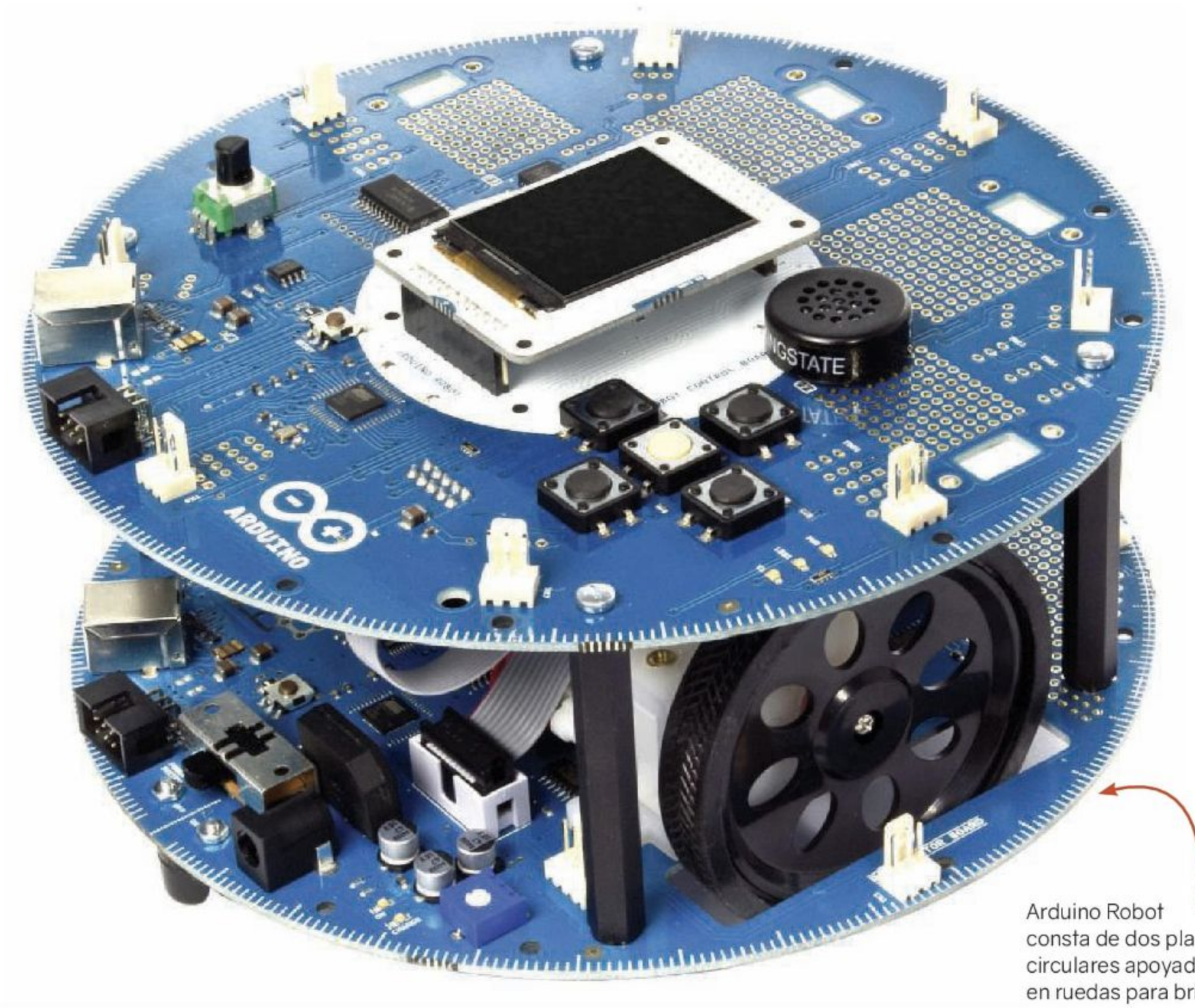
Por supuesto, la robótica no podía quedar fuera de este pequeño listado. Encontramos diversos proyectos que hacen uso de Arduino para construir y programar robots.

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 brindar movilidad.

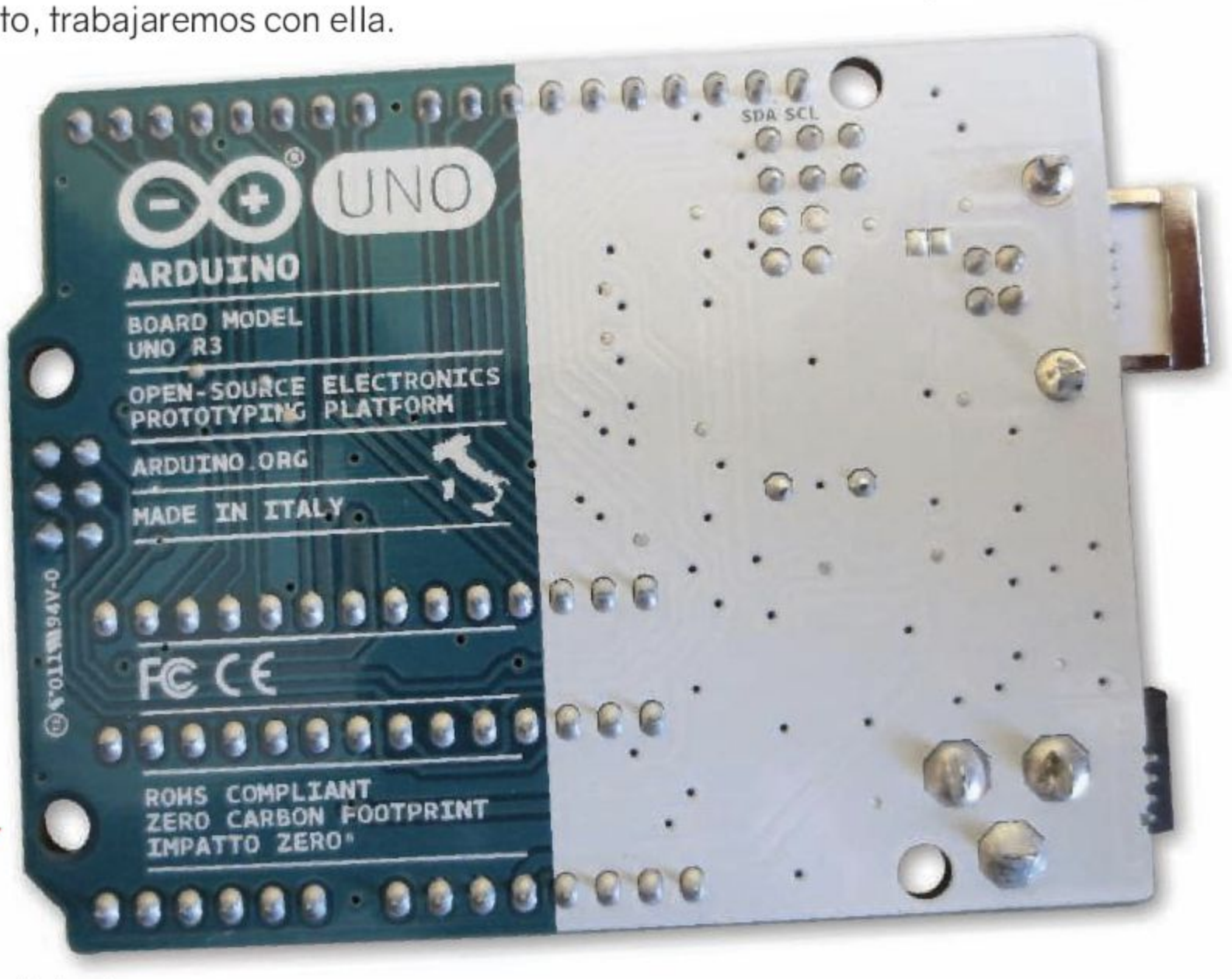
¿Qué necesitamos?

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

Para comenzar a trabajar con Arduino precisamos, en primer lugar, una placa de pruebas. Podemos utilizar cualquiera de la gama de placas oficiales o, también, una compatible. En este caso recurriremos a Arduino UNO, por ser la más usada. Pero Arduino UNO no puede trabajar sola: se requiere contar con una serie de componentes electrónicos adicionales, cada uno de los cuales nos servirá para una tarea específica y, en conjunto, para lograr proyectos completos. Para analizar en detalle lo que vamos a utilizar, los dividiremos en **placa de desarrollo** y **elementos adicionales**; estos últimos los iremos conociendo en detalle en las siguientes secciones.

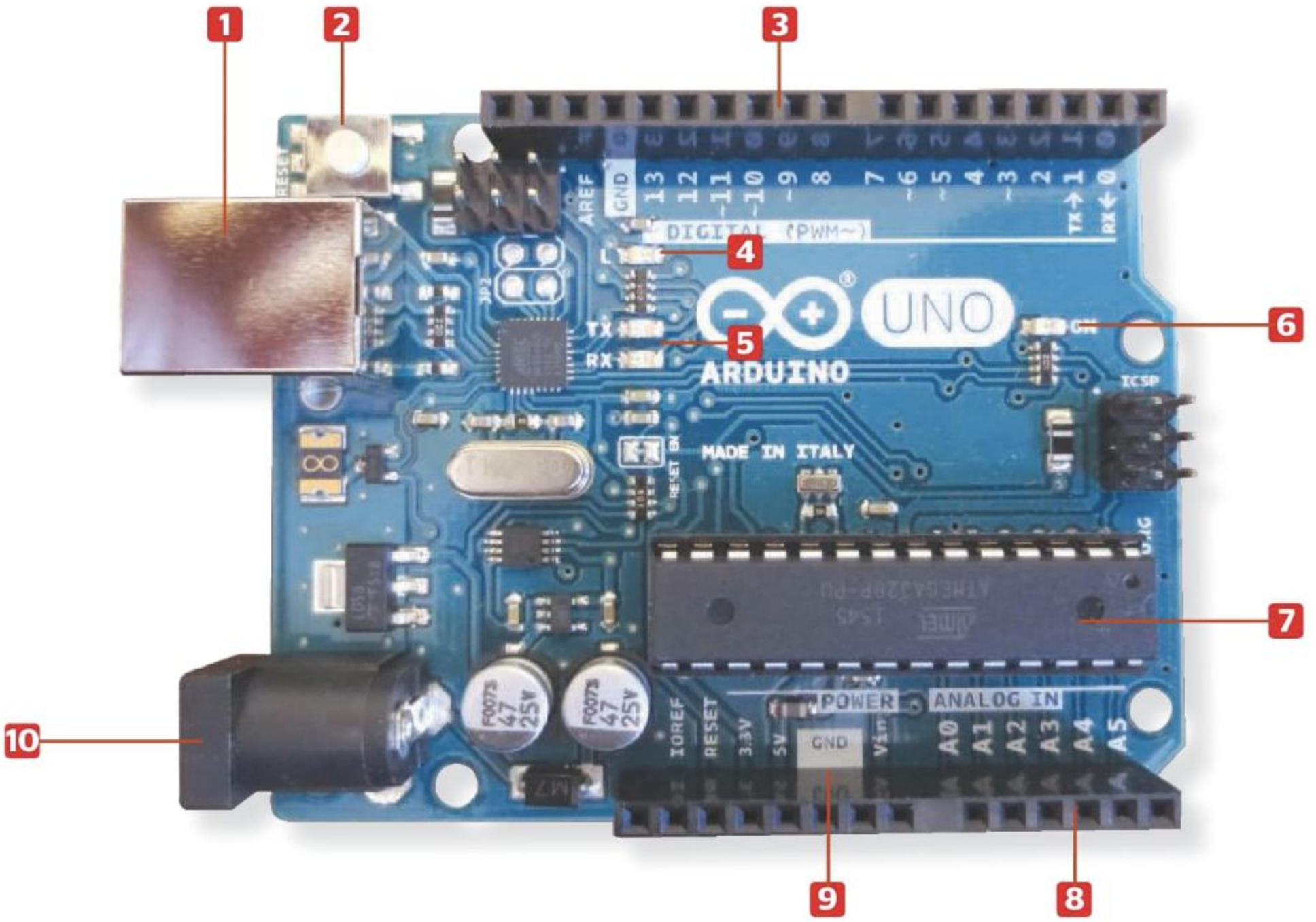
Placa de desarrollo

Como mencionamos, para iniciarnos en Arduino utilizaremos la placa Arduino UNO. Es muy versátil y nos ofrece todo lo necesario para desarrollar proyectos tanto sencillos como avanzados. En la actualidad, la última versión de esta placa es la **R3**, por lo tanto, trabajaremos con ella.



Si damos vuelta 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.
Tomemos la placa y ubiquémosla teniendo en cuenta la posición correcta del logo de Arduino; de esta forma, describiremos los componentes en orden. En la siguiente **Guía visual** analizamos los elementos más relevantes:



1 **Puerto USB**: se trata de un puerto USB tal como el que encontramos en dispositivos como una impresora. Lo usaremos para entregar energía a la placa Arduino mientras estamos trabajando con la PC. También es necesario para cargar los bocetos o sketches; en definitiva, es la forma de comunicación de la placa con la computadora.

2 **Botón de reinicio**: es un pequeño botón que sobresale de la placa Arduino, cuya función es permitirnos resetear el microcontrolador ATmega; de este modo eliminaremos lo que hayamos cargado y podremos comenzar con un nuevo proyecto o sketch. Resulta bastante útil y siempre debemos tenerlo en cuenta pues, al principio, podemos equivocarnos bastante mientras cargamos los bocetos.

3 **Clavijas digitales**: pueden usarse, por ejemplo, para **digitalRead()** o **analogWrite()**, entre otras opciones.

4 **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.

5 **LED TX y RX**: son LEDs que están perfectamente indicados en la placa Arduino y se utilizan para verificar que existe comunicación entre esta y la computadora. La forma en que verificamos la comunicación es esperando a que parpadeen mientras cargamos el código generado en el IDE de Arduino o cuando se efectúa una comunicación en serie.

6 **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.

7 **Microcontrolador ATmega**: sin duda, se considera el corazón de la placa Arduino UNO. Es un microcontrolador creado por ATmel; para el caso de esta placa se trata del ATmega328P-PU.

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

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

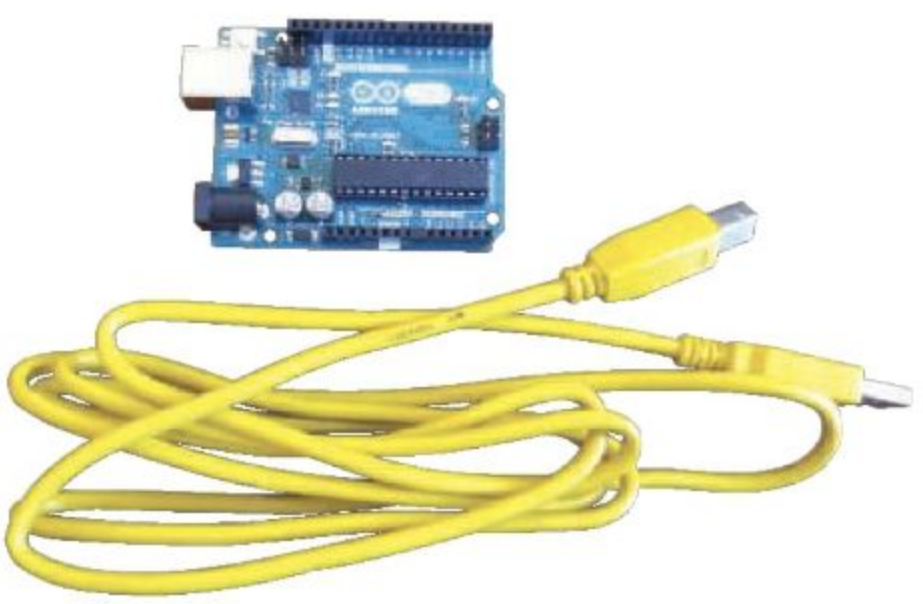
10 **Conector de corriente**: en el primer punto conocimos una forma de energizar nuestra placa Arduino, mediante el puerto USB. Pero este puerto solo proporcionará energía a la placa mientras la mantengamos conectada a la computadora. Cuando esto no suceda, utilizaremos el conector de corriente para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

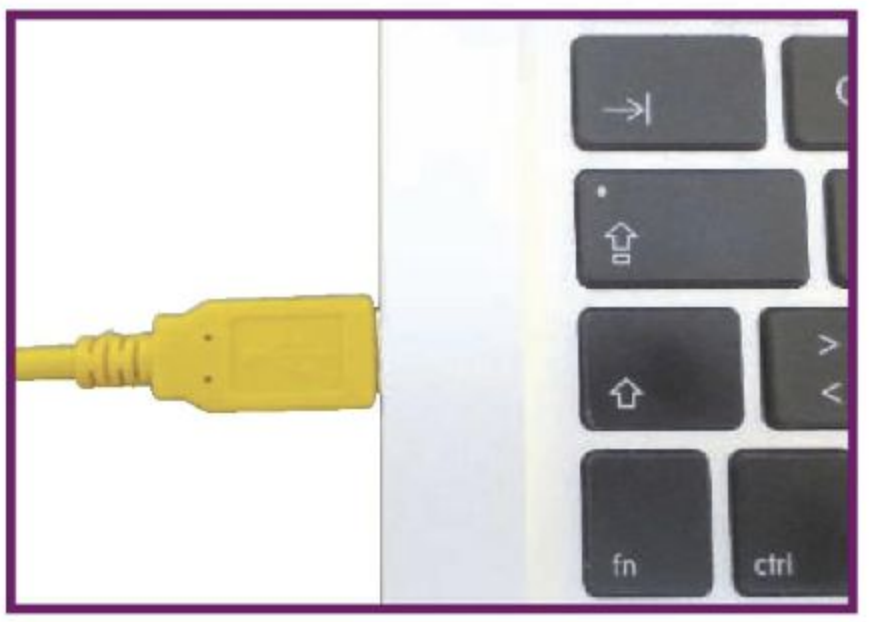
Prueba de conexión

Con los principales elementos de la placa Arduino UNO ya descriptos, podemos realizar el primer test de funcionamiento.

Para probar la placa Arduino UNO la conectamos a la computadora y verificamos que se le proporcione corriente. Tal como mencionamos en la **Guía visual** anterior, utilizaremos algunos de los LEDs que existen en la placa.

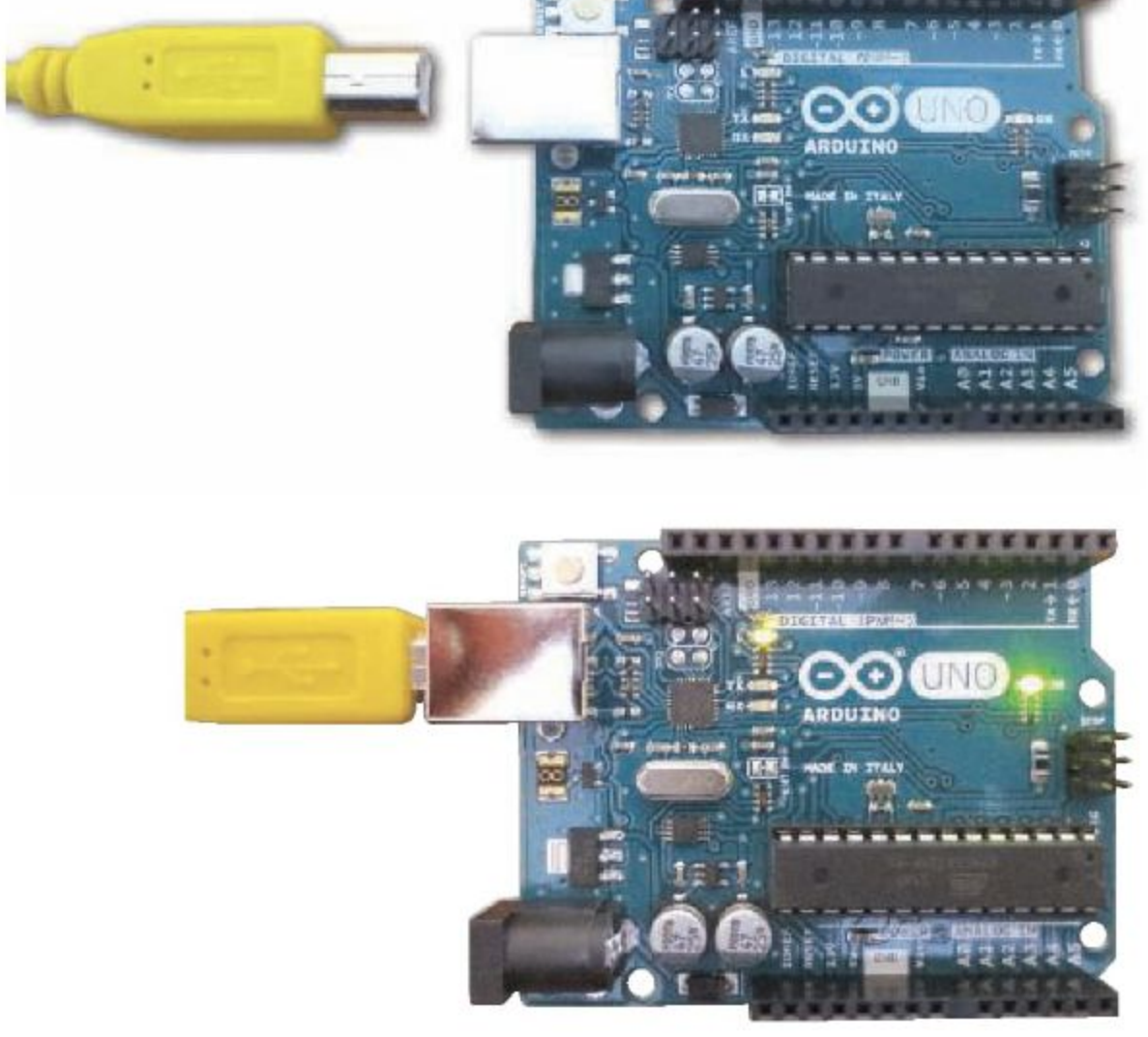
Prueba de conexión con Arduino UNO





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

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



3 Luego, conectamos el siguiente extremo a la placa Arduino UNO. Para hacerlo, usamos el puerto USB que se describe en el punto 1 de la Guía visual anterior.

4 Con la placa conectada a la PC, verificamos el LED de corriente y la clavija 13 LED. Si la conexión y el paso de corriente son correctos, el LED de corriente, indicado con el número 6 en la Guía visual, deberá mostrar una luz constante de color verde; mientras que la clavija 13 LED, indicada con el número 4 en la Guía visual, mostrará una luz parpadeante de color anaranjado.

Características de Arduino UNO

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

- **Microcontrolador: ATmega328**
- **Voltaje: 5 V**
- **Voltaje de entrada recomendado: entre 7 V y 12 V**
- **Límite de voltaje de entrada: entre 6 V y 20 V**
- **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**

Terminales digitales

Las placas Arduino 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, vemos los terminales digitales generales de una placa Arduino y sus usos.

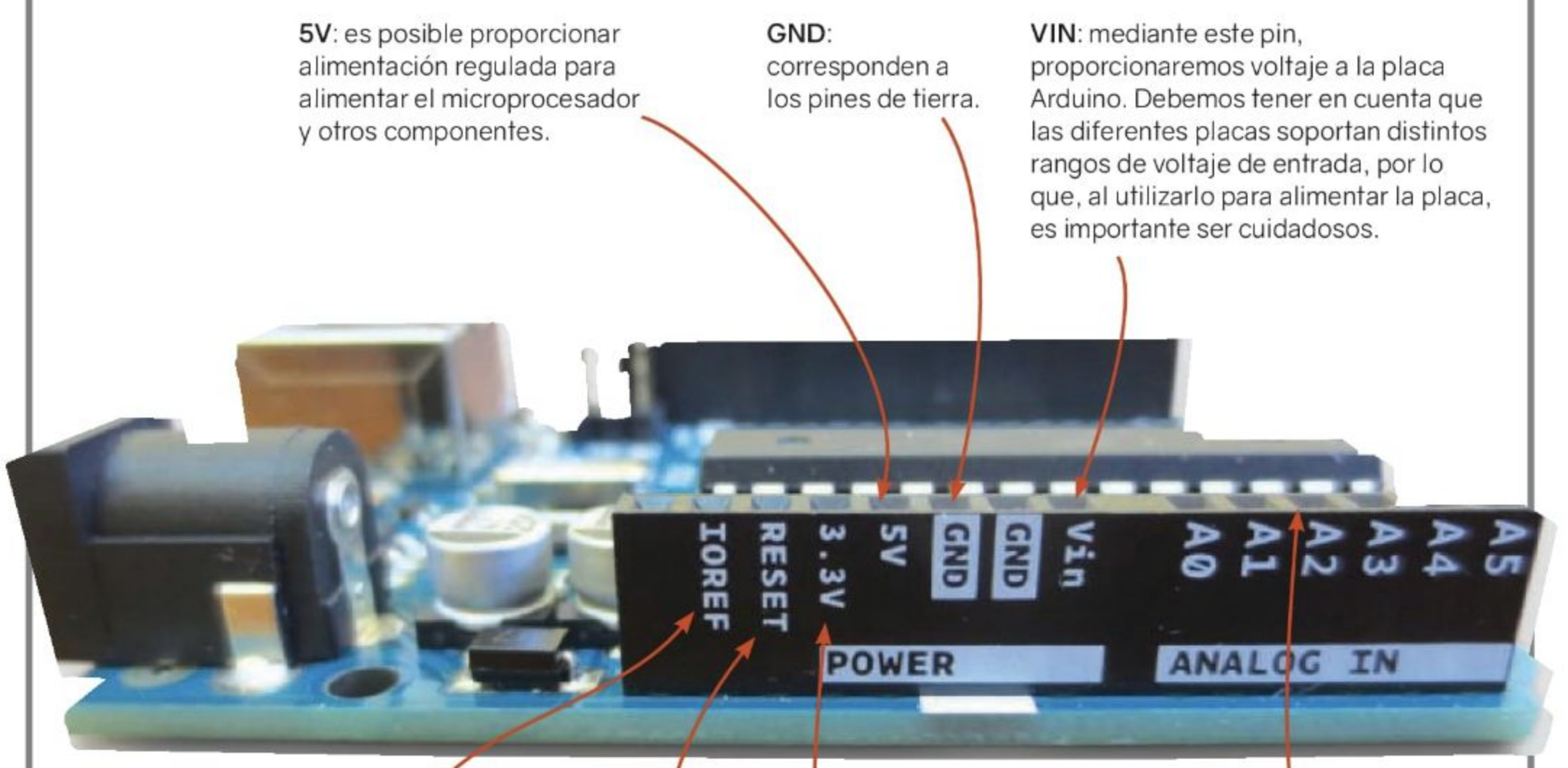
EJEMPLO DE PINES DIGITALES QUE PODEMOS ENCONTRAR EN PLACAS ARDUINO

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**	Entrega salidas PWM de 8 bits; para esto 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.

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

Otros terminales

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



Comunicación serie

La comunicación serie es muy importante porque gran parte de los protocolos utilizados actualmente son serie y además muchos dispositivos de comunicación inalámbrica usan la comunicación serie para hablar con Arduino como los módulos bluetooth y los módulos Xbee. También la comunicación serie es la que se usa generalmente para comunicar el Arduino con el la computadora.

Comparación con otras placas

Aunque ya elegimos 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 Arduino Leonardo, porque se trata de una placa que, a simple vista, tiene un gran parecido con la anterior. Veamos primero las similitudes. En realidad, se trata de dos placas que presentan el mismo tamaño y poseen igual cantidad de pines, que se disponen de la misma forma en ambas. Estos presentan idénticos requerimientos de alimentación, de 7 V a 12 V, con 6 V y 20 V como límites. Además, poseen la misma frecuencia de operación (16 MHz) e igual voltaje de operación (5 V). Analicemos 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 emplea un microcontrolador ATmega32u4 que sí integra la 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 de los pines 2 y 3.
Puede parecer una diferencia mínima, pero la distinta ubicación 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

Por el 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 Arduino UNO tenemos los pines 2 y 3, para las interrupciones 0 y 1. Arduino Leonardo ofrece cinco pines para interrupciones externas: 3, 2, 0, 1 y 7.

MEMORIA

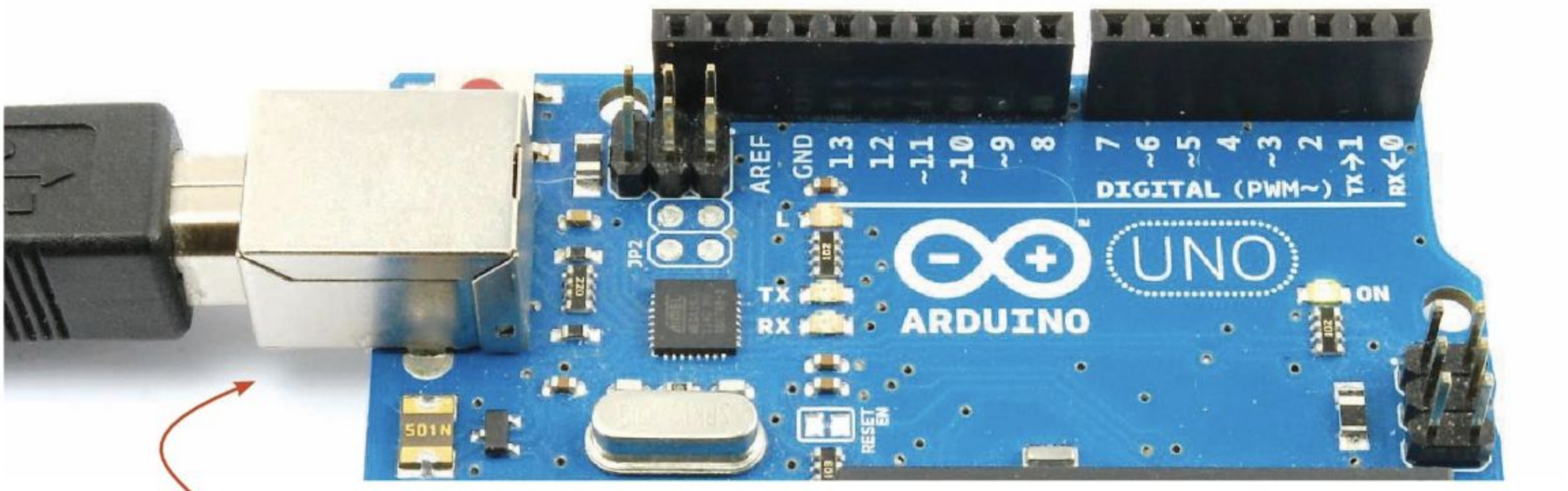
Según la información oficial sobre ambas placas, también hallamos una pequeña diferencia en el apartado de memoria. En Arduino UNO, el ATmega328 posee 32 Kb de memoria flash, con 0,5 Kb para el bootloader; también tiene 2 Kb de SRAM y 1 Kb de EEPROM. Por otra parte, en Arduino Leonardo, con un microprocesador 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 usa el conector ICSP, que se encuentra en uno de sus extremos. Al igual que ocurre con la comunicación I2C, encontraremos shields que no son compatibles con la tarjeta Arduino Leonardo.

CABLE DE CONEXIÓN

Un punto importante en 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 precisamos un cable A-micro B, como el que se usa para los teléfonos inteligentes.

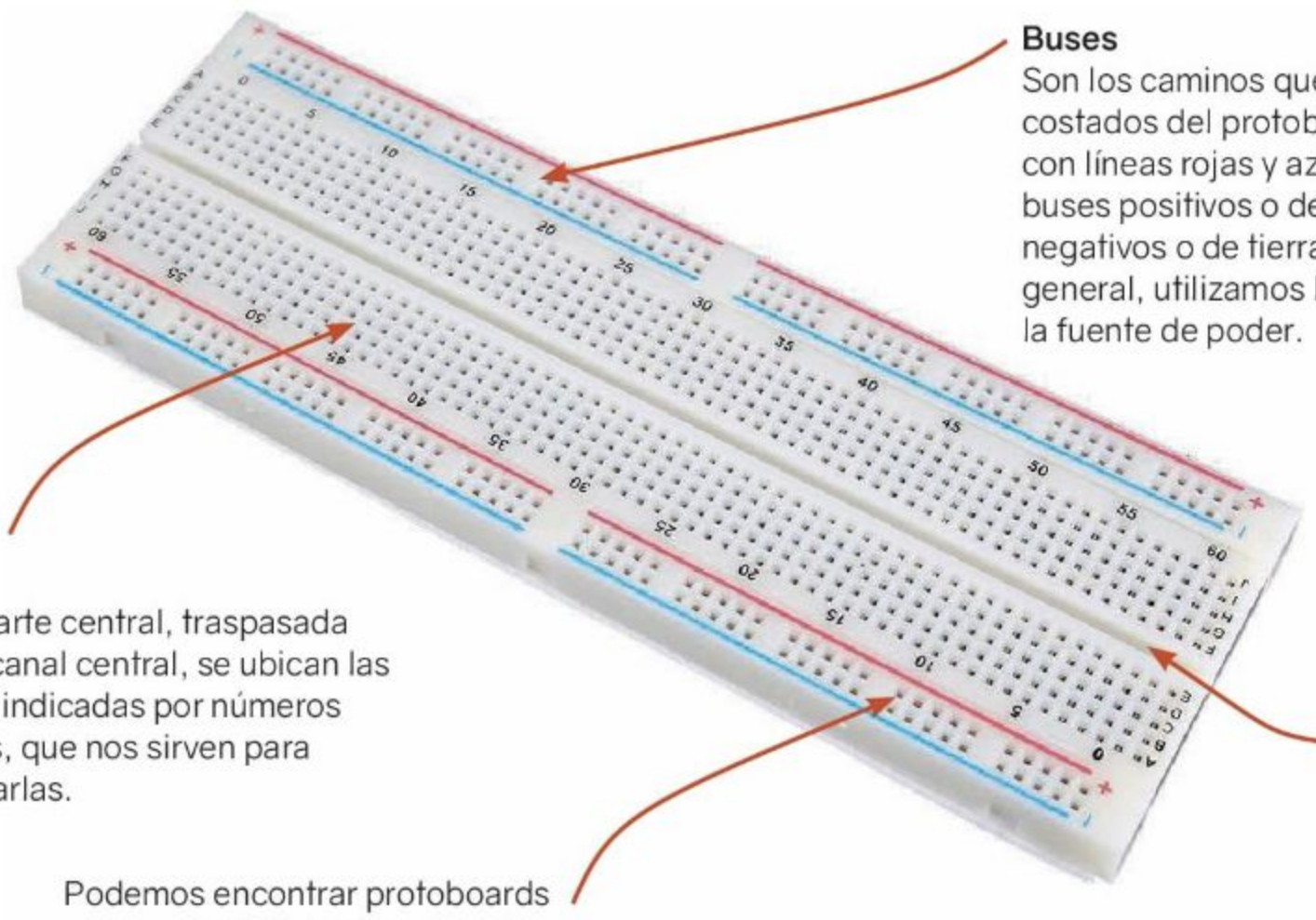


En Arduino UNO se utiliza un cable USB A-B, mientras que, para conectar Leonardo a la PC, será necesario un cable A–micro B.

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 de este último tipo.
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.

Pistas
En la parte central, traspasada por el canal central, se ubican las pistas, indicadas por números y letras, que nos sirven para localizarlas.

Canal central
Es la ranura o sección ubicada justo en el centro del protoboard, donde podemos colocar los circuitos integrados.

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 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 y circuitos integrados, entre otros.

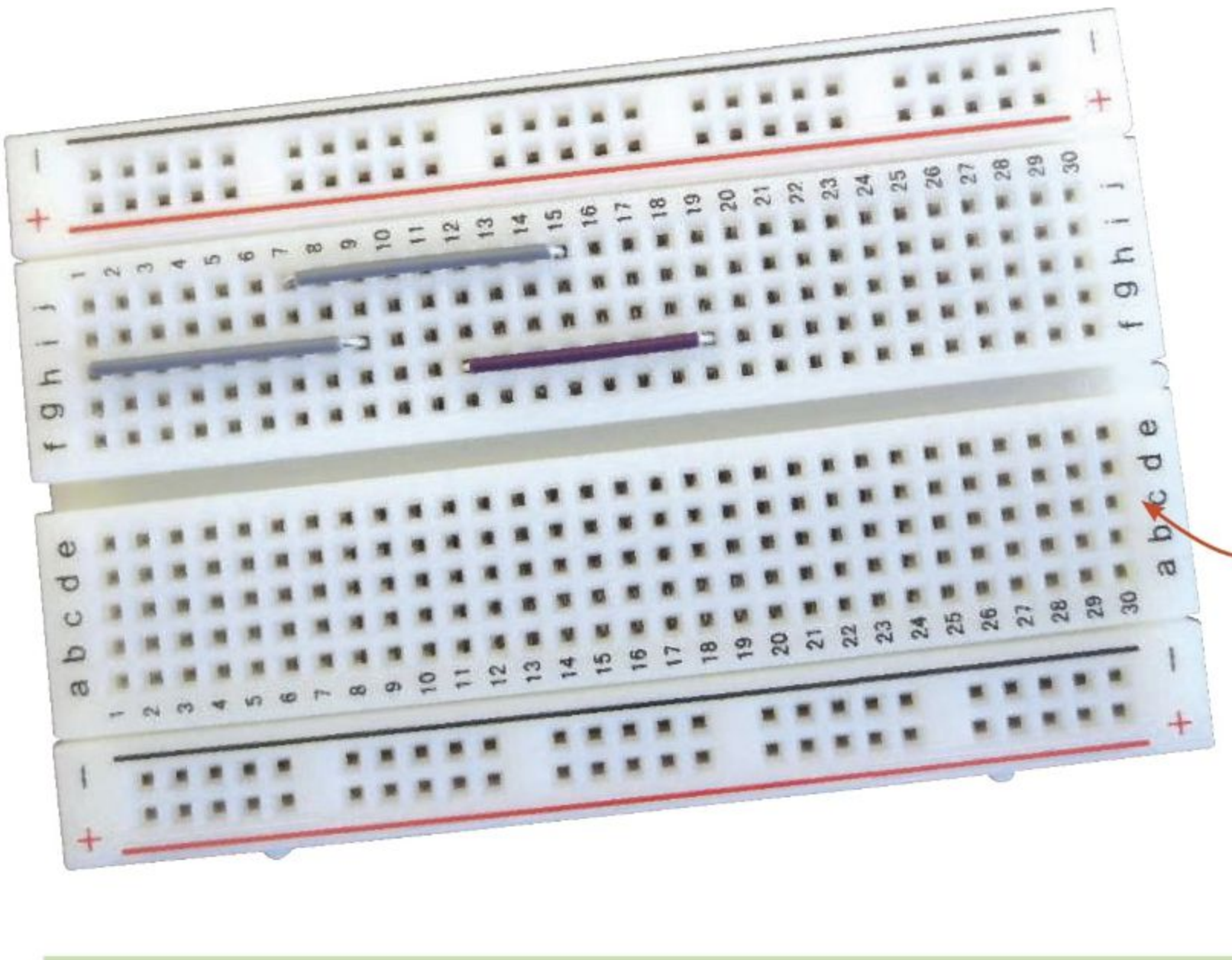
Para efectuar las conexiones, utilizaremos cables específicamente diseñados para ser usados en el protoboard, los que conoceremos más adelante.

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.

Estos cables se usan para lograr la conexión de componentes en el protoboard 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 modo que se pueden identificar en forma 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.



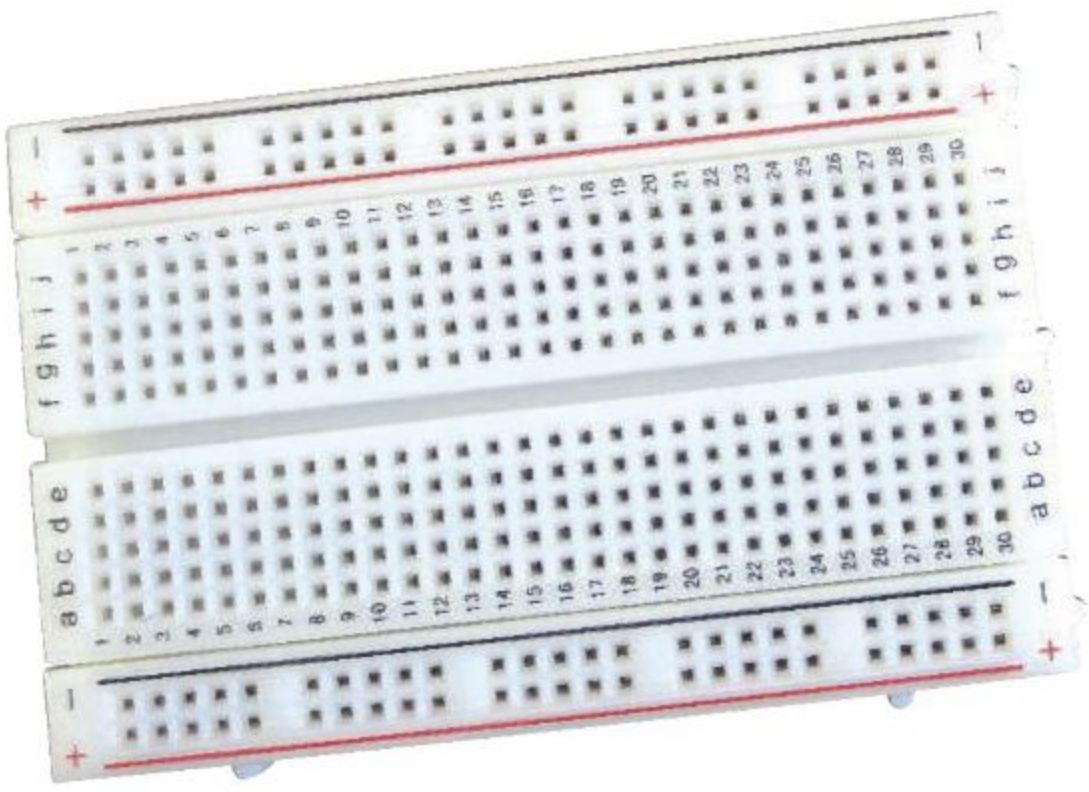
Es bueno probar la conexión de algunos cables de puente; de esta manera, nos acercaremos a la forma de trabajo que implementaremos más adelante y así podremos conocer el uso de estos elementos en el protoboard no energizado.

Protoboard no energizado

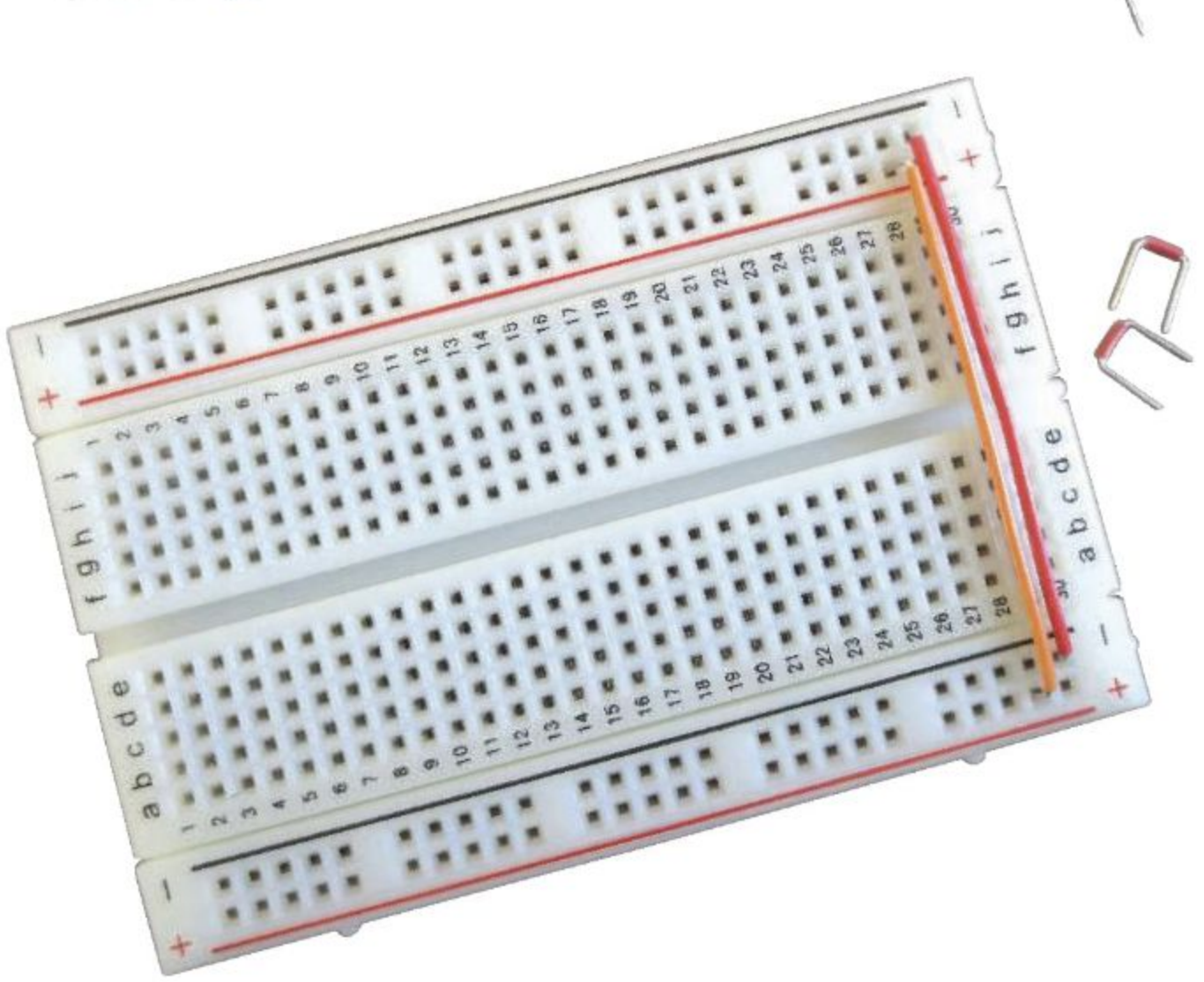
Es importante recordar que el protoboard no debe estar energizado en el momento de realizar las pruebas de conexión iniciales con los cables de puente. Si trabajamos con la placa energizada y efectuamos una conexión errónea, podríamos dañar la placa al exponerla a la circulación de corriente por caminos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

En este punto, aprenderemos a realizar una conexión básica, para lo cual vamos a usar los dos elementos que conocemos: el protoboard y los cables de puente.

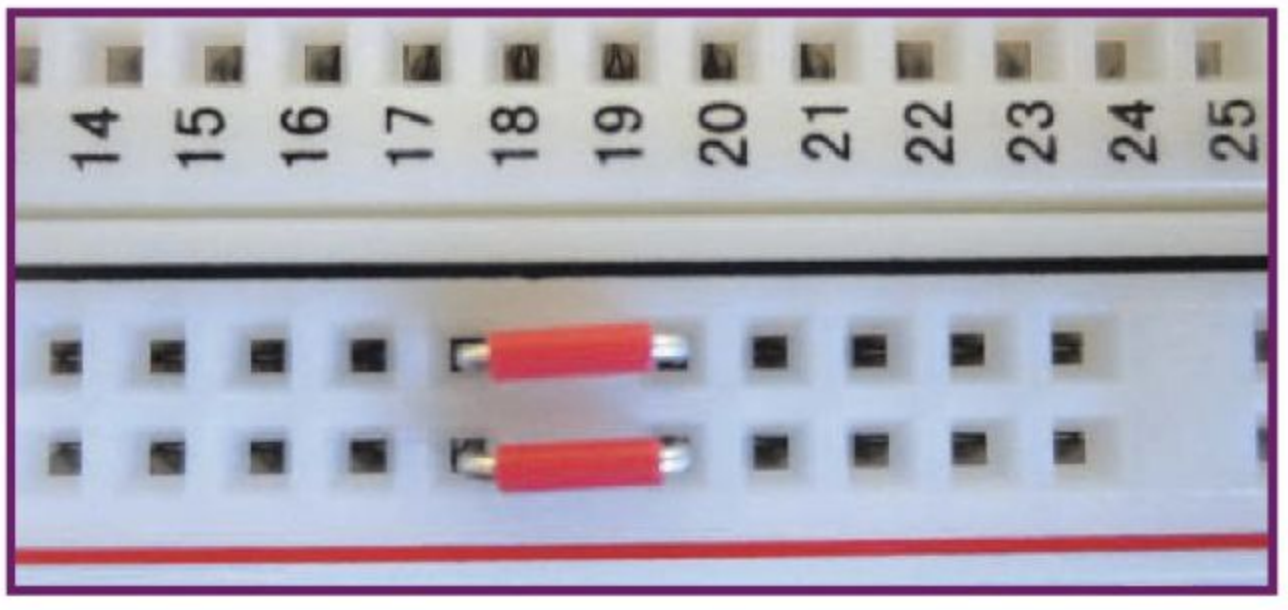
Conexiones básicas



1 Para esta actividad básica, necesitaremos un protoboard y algunos cables de puente, tal como se observa en la imagen.



2 En primer lugar, realizamos una conexión que servirá para que ambos pares de buses sean capaces de conducir corriente cuando les agreguemos una fuente de poder. Así, será más sencillo manipular los circuitos integrados. Conectamos ambos buses teniendo en cuenta sus sectores positivos y negativos.



3 En algunos protoboards encontraremos que la parte media de los buses está separada; en estos casos será necesario establecer un puente que permita darle continuidad a la corriente suministrada. Si es necesario, debemos 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 un toma de corriente y el 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 separadas por un material aislante. Estas láminas se cargan eléctricamente al conectar una fuente de tensión o batería.
Ambas placas se llenan con la misma cantidad de carga, pero con diferentes signos; es decir, una tiene carga positiva mientras que la otra placa tiene carga negativa. Una vez cargadas, se presenta una tensión entre ellas y pueden soltar la carga almacenada cuando se conecta 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 tensión del condensador.

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

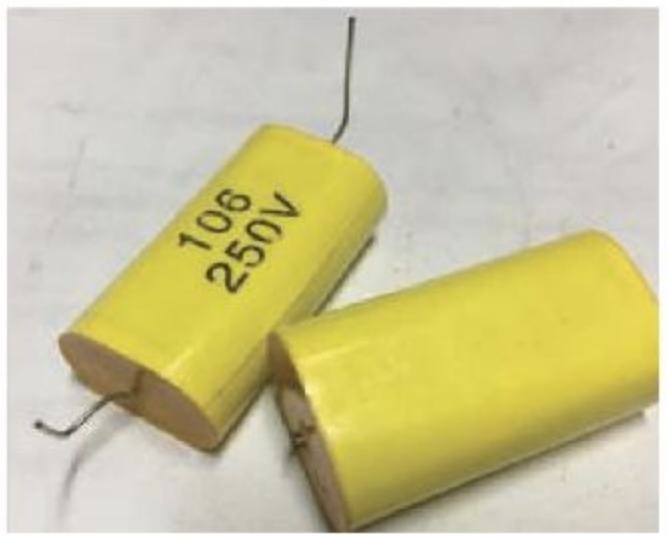
Un condensador eléctrico es un dispositivo pasivo, utilizado en electricidad y electrónica, capaz de almacenar energía sustentando un campo eléctrico.

Tanto la carga como la descarga de un condensador no se realizan en forma instantánea, razón por la cual 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:

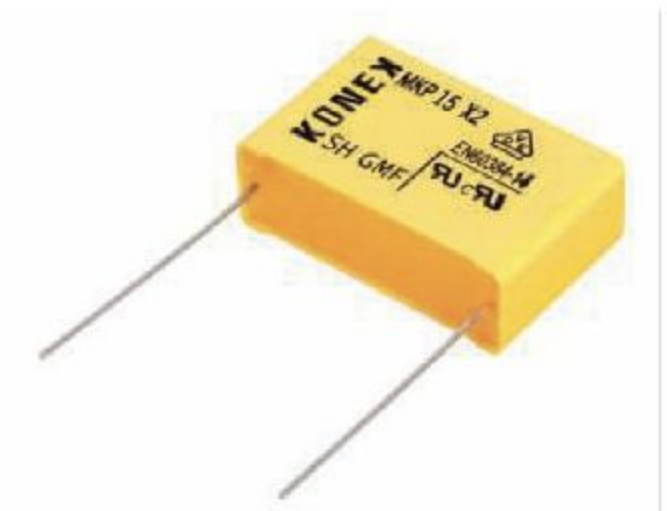
- **Variable**: este tipo de condensador puede cambiar su capacidad en forma mecánica o electrónica. Existen los **trimmers**, que permiten elegir entre varios valores de capacidad; y también los **de sincronización**, que presentan una capacidad entre límites establecidos.

- **Electrolítico**: se trata de un condensador que posee una de sus placas formada por líquido iónico. Se destaca por sobre otros por su mayor capacidad.

- **Cerámico**: este tipo de condensador posee un elemento cerámico revestido en láminas metálicas, como material dieléctrico. Gracias a que la constante cerámica es bastante alta, estos condensadores tienen 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**: utilizan una delgada placa plástica como material dieléctrico; los hay de diferentes clases, dependiendo del tipo de plástico utilizado, por ejemplo, polipropileno, policarbonato, poliestireno, poliéster, teflón y 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 conecta 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.

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, 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 longitud 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 emite luz. Los colores emitidos dependerán del material con que se fabrique el semiconductor.

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

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

- **Contadores**: se utilizan para desplegar contadores.
- **Corriente continua**: podemos usarlos 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.

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.

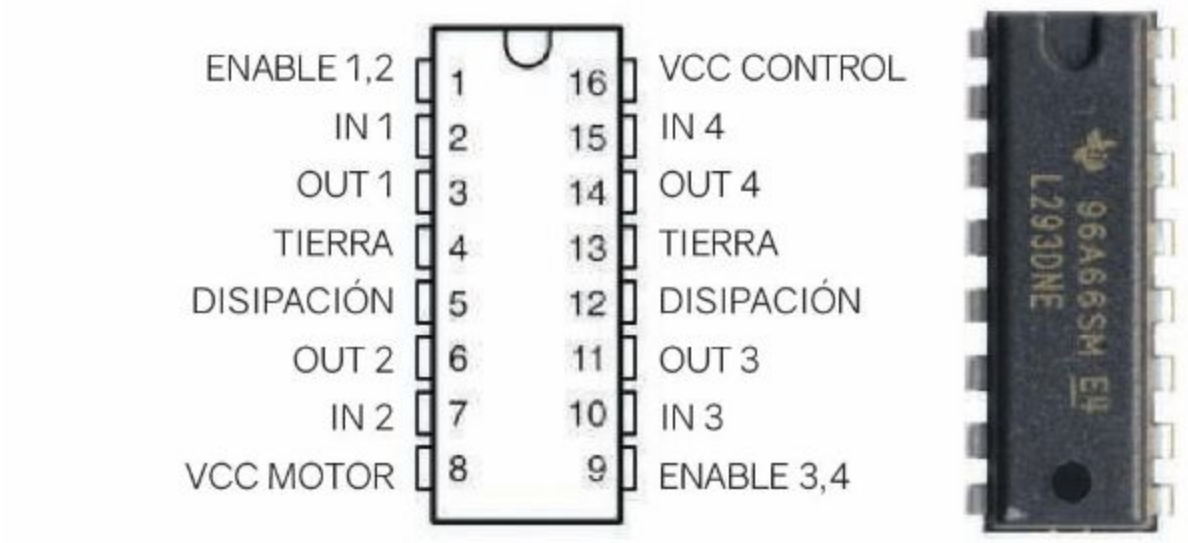
Si se aplica una tensión adecuada a los terminales de un diodo emisor de luz (LED), los electrones se recombinan con los huecos en la región de la unión p-n del dispositivo, liberando energía en forma de fotones. El color de la luz generada se determina por la anchura de la banda prohibida del semiconductor.

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: podemos construirlo utilizando algunos pocos componentes individuales o 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 una disposición específica de componentes, como transistores y diodos, que nos permiten controlar la polaridad de los terminales de salida.

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.



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 el 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 4,5 V hasta 36 V.

Broche de presión de pila

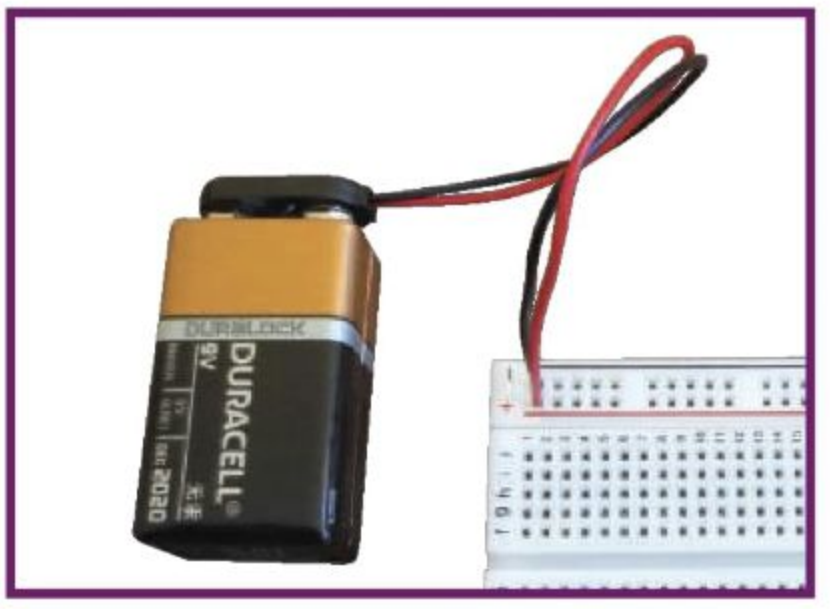
Aunque se trata de un componente muy sencillo, lo necesitamos para conectar una pila de 9 V a clavijas de corriente, que a su vez conectaremos al protoboard o a la placa Arduino.

También podemos encontrarlo con el nombre de **conector de batería 9 V**. Se presenta como una estructura que posee dos broches adecuados para las baterías de 9 V, 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:

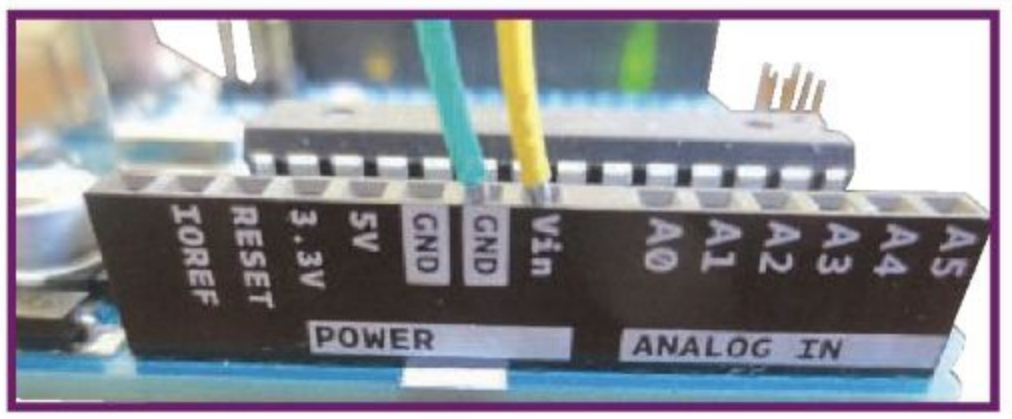
- **Mediante el puerto USB**: se trata de la manera más habitual; mantenemos el cable USB conectado a la placa y a la computadora, de modo que la mantendremos con corriente. Lo malo es que necesitamos tener una computadora cerca en todo momento.
- **Utilizar una batería 9 V y el conector de corriente**: en este caso, hacemos uso de una batería 9 V; para esto, 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 9 V y los pines de Arduino**: mediante esta alternativa, también utilizaremos una batería 9 V y un broche de presión de pila, pero, en vez 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.

Alimentar Arduino mediante el protoboard

1 En primer lugar, conectamos la batería de 9 V al broche de presión de pila. Debemos efectuar la conexión a la pila hasta sentir el clic que indica una posición correcta.



2 Tomamos los filamentos rojo y negro que corresponden al positivo y al negativo del broche de presión y los conectamos en los lugares adecuados del protoboard.



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

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

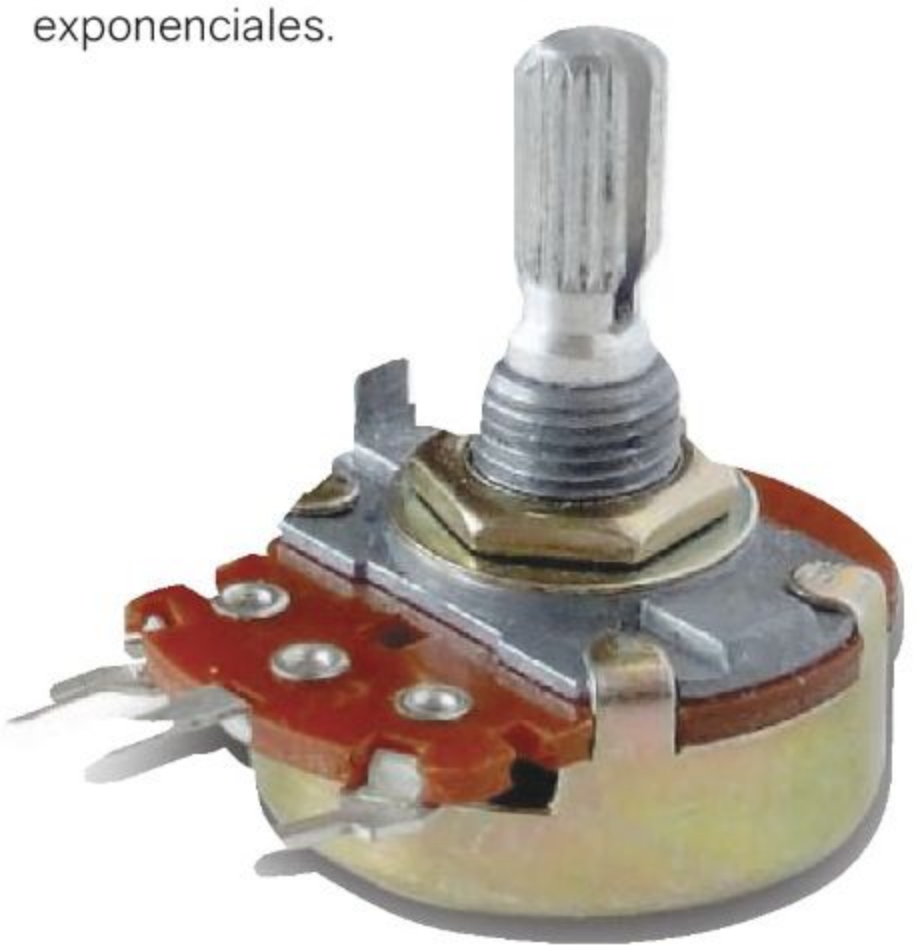
Potenciómetro

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 (5 k, 10 k o 20 k ohmios).
Si consideramos su forma, podemos encontrar dos tipos de potenciómetros:

- **Lineales**: tienen 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 entre 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.

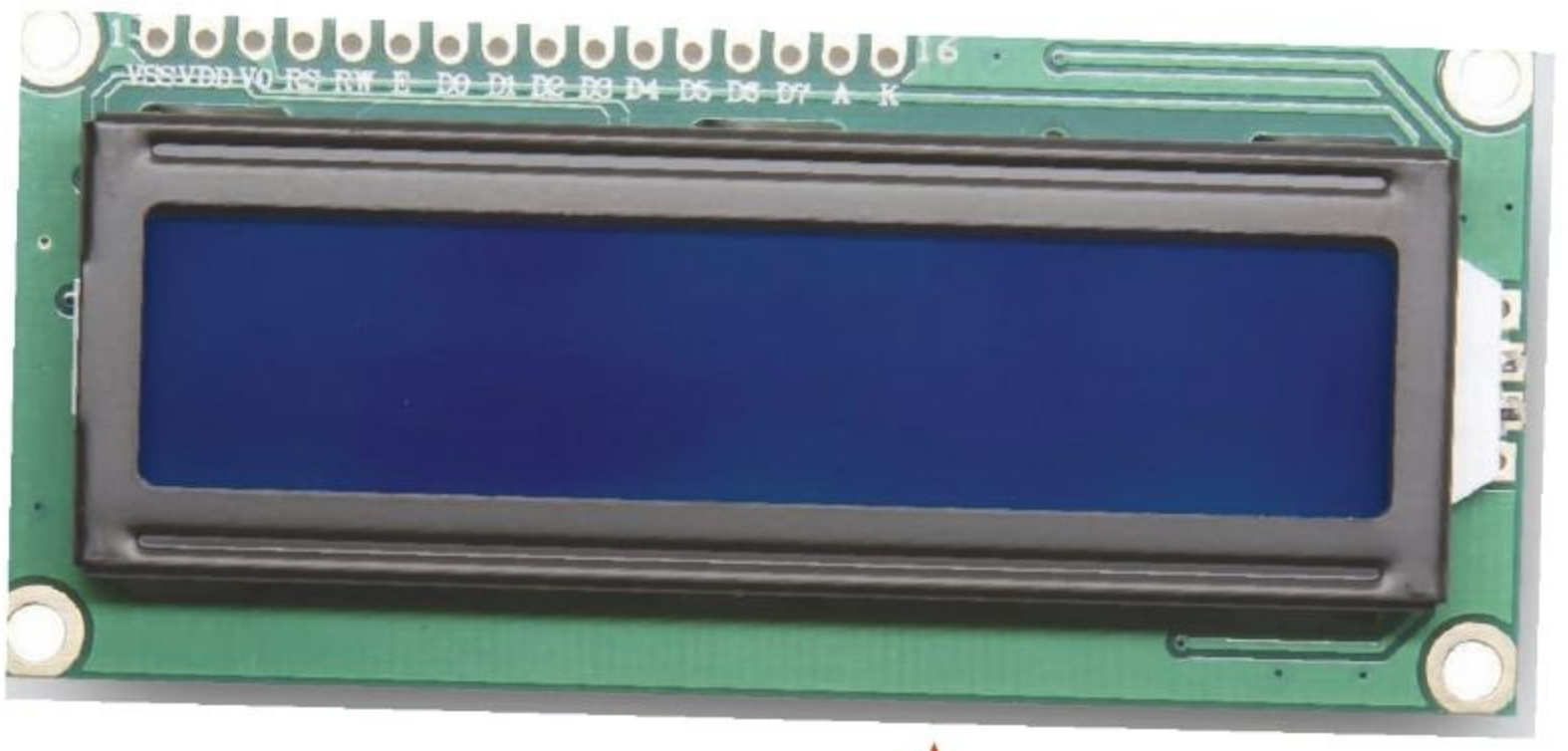
Conectar un potenciómetro a una placa Arduino UNO requiere el uso del pin analógico 0, además de los pines 5 V y GND. Luego necesitaremos un sencillo código que será el encargado de leer e interpretar el valor de tensión:

```
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);
}
```

Pantalla de cristal líquido

Aunque existen pantallas de cristal líquido de 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.



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.

Para enviar datos a la pantalla LCD, utilizaremos la librería LiquidCrystal.
Por ejemplo, en una pantalla de dos filas y 16 caracteres en cada una, hallaremos un listado de pines como el siguiente:

- Pin 1: VSS o GND
- Pin 2: VDD o alimentación +5 V.
- 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 los cuales se envía o recibe la información.
- Pin 15: es el ánodo del LED de iluminación para la luz de fondo (+5 V).
- Pin 16: es el cátodo del LED de iluminación para la luz de fondo (GND).

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 5 V 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.

Motor de corriente continua

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

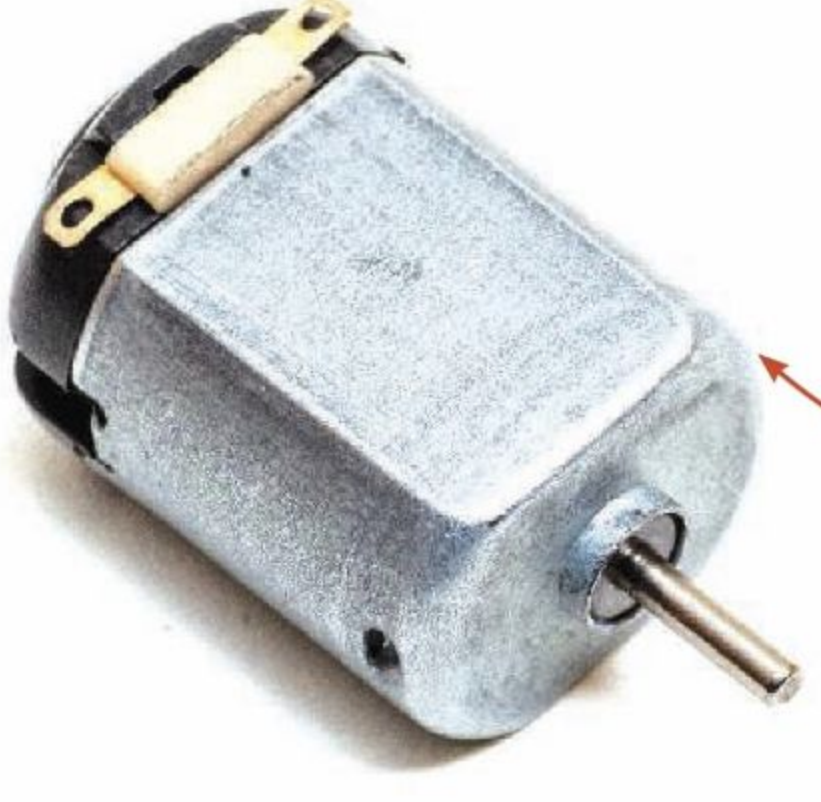
La verdad es que estamos rodeados de motores eléctricos, de todos los tipos y tamaños; por ejemplo, el motor que mueve el automóvil, o los que hacen girar el plato del microondas o proporcionan el movimiento para reproducir un disco compacto.

En nuestros proyectos con Arduino, generalmente trabajaremos con pequeños motores de corriente continua.

En primer lugar, debemos recordar que los imanes poseen un polo positivo y uno negativo y, mientras que los polos opuestos se atraen, los iguales son repelidos. 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.

El motor de corriente continua, motor CC o DC, es una máquina que convierte energía eléctrica en mecánica, provocando un movimiento rotatorio.



En esta imagen podemos apreciar un pequeño motor de corriente continua de 3 V, 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 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 modo que la fuerza que se ejerce sobre el rotor se multiplica proporcionalmente.

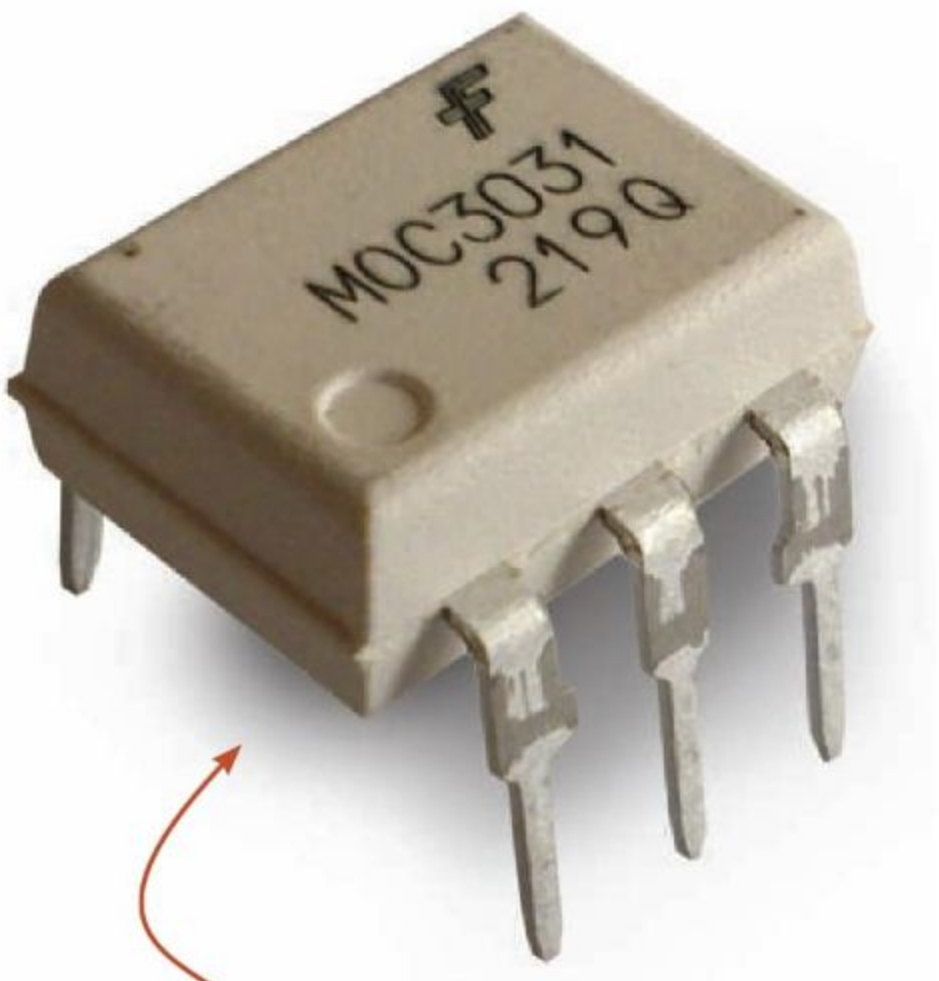
Optoacoplador

También denominado 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.

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, y cuando precisamos interfaces en circuitos lógicos o entre señales de corriente alterna y circuitos lógicos, entre otras tareas.

En su construcción, se trata de un circuito integrado sencillo, generalmente compuesto por un LED y un fototransistor.

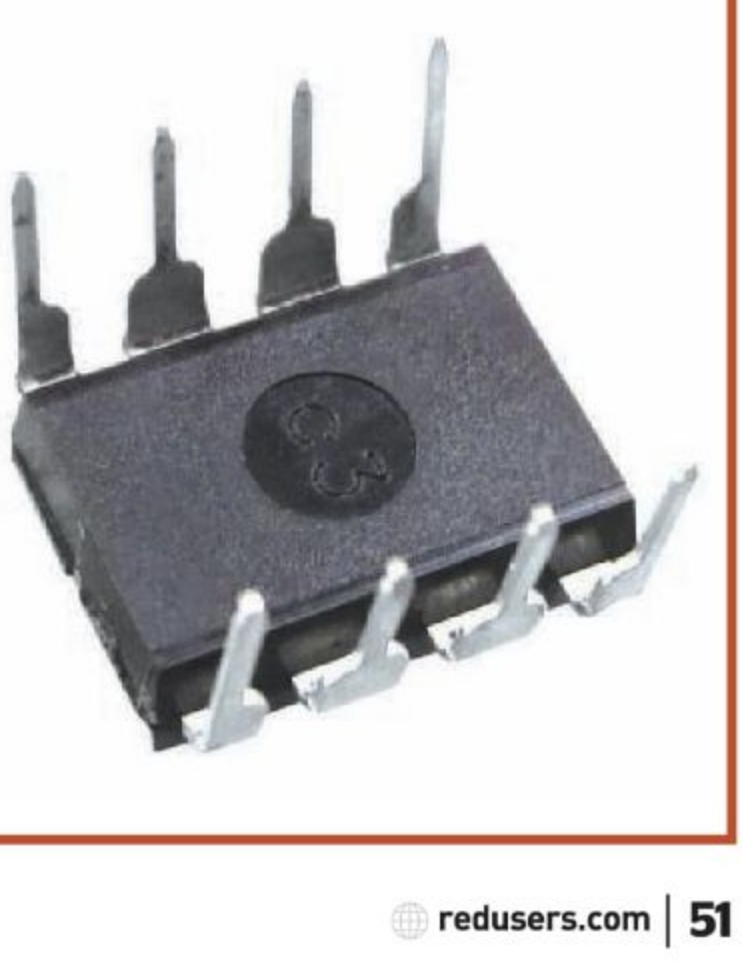
Cuando hacemos circular una señal eléctrica a través del LED, se enciende; la luz emitida es detectada por el fototransistor que trabajará en modo saturación. De esta forma, puede actuar como interfaz entre dos circuitos, que estarán unidos ópticamente y así se protegerán contra los picos de tensión.



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.

Optoacoplador 4N35

El 4N35 es uno de los optoacopladores más utilizados, por lo que seguramente nos toparemos con él en muchos proyectos. Las principales características de su LED son las siguientes: una caída de voltaje típica de 1,15 V 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 45 V para una corriente de colector de 1 mA, caída de voltaje del emisor a la base de 7,8 V para una corriente de emisor de 100 µA, caída de voltaje del colector a la base de 100 V 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 5 V de 400.



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. 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, vemos que en su parte inferior se presentan cuatro patillas de conexión, porque ambos pares están internamente conectados de modo que, en realidad, funcionan como dos patillas de conexión.

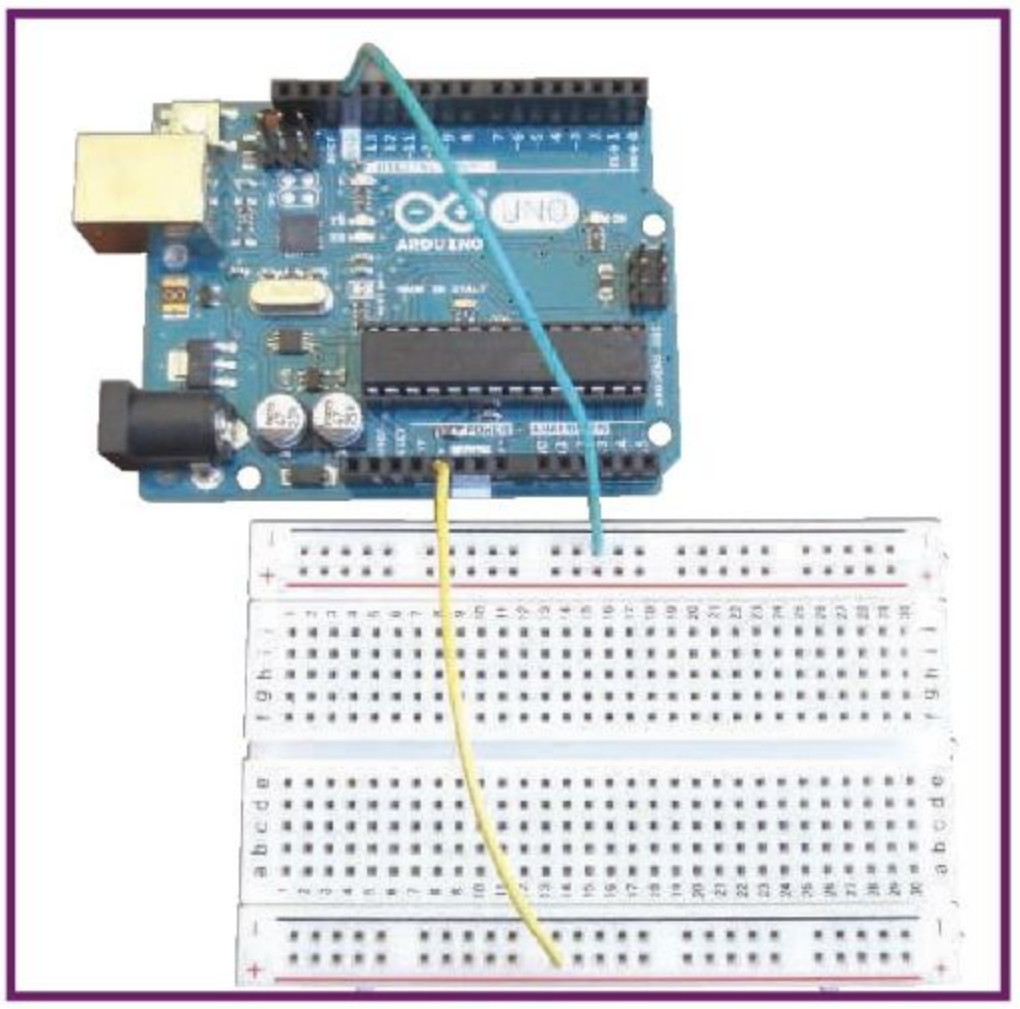
Aunque puede parecer que el pulsador posee 4 terminales, la verdad es que internamente ambos pares se encuentran unidos, y se comportan como dos terminales.

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

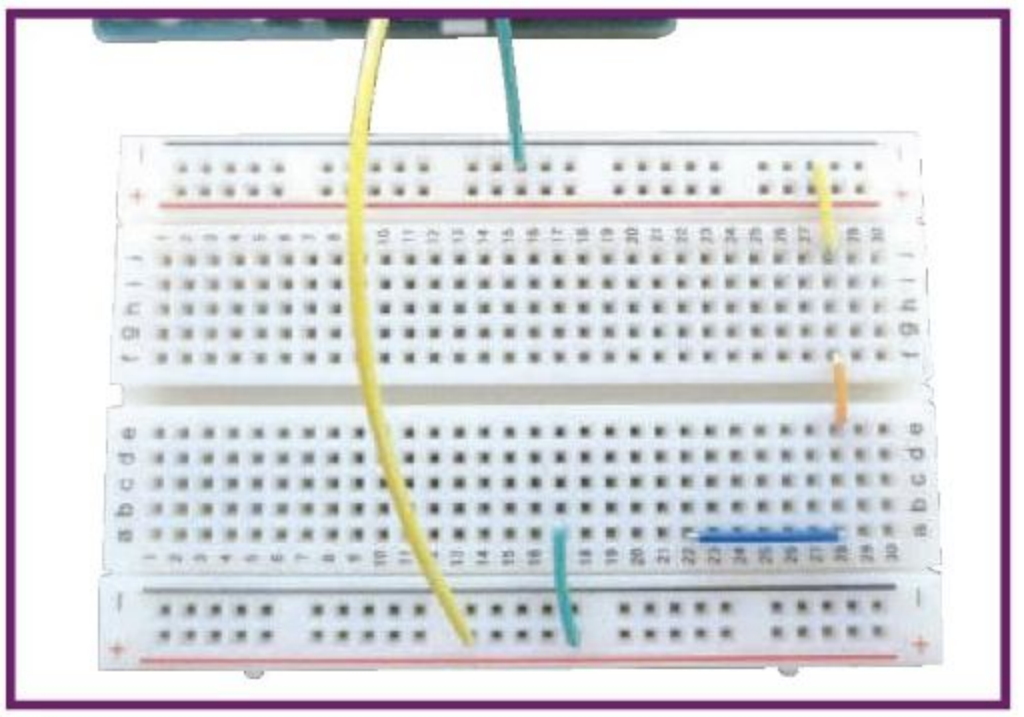
Sin embargo, debemos considerar que el pulsador presenta un funcionamiento comparable con el de un timbre, es decir que cerrará el circuito y permitirá el funcionamiento de otro elemento solo mientras lo mantengamos pulsado. Por esta razón, no sería práctico utilizarlo para hacer funcionar un motor, pues tendríamos que mantenerlo siempre pulsado.
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, pero por ahora solo usaremos esta conexión para obtener la energía necesaria.

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.

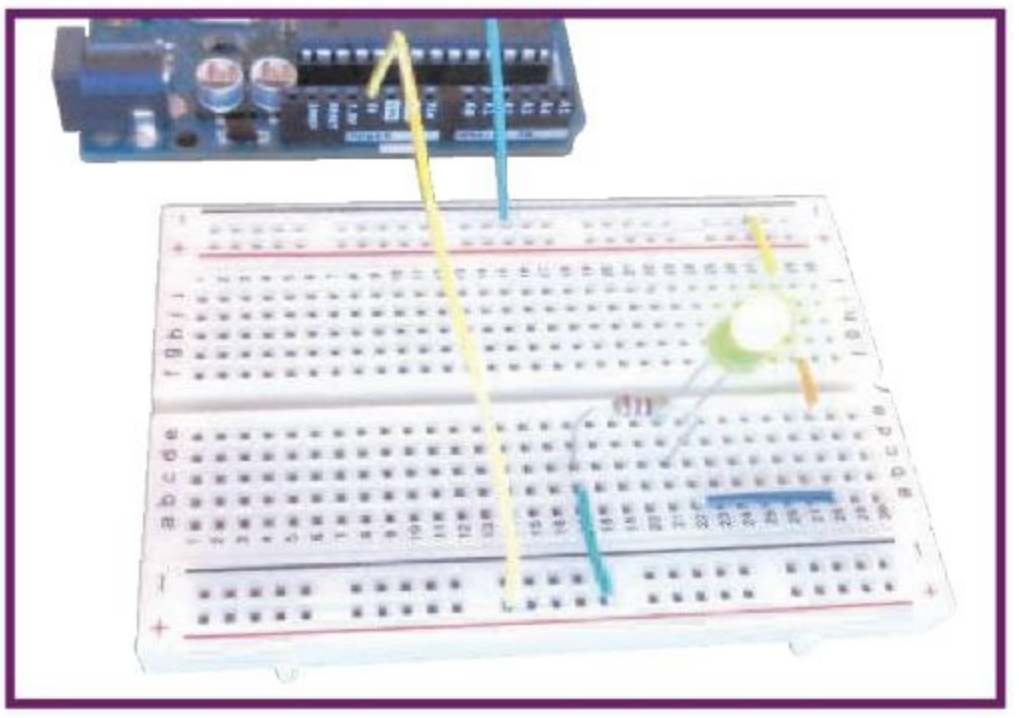
Control sencillo de un LED



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

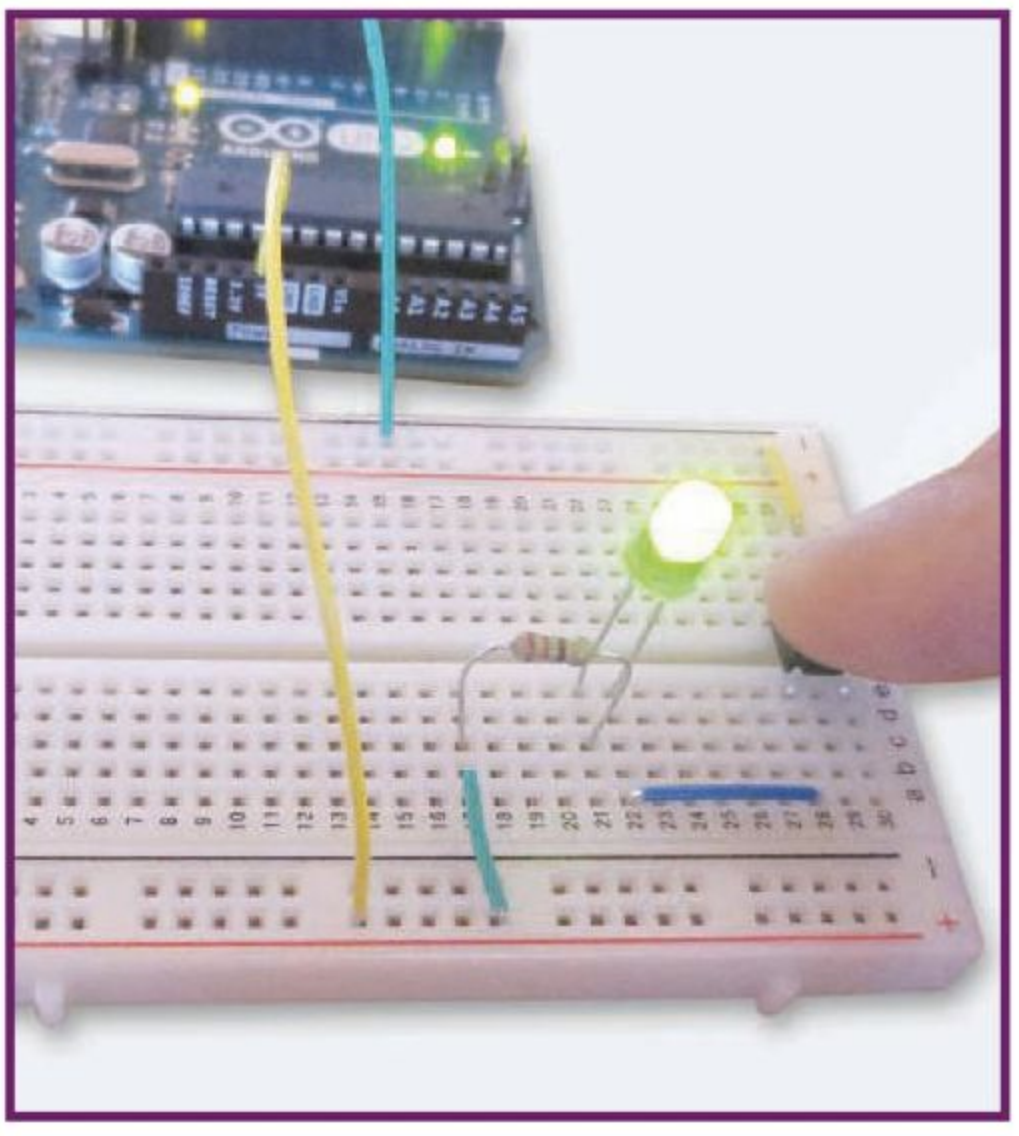


2 En primer lugar, preparamos las conexiones para que se encienda un LED sin necesidad de utilizar un pulsador. Colocamos las conexiones de puente tal como indica la imagen.



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

4 Al terminar de montar este circuito, lo conectamos a la PC mediante USB; veremos que el LED se enciende de forma continua. Las salidas 5 V y GND permiten hacer un circuito igual que como lo haríamos si utilizáramos los cables conectados a una pila.



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

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

Resistencias

Una resistencia es un componente capaz de oponerse al paso de la corriente, por lo que 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 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.

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 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.

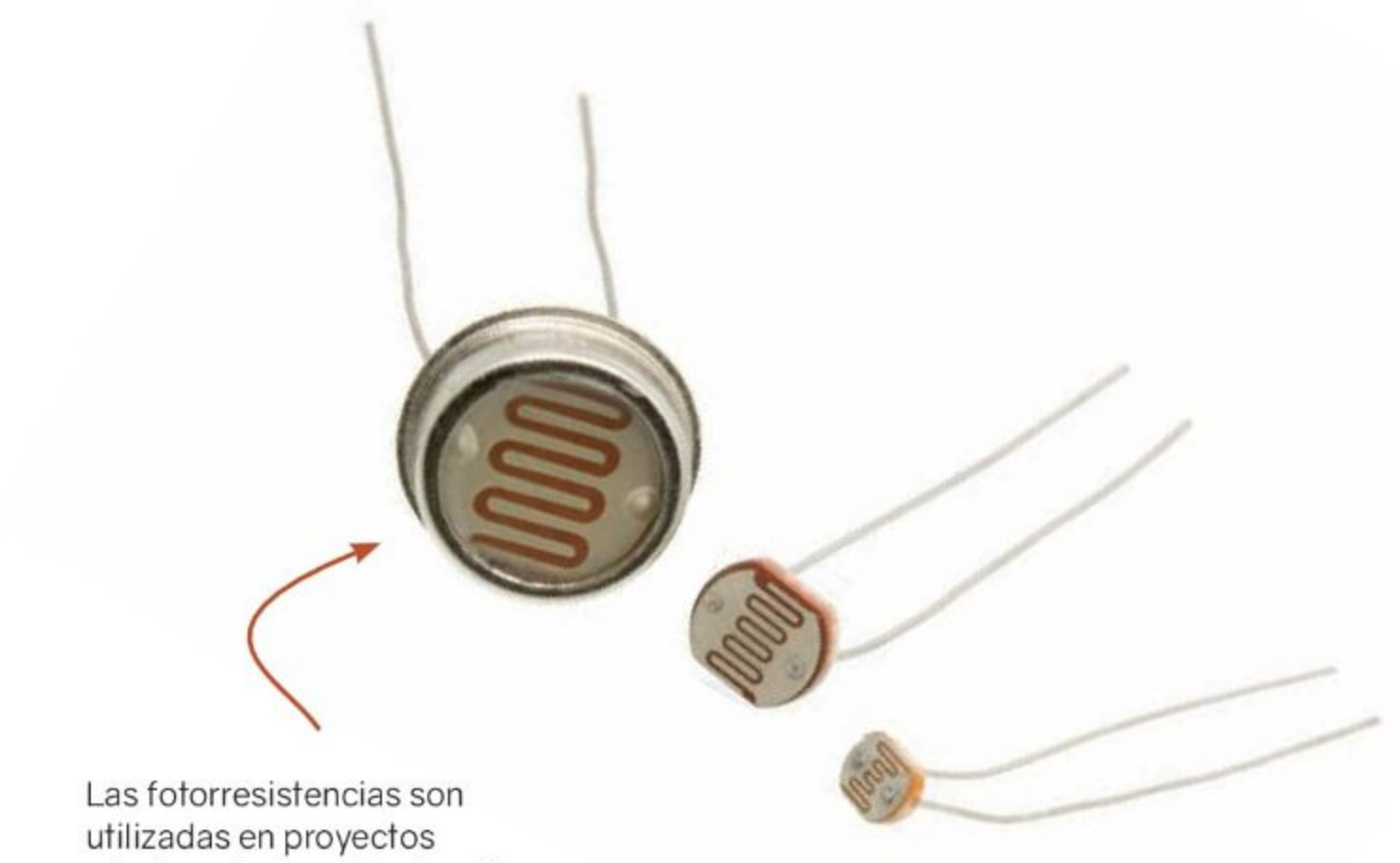
CÓDIGOS DE COLOR QUE NOS PERMITEN IDENTIFICAR EL VALOR DE UNA RESISTENCIA O RESISTOR.

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%

Fotorresistencia

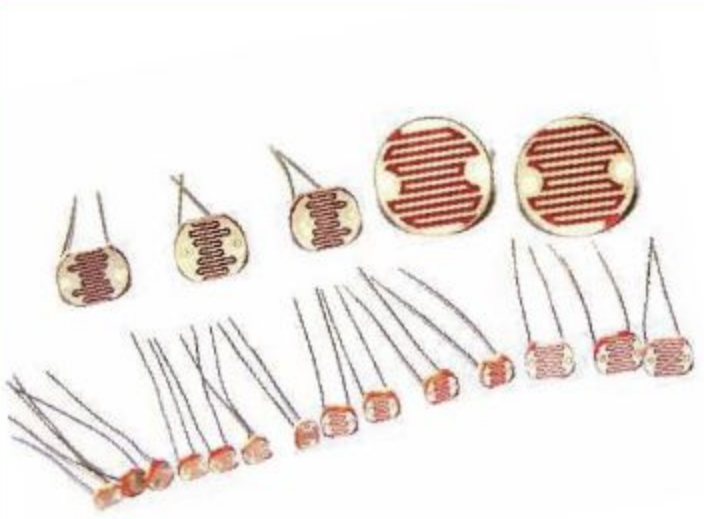
Una fotorresistencia, resistencia dependiente de la luz o fotocélula es una resistencia del tipo variable, capaz de cambiar el valor de su resistencia dependiendo del nivel de luz que toque su superficie.

En un elemento de este tipo, 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 es el siguiente: 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.



Radiación luminosa

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

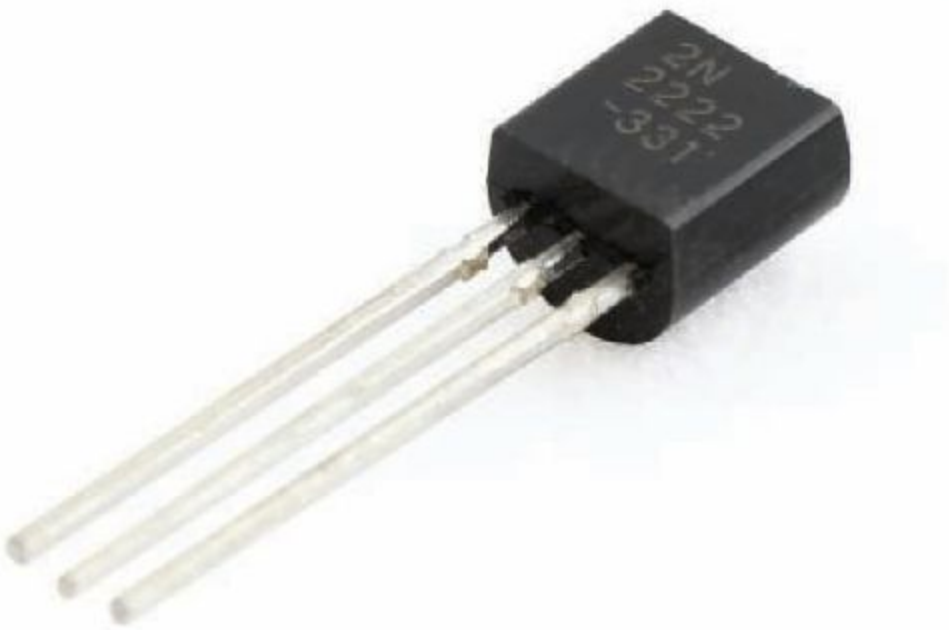
Transistor

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 lo que 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.

Un transistor puede presentar tres estados posibles:

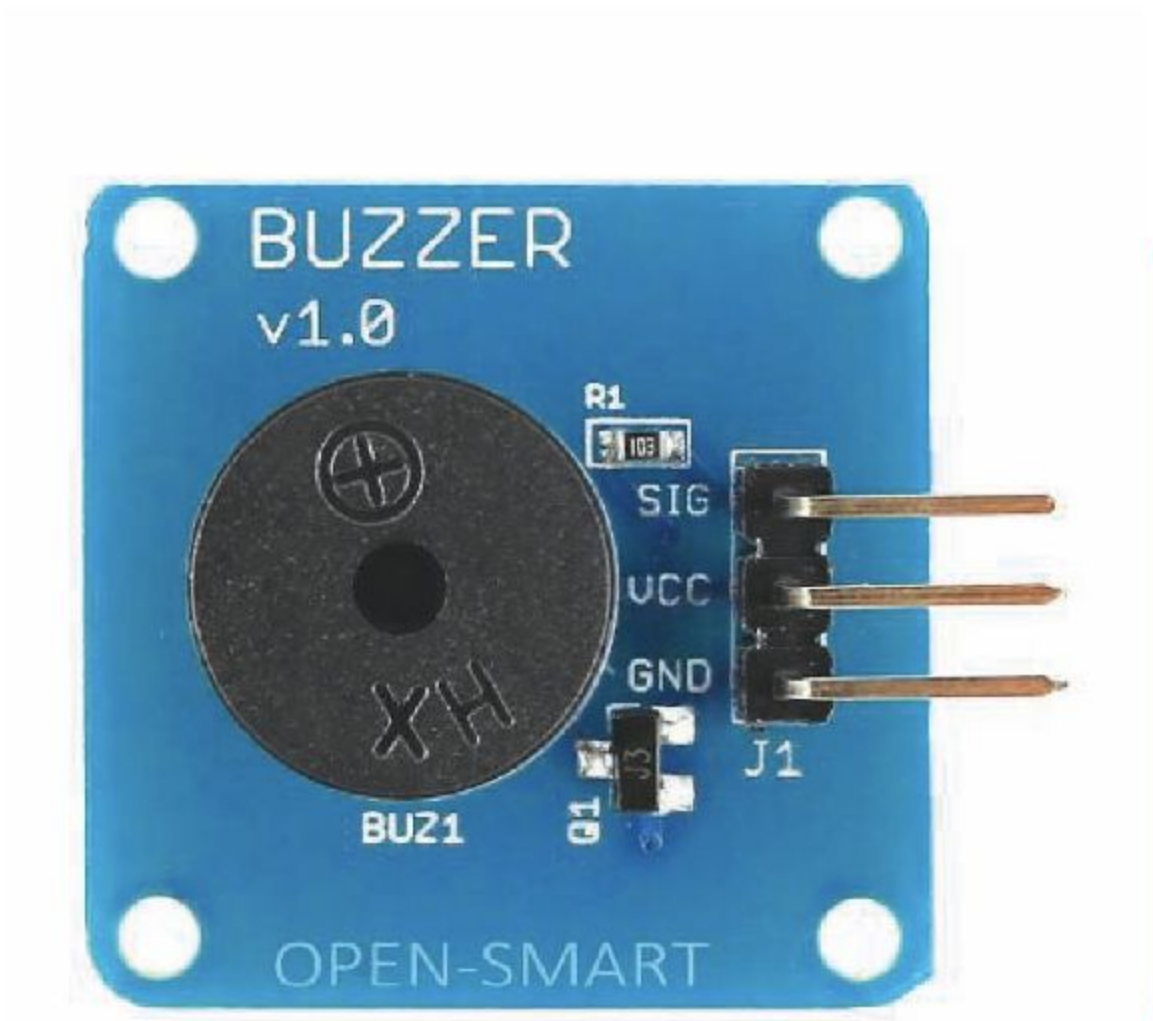
- **Activo**: puede dejar pasar más o menos corriente. Actúa como un amplificador.
- **Corte**: impide el paso de la corriente. No circula intensidad por la base, por lo que la intensidad de colector y emisor es nula. El transistor, entre colector y emisor, se comporta como si se tratara de un interruptor 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.

La **piezoelectricidad** es una propiedad que 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, vibran.
Un ejemplo de su aplicación se encuentra 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.
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.

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



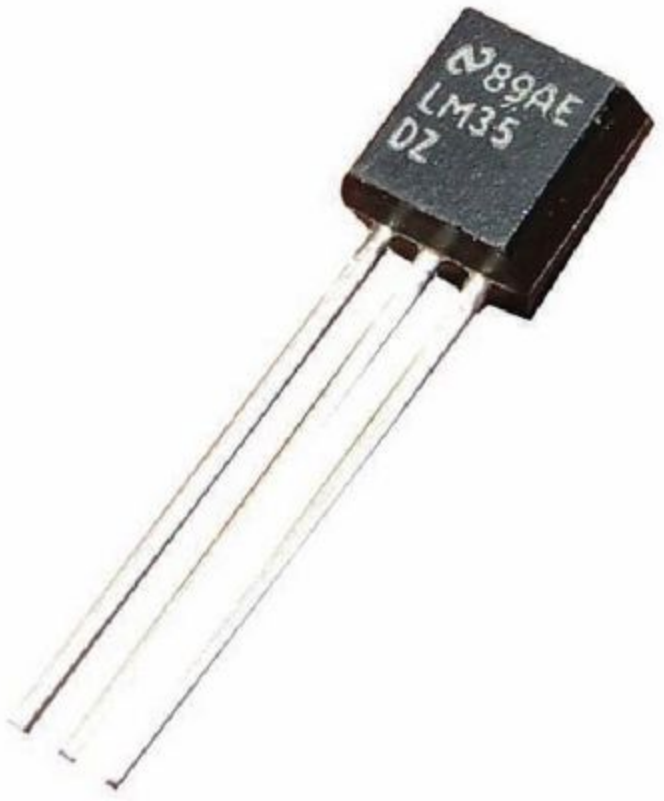
Realizar una conexión básica de un zumbador piezoeléctrico a una placa Arduino es bastante sencillo, solo necesitamos conectar el negativo a GND y el positivo al pin 9 de la placa. En este punto debemos ser cuidadosos, pues los buzzers poseen polaridad, de modo que no sonarán si los conectamos en forma incorrecta.

Sensores

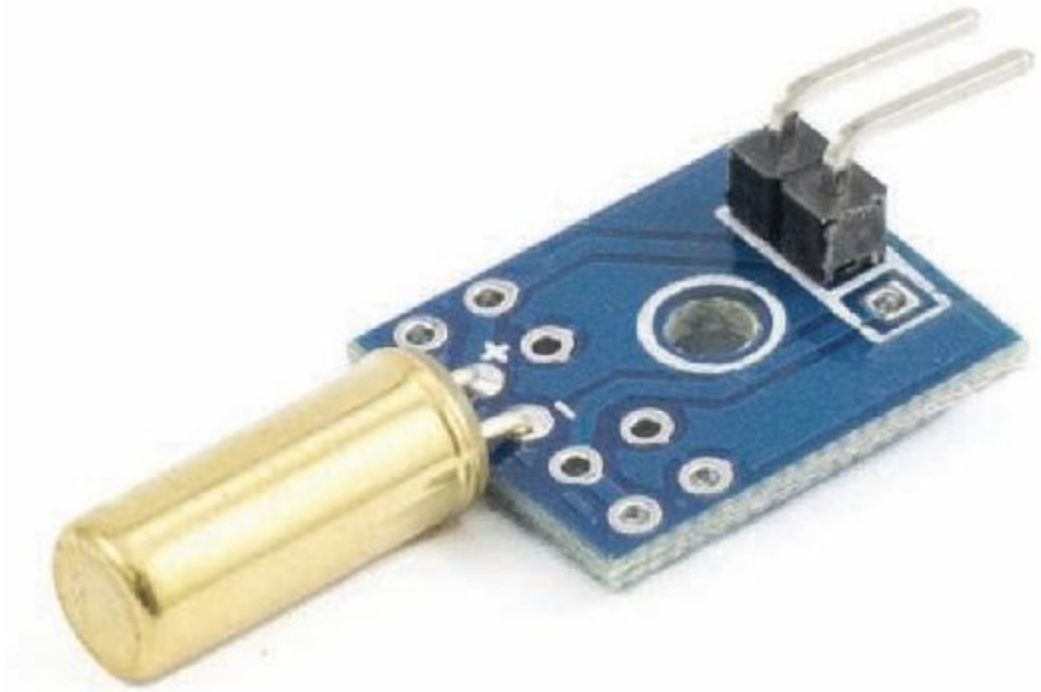
Existe una gran oferta de sensores listos para conectar a nuestra placa Arduino; en este caso detallaremos el sensor de temperatura y el de inclinación.

Sensor de temperatura

Es un sensor que posee la capacidad de cambiar la 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 -55º C hasta los 150º C; 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

Se trata de un dispositivo que funciona como un interruptor, abriéndose o cerrándose, dependiendo de su orientación.
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 lo hará el paso de la corriente.

Otro de los componentes importantes para trabajar en proyectos con Arduino es el sensor de inclinación.

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 desde una placa Arduino.

Los **servomotores**, más conocidos como **servos**, son utilizados en tareas específicas, por ejemplo, relacionadas con la robótica y la automatización.
Los motores DC tienen 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.

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

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.

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

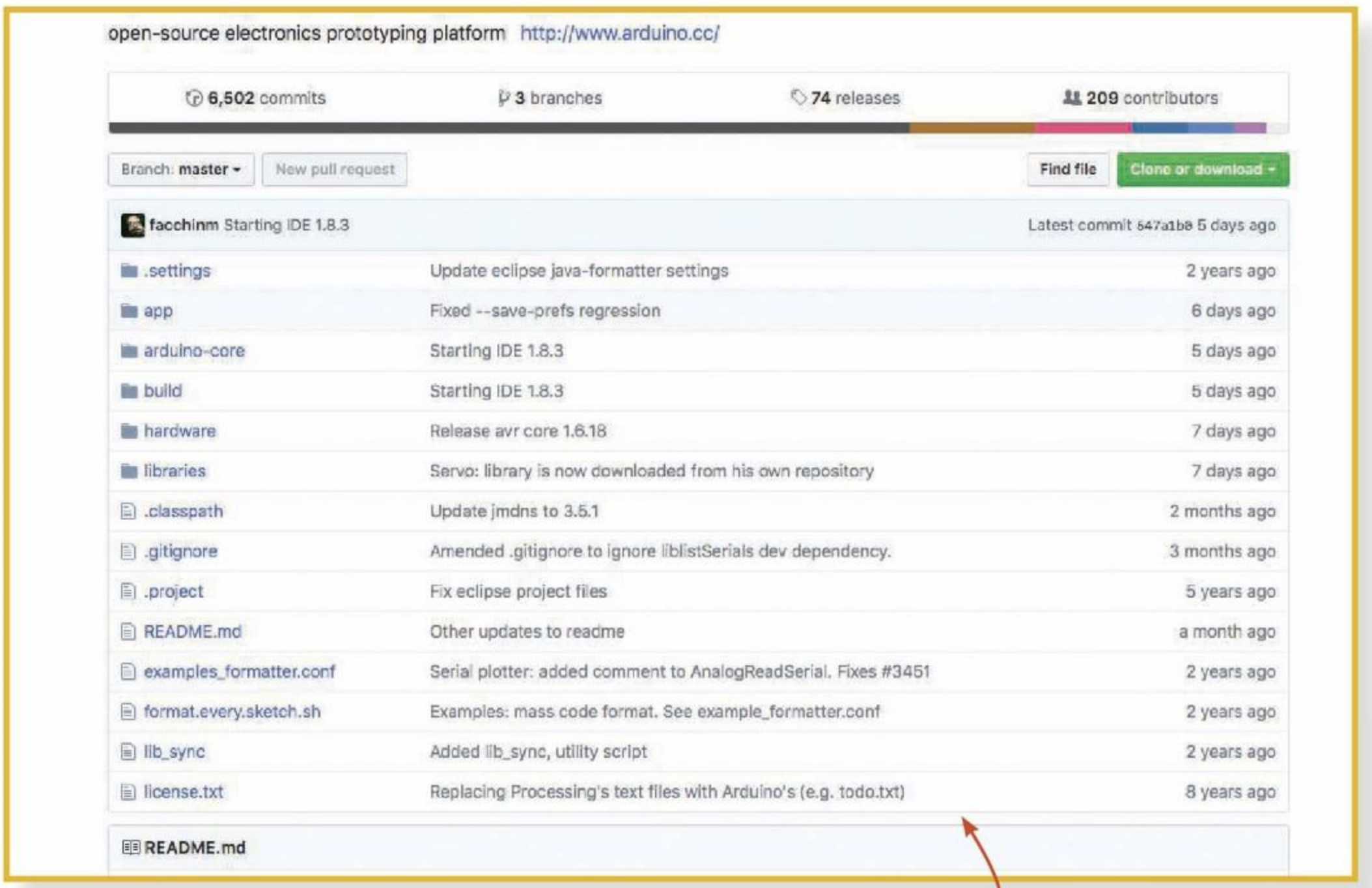
- **Giro limitado**: son los más comunes. Permiten una rotación de 180º, por lo que no pueden realizar una vuelta completa.
- **Rotación continua**: son capaces de girar 360º, es decir, pueden dar una vuelta completa. Son similares a un motor convencional, pero poseen las características adicionales de servomotor y por eso es posible controlar su posición y velocidad.

¿Qué es Arduino IDE?

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 poder 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:

- **Gratuito 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, de modo 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 or download**.

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

- **Multiplataforma**: una de las ventajas de este IDE es que se trata de una aplicación multiplataforma, es decir, que puede ser instalada y utilizada 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 esto parezca una debilidad por las reducidas opciones que vemos al acceder por primera vez, a medida que la conozcamos más a fondo, veremos que nos proporciona todo lo necesario 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 está justo donde la necesitamos, sin que la ventana principal ocupe toda la pantalla.

- **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 revisaremos en detalle el proceso de configuración del IDE.

- **Nuevas opciones**: la última versión de este IDE nos propone algunas características novedosas, como 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 sketches vía red para algunas placas.

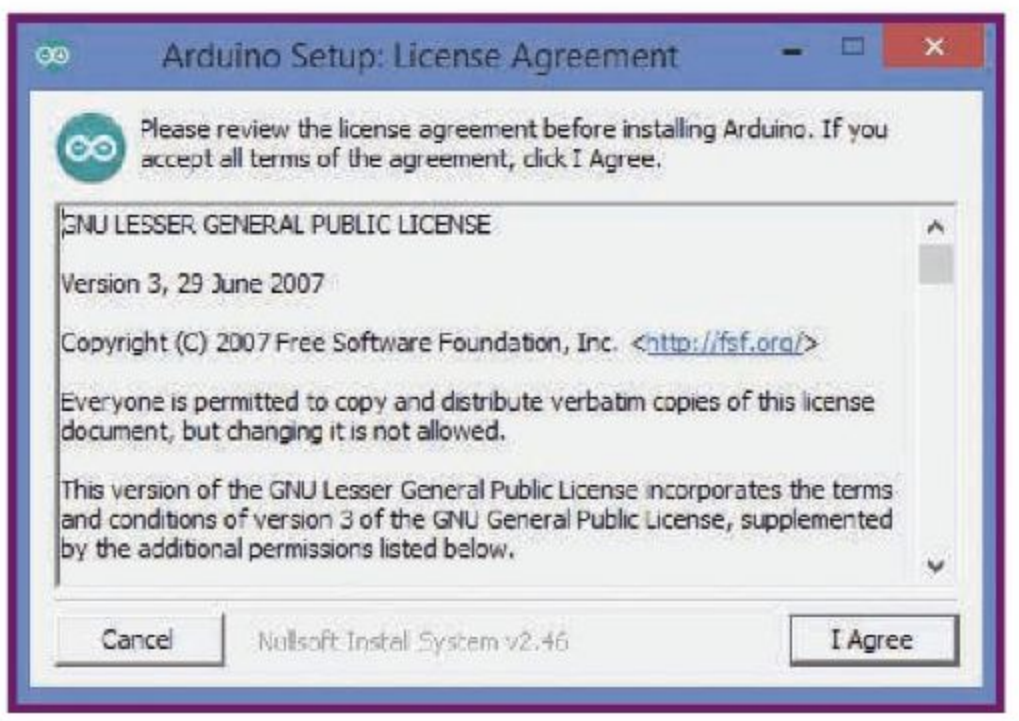
La interfaz del Arduino IDE es bastante sencilla; como vemos en esta imagen, la sección dedicada al código ocupa gran parte de la ventana.

Instalación del IDE

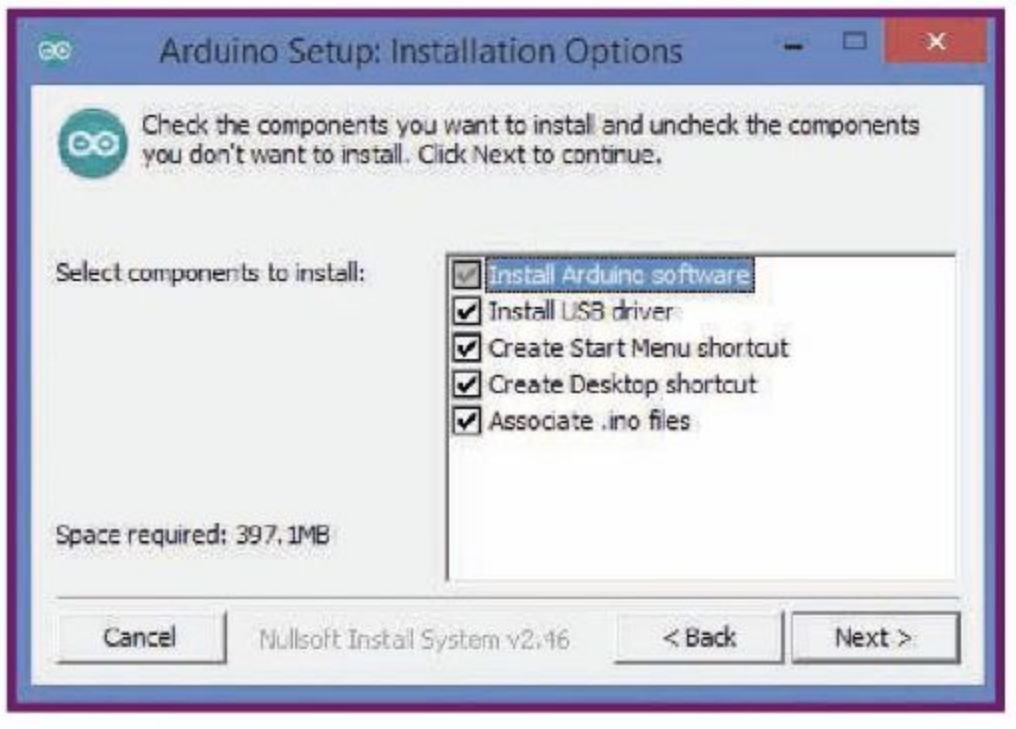
Antes de conocer y trabajar con Arduino IDE, debemos realizar las tareas que nos permitirán instalarlo. Como sabemos, se trata de una herramienta multiplataforma, por lo que podremos trabajar en diversos sistemas operativos. En primer lugar, revisaremos la forma en que debemos instalar el IDE en un sistema Windows.

Instalar Arduino IDE en Windows

1 Ingresamos en la dirección **www.arduino.cc** y hacemos clic en el enlace **Software**, que se encuentra en la barra superior de opciones. En la ventana que se presenta, bajamos hasta la sección Arduino IDE y presionamos sobre **Windows Installer**.

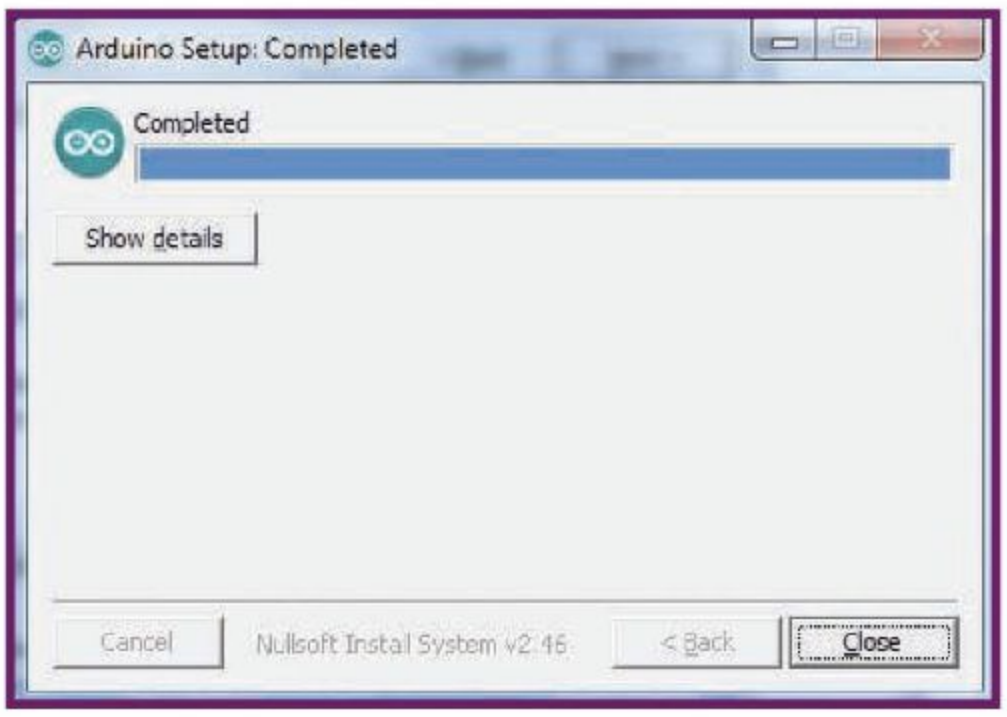


2 Una vez que la descarga haya finalizado, hacemos doble clic sobre el archivo adecuado y esperamos mientras se inicia el asistente de instalación. En la primera pantalla será necesario aceptar el acuerdo de licencia, para lo cual presionamos **I Agree**.



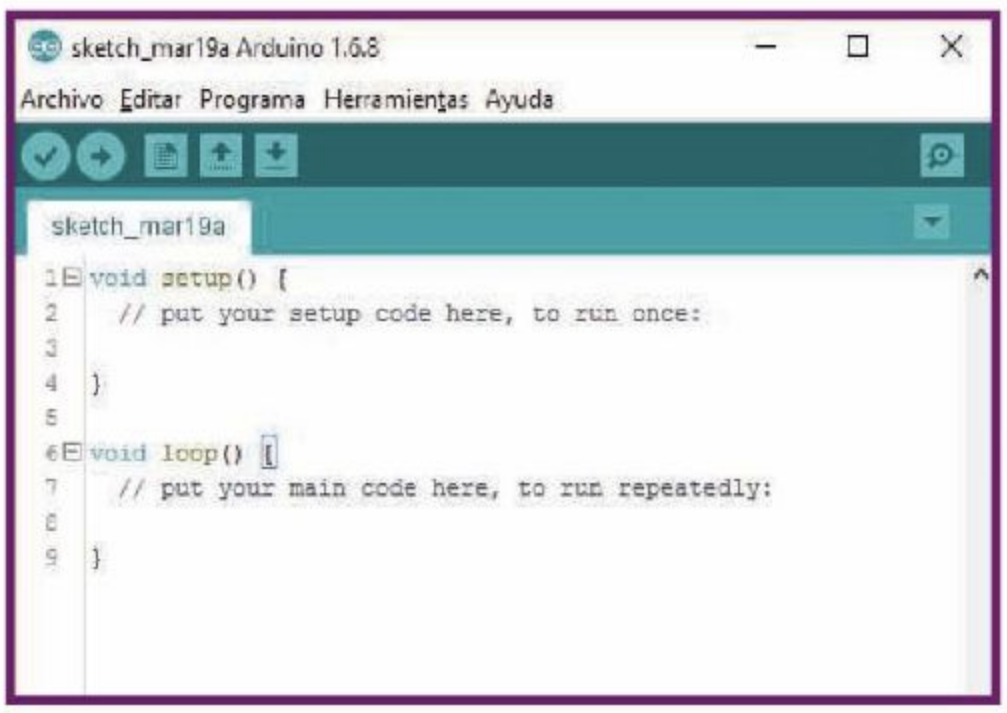
3 Seleccionamos los componentes que serán instalados; es recomendable marcar, al menos, las opciones **InstallArduino Software**, **Install USB driver** y **Associate .ino files**; luego presionamos el botón **Next**.

4 Es posible que aparezca una ventana que indica que se instalarán algunos drivers, aceptamos que se instalen. Luego el proceso continuará.



5 El proceso de instalación solo tardará un momento y podremos ver su avance gracias a la barra de la ventana. Cuando se complete, hacemos clic sobre el botón **Close**.

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



7 Una vez cargado, la apariencia del IDE será similar a la que se observa 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**, hacer clic sobre **Software** y elegir 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.

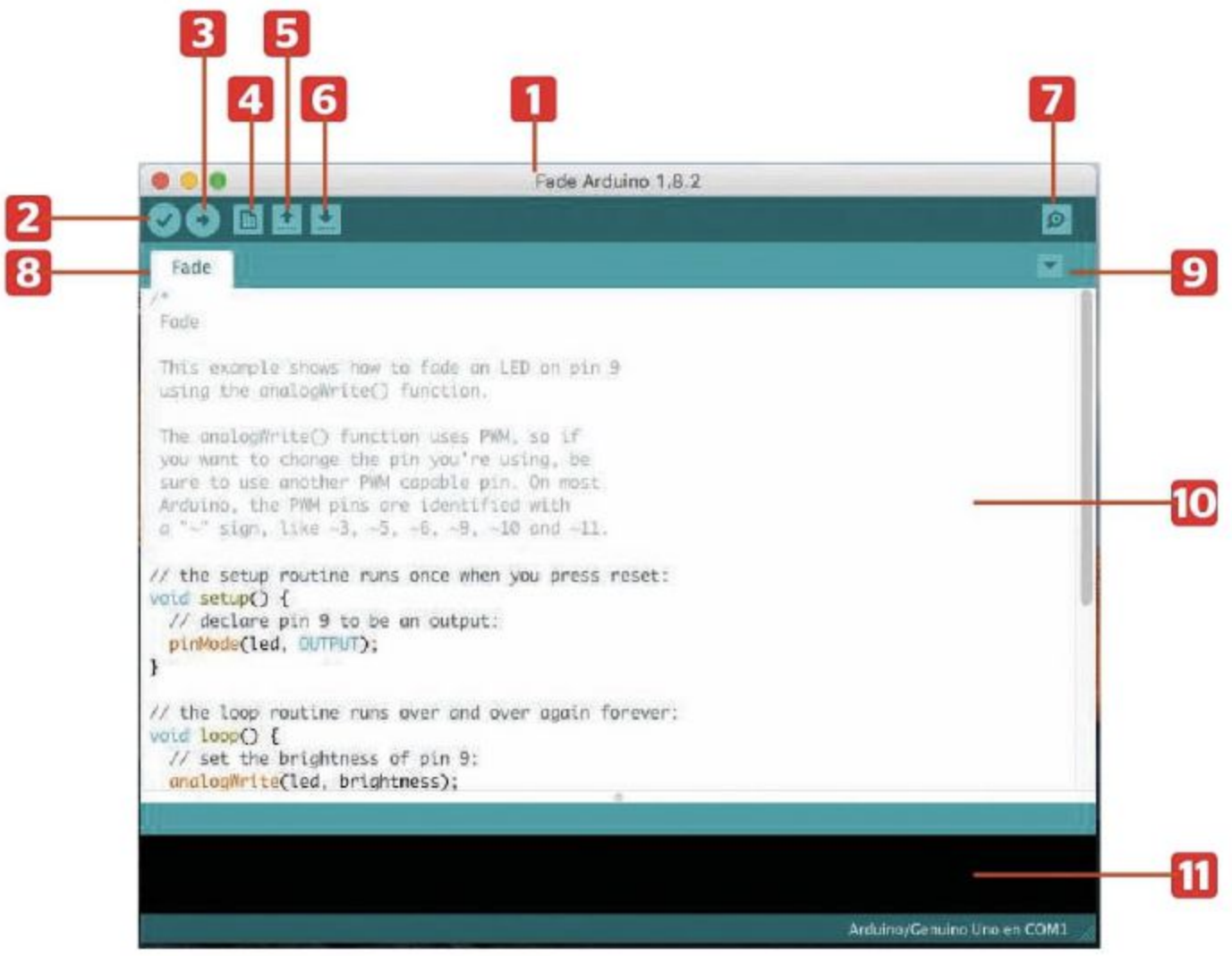
Más opciones

Otra de las opciones disponibles para desarrollar código para Arduino es utilizar un IDE en línea, de modo que 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 se ejecuta 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.

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.



1 **Información**: en esta barra vemos el nombre del proyecto en el que estamos trabajando, así como también la versión del IDE.

2 **Botón Verificar**: si hacemos clic en este botón, el IDE verifica el programa o sketch que hemos escrito. Si se encuentran problemas, aparecen en la consola de error; de lo contrario, la misma consola nos entregará información relevante sobre el programa.

3 **Botón Subir**: mediante este botón, podemos subir el programa ya verificado a la placa Arduino.

4 **Botón Nuevo**: crea un nuevo proyecto en forma inmediata; veremos otra ventana donde tendremos acceso a todas las herramientas y opciones disponibles para trabajar.

5 **Botón Abrir**: al hacer clic sobre este botón, se despliega un menú que permite abrir un proyecto almacenado con anticipación. También se presenta una serie de sketches de prueba listos para cargar y utilizar.

6 **Botón Salvar**: permite guardar el trabajo que hemos realizado hasta este momento.

7 **Botón Monitor Serie**: muestra los datos que son enviados por Arduino mediante el puerto serie; además, permite enviar a Arduino utilizando el puerto serie. Debemos tener en cuenta que existen alternativas al monitor serie, y en algunas ocasiones puede ser necesario acudir a ellas pues la herramienta incluida en Arduino IDE es muy sencilla.

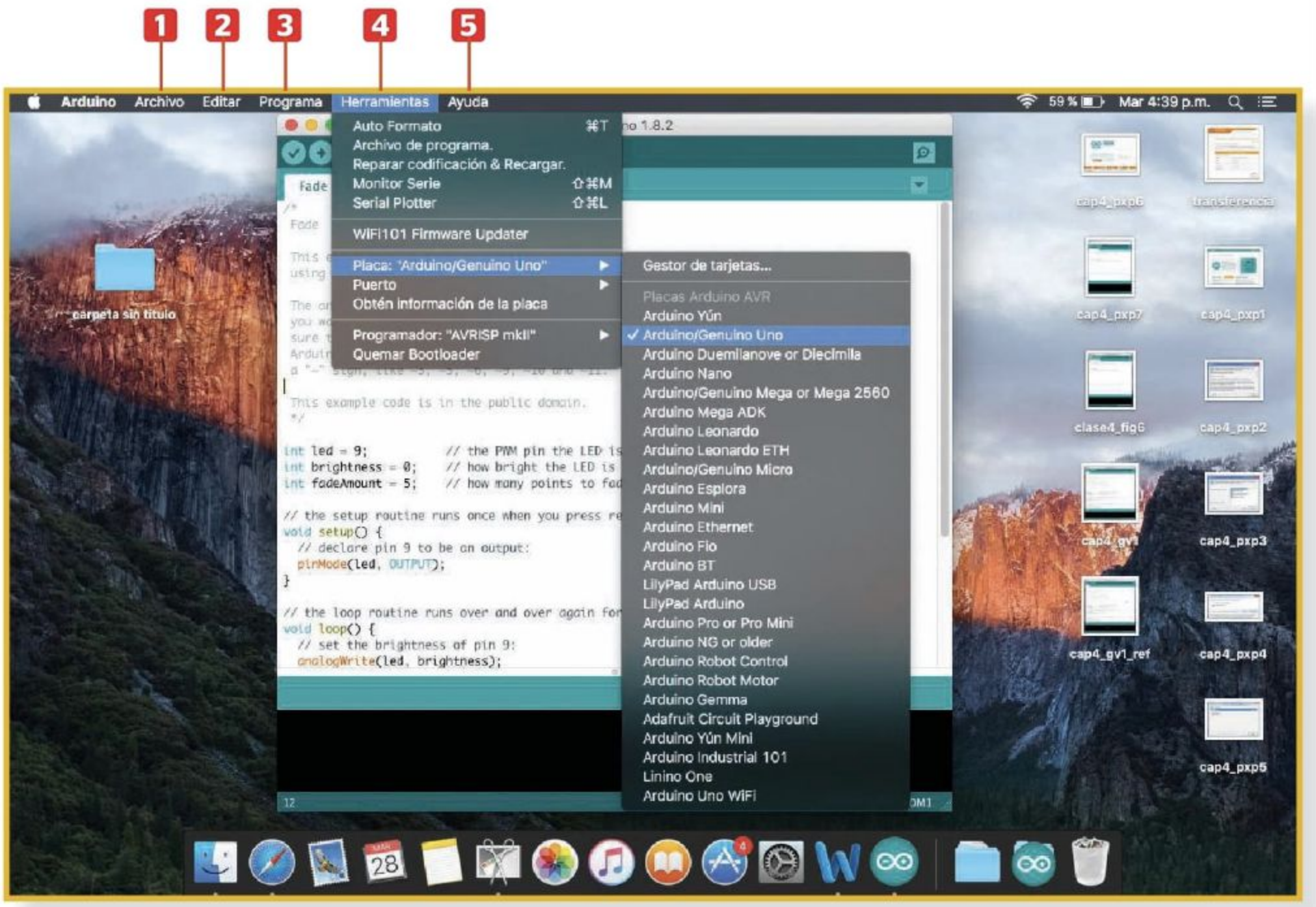
8 **Pestañas**: en esta sección se presentan las pestañas que están abiertas, aquellas que corresponden a los códigos en los que estamos trabajando. Para activarlas es necesario hacer clic sobre su nombre.

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

10 **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.

11 **Consola de error**: en este apartado se muestran los errores que puedan presentarse al verificar el código. Debemos poner atención en esta sección, pues entrega la información necesaria para corregir los posibles problemas.

En la parte superior de la interfaz principal del IDE, veremos la barra de menús, en donde se encuentra el acceso a las herramientas y opciones que utilizaremos:



1 **Archivo**: en este menú están 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.

2 **Editar**: incluye 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.

3 **Programa**: reúne las opciones adecuadas para verificar el código, subirlo a la placa Arduino, exportar binarios compilados, acceder a la carpeta del programa o incluir las librerías necesarias.

4 **Herramientas**: entre las herramientas que encontramos en este menú, están 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.

5 **Ayuda**: brinda 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.

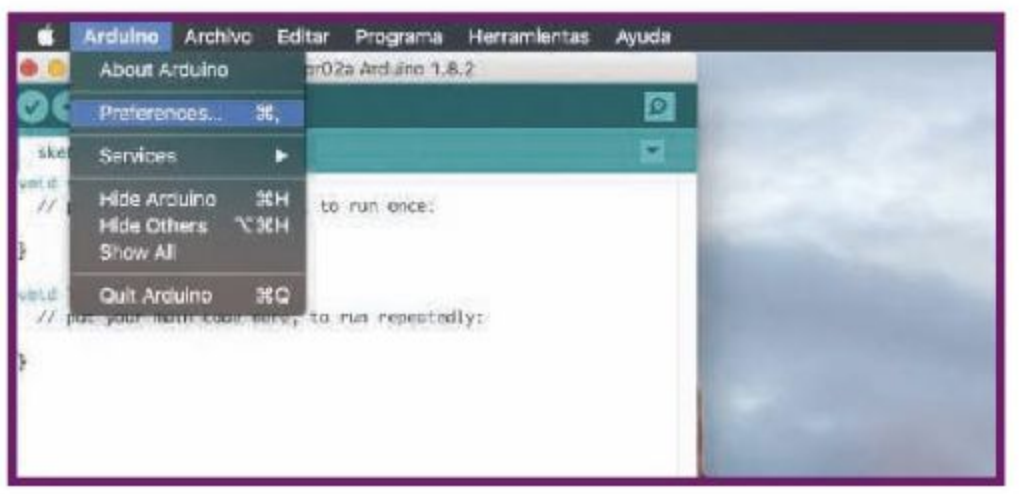
Utilizando el menú Herramientas, podremos elegir, entre una gran cantidad de opciones disponibles, la placa Arduino con la que trabajaremos en el IDE; en este caso, se trata de Arduino UNO.

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 vamos a desarrollar.

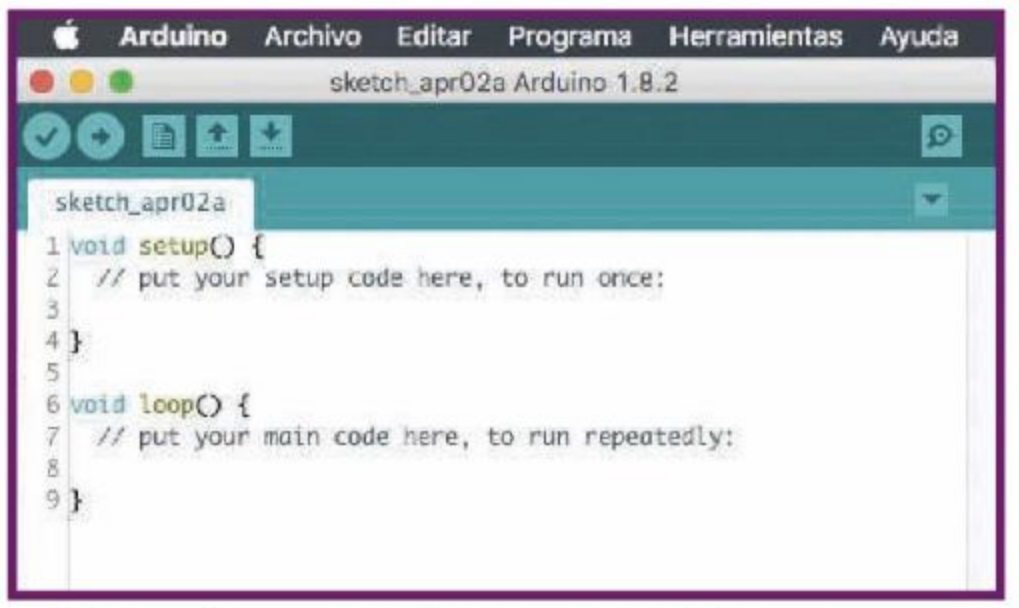
Configuración inicial del IDE

1 Como primera instancia, iniciamos el IDE de Arduino que instalamos siguiendo las instrucciones expuestas en una sección anterior. Cuando aparezca la interfaz principal del IDE, estaremos listos para comenzar el proceso de configuración inicial.



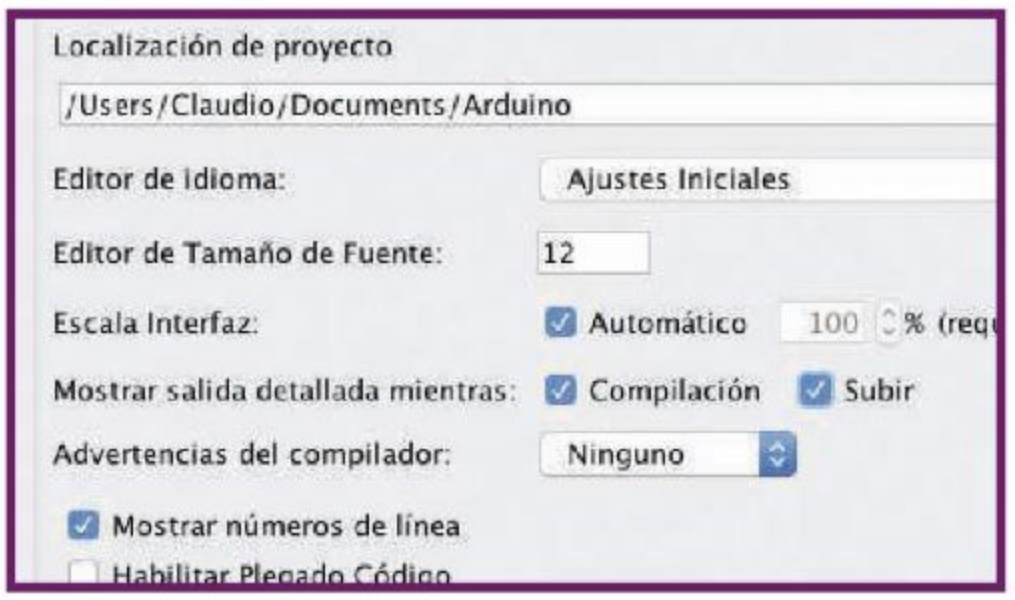
2 Si hemos iniciado el IDE en un sistema Windows, tendremos que hacer clic en el menú **Archivo** y, luego, elegir la opción **Preferencias**. Por otra parte, si estamos en un sistema Mac OSX, presionamos en el menú **Arduino** y, posteriormente, sobre **Preferencias**.

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

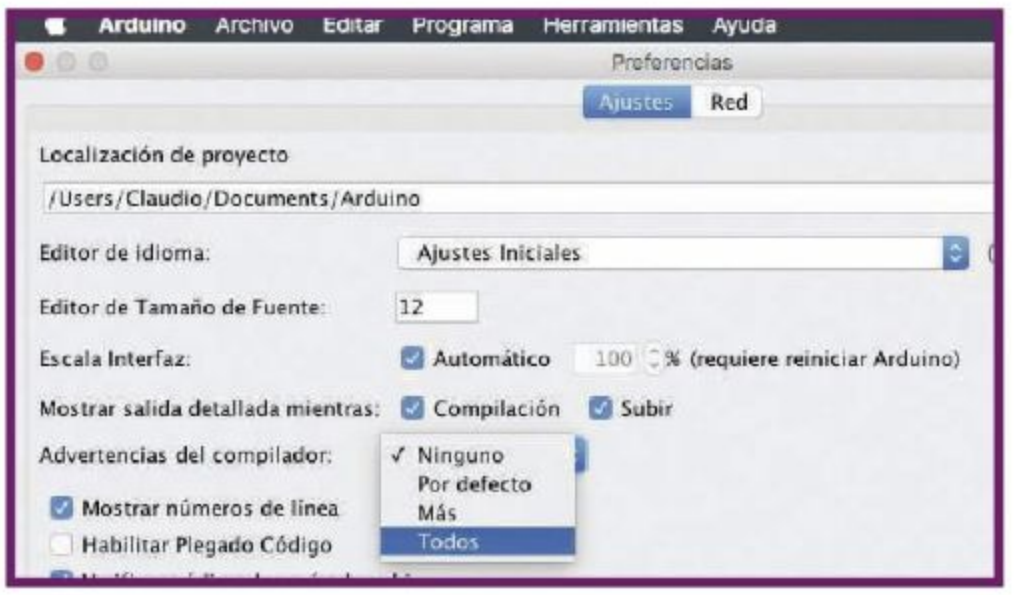


4 Lo primero que haremos será buscar la casilla que corresponde a la opción **Mostrar números de línea**. Esta opción permite ver los números que corresponden a cada línea de los programas que realicemos. La importancia de esto radica en que será más sencillo ubicar una línea específica, sobre todo, cuando trabajemos con códigos extensos.

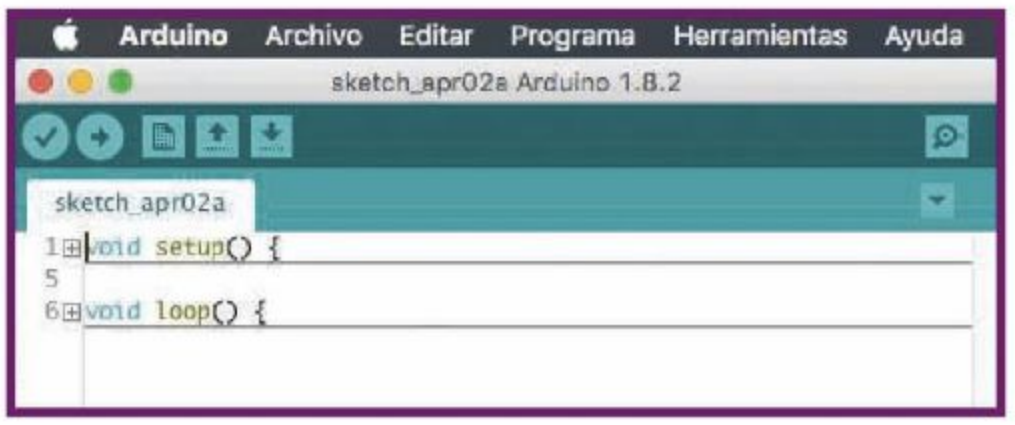
Siempre podemos volver a la sección de preferencias y desactivar aquellas opciones que nos complican la tarea de escribir código o que no nos convencen del todo.



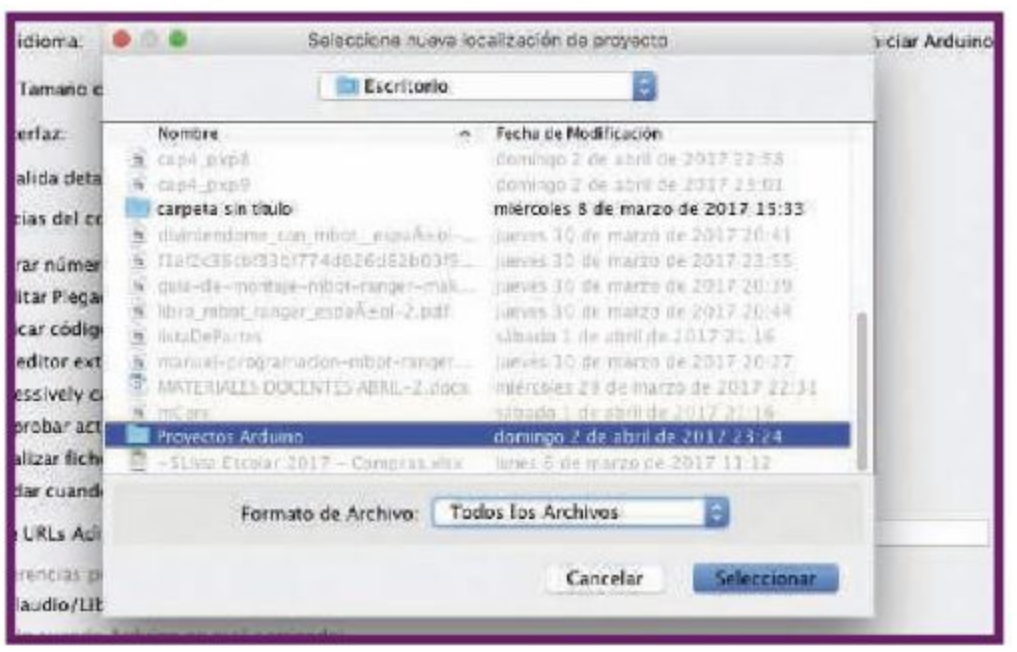
5 Para continuar, buscamos la opción **Mostrar salida detallada mientras**. Junto a ella se presentan dos alternativas: **Compilación** y **Subir**. Marcamos ambas casillas; de esta manera, tendremos acceso a información detallada tanto cuando realicemos la compilación del programa como cuando lo subamos a Arduino; esta salida puede incluir datos importantes para ubicar algún problema o dificultad que pudiera presentarse en el código.



6 Ahora vamos a la opción denominada **Advertencias del compilador**, desplegamos el menú que se encuentra a su lado y elegimos **Todos**. La opción predeterminada es **Ninguno**, que puede ser útil cuando tengamos experiencia en el manejo del IDE, pero si recién comenzamos en la programación con Arduino, bien vale la pena tener a nuestra disposición toda la información que pueda ayudarnos.



7 La siguiente opción que marcaremos será el plegado de código. Esto nos facilitará el trabajo cuando manejemos códigos extensos, pues permitirá agrupar y plegar ciertos segmentos para tener el programa más organizado, hacer que sea más entendible y no tener que navegar mucho tiempo para llegar a la línea de código que necesitamos.



8 Para finalizar la configuración inicial del IDE, elegimos la ruta que utilizaremos para almacenar nuestros proyectos. En forma predeterminada encontraremos una ubicación como la siguiente: **Users/Claudio/Documents/Arduino**, pero podemos elegir la ruta que más nos guste, por ejemplo, **Escritorio/Proyectos Arduino**.

Configuración avanzada

Aunque la configuración inicial del IDE debería de bastar para cualquier usuario principiante, existen opciones de configuración avanzadas que están reservadas para usuarios con mayor experiencia. Para acceder a ellas, debemos abrir `preferences.txt`, en `AppData/Local/Arduino15`. Hay que tener mucho cuidado al editar las opciones de este archivo y realizar las modificaciones solo cuando el programa no se encuentra en ejecución.

Una vez que hayamos aplicado todas las opciones mencionadas, será necesario hacer clic sobre Aceptar, para que se almacenen y estén disponibles para todos los proyectos que iniciemos de aquí en adelante.

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:

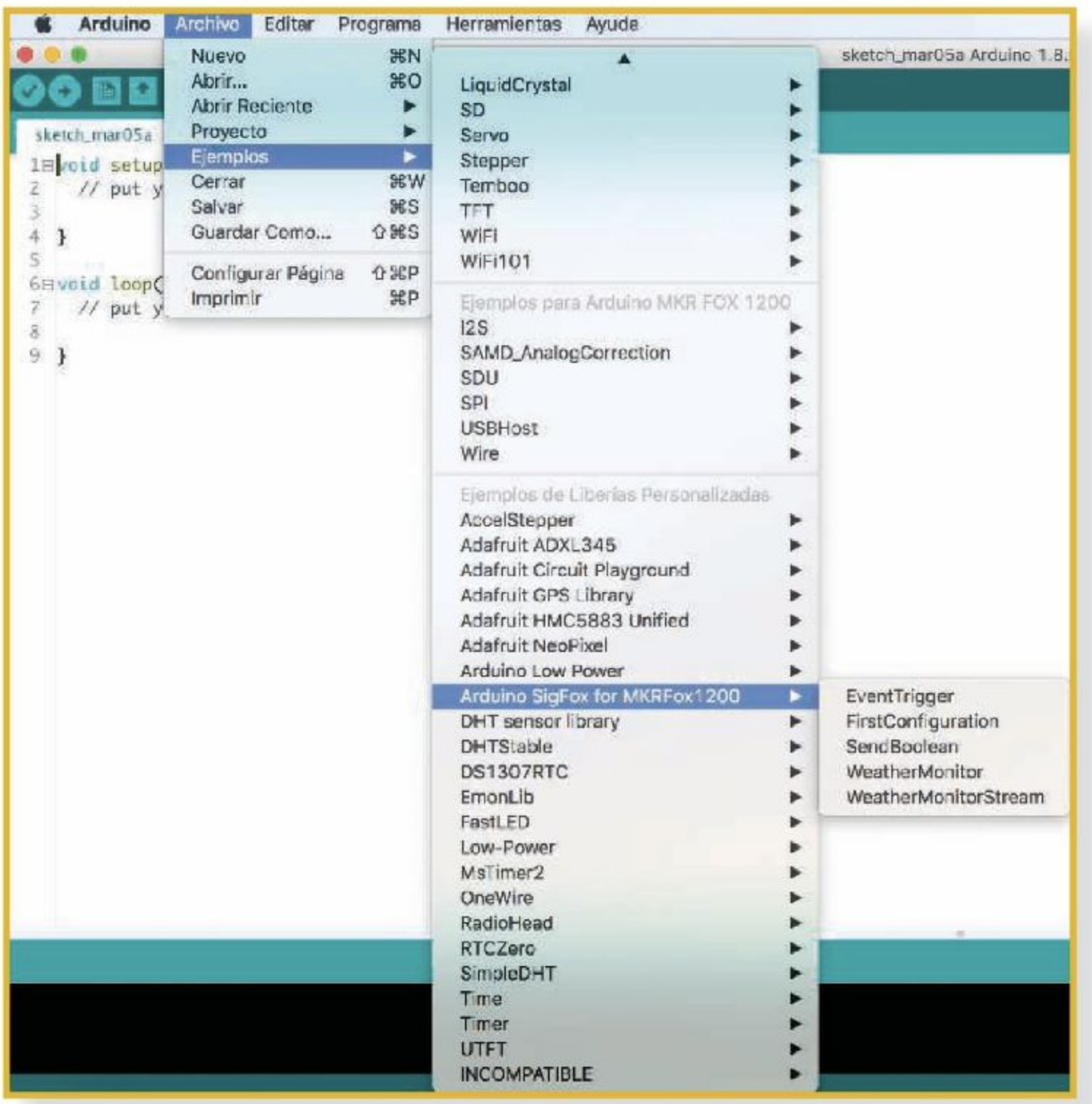
```
*sOut&= ~sBit;
 *rReg&= ~rBit;
 *rOut&= ~rBit;
 *rReg |= rBit;
 *rReg&= ~rBit;
 *sOut |= sBit;
interrupts();

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 **CapacitiveSensor**; en este caso solo necesitamos el siguiente código:

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

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, y así optimizar 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 las cuales es posible mejorar la creación de los programas o sketches. Para simplificar esta idea, mencionaremos que una librería no es más que una colección de funciones que podemos incluir de un modo bastante sencillo y rápido 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 y su propósito es ocultar la complejidad que relaciona al trabajo con el microprocesador; gracias a ella, podemos realizar tareas en forma sencilla sin necesidad de programar cientos de línea de código.

Esta librería base es la que diferencia a la placa de desarrollo Arduino, si la 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**.

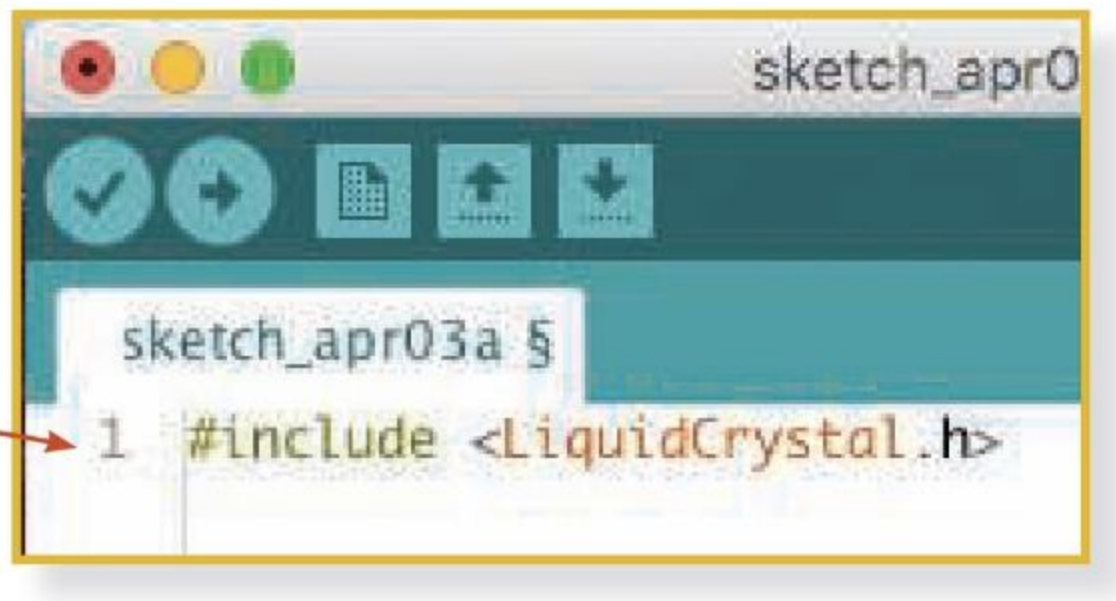
Librerías estándar

Las **librerías estándar** son aquellas que se incluyen en el IDE de Arduino, porque los desarrolladores consideraron que se trataba de funciones necesarias para muchos proyectos; por este motivo, 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 estándar 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.

LIBRERÍAS ESTÁNDAR, JUNTO A SUS RESPECTIVAS DESCRIPCIONES Y EJEMPLOS DE LAS FUNCIONES QUE INCORPORAN.

LIBRERÍA	DESCRIPCIÓN	EJEMPLOS DE FUNCIONES INCORPORADAS
ArduinoTestSuite	Entrega los métodos estándar y las funciones que se pueden utilizar para probar los sketches 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.	**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).
EEPROM	Permite acceder a la EEPROM de Arduino mediante las funciones read y write.	**Read** (lee el valor de un byte almacenado en una posición de la EEPROM) **Write** (escribe un valor en una posición de la EEPROM)

LIBRERÍA	DESCRIPCIÓN	EJEMPLOS DE FUNCIONES INCORPORADAS
SD	Entrega funciones básicas para que la placa Arduino interaccione con las tarjetas SD. Es necesario utilizarla con precaución, pues consume mucha memoria del programa.	**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)
Ethernet	Permite configurar Arduino como servidor para recibir conexiones de clientes, o como cliente, para conectarse a servidores.	**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)
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.	**Begin** (inicializa la librería Firmata) **printVersion** (envía versión del protocolo a la PC), **setFirmwareVersion** (establece la versión del firmware).
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.	**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)
Servo	Se usa para controlar hasta 12 motores servos con Arduino estándar, y 48 con Arduino Mega.	**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	Permite establecer la velocidad de la rotación del motor, el número de pasos que queremos dar y la dirección de estos pasos.	**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)

Las librerías son trozos de código que podemos incluir y utlizar en nuestro sketch de una forma sencilla y sin complicaciones. El uso de librerías nos facilita la programación y permite la abstracción logrando que nuestro programa sea más sencillo de escribir y de mantener.

LIBRERÍA	DESCRIPCIÓN	EJEMPLOS DE FUNCIONES INCORPORADAS
SPI	Proporciona las funciones necesarias para interactuar con periféricos SPI.	**Begin** (inicializa el bus SPI, y pone los pines MOSI y SCK en baja y el SS en alta) **End** (desactiva el bus SPI) **setBitOrder** (establece el orden en el cual se cargan los bits en el bus) **setClockDivider** (establece el divisor de reloj SPI como una fracción del reloj del sistema)
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.	**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)
SoftwareSerial	Es la solución cuando necesitamos más puertas seriales para nuestros proyectos.	**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)

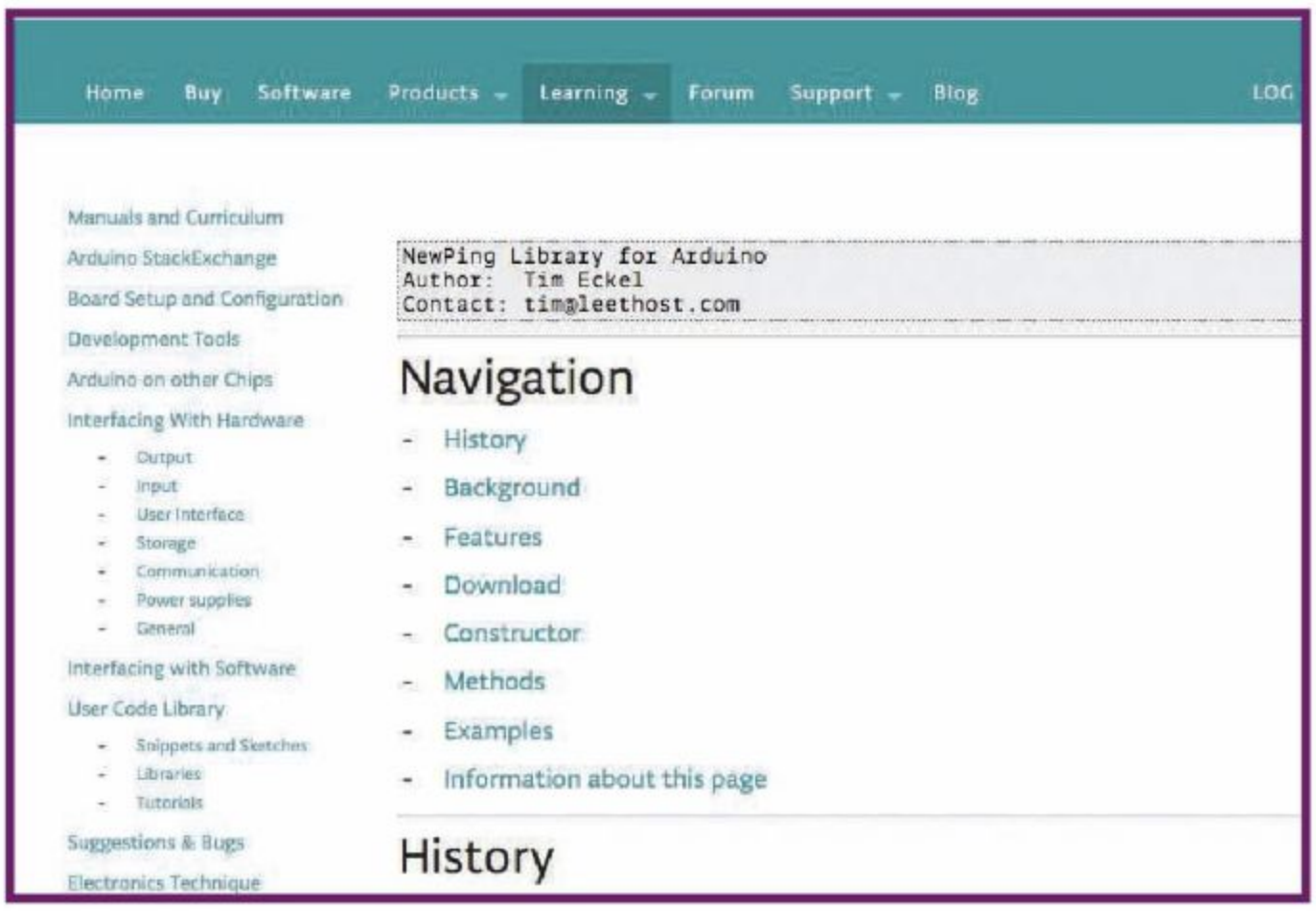
Contribuciones

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 estándar que conocimos en la sección anterior.

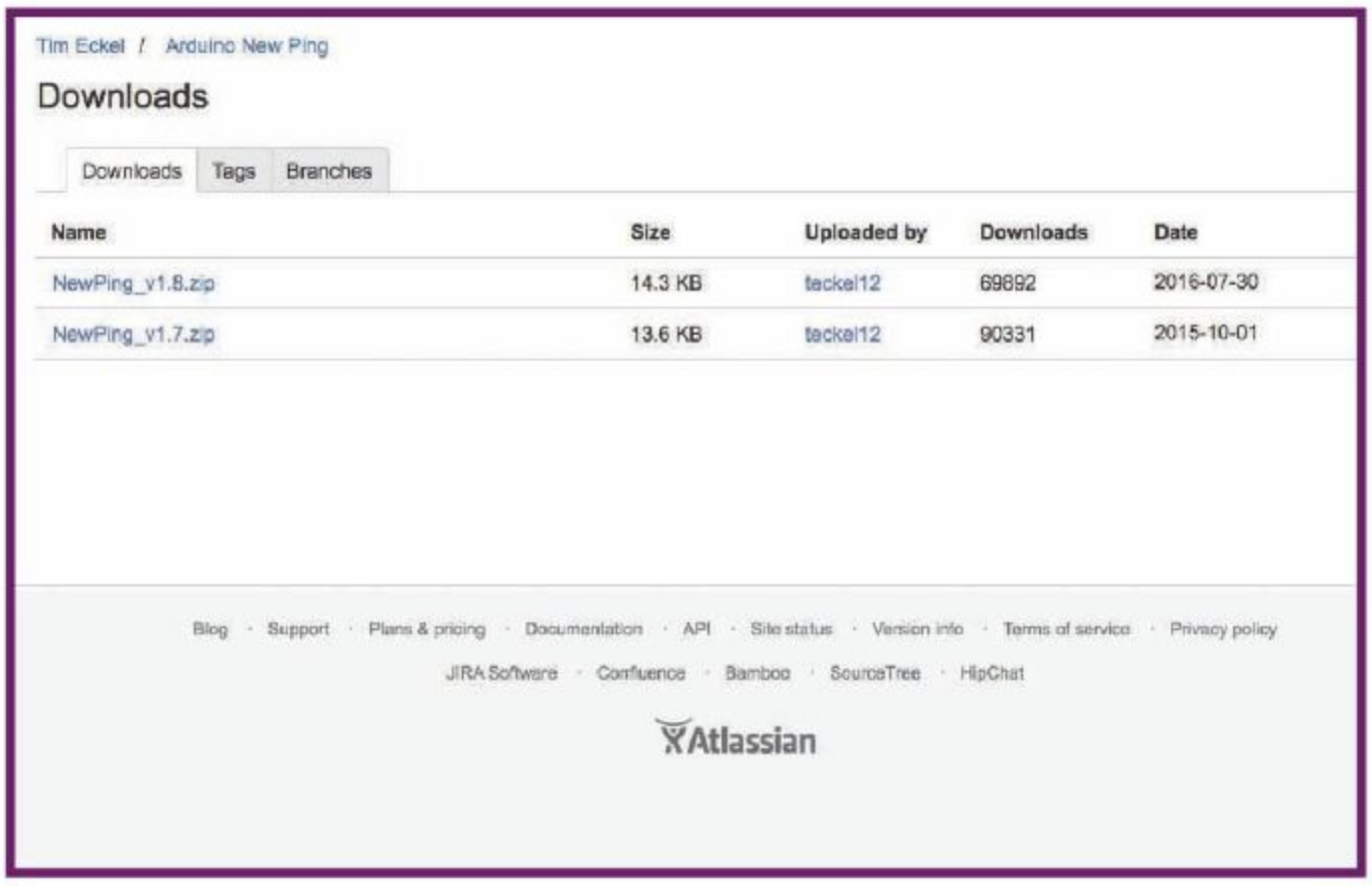
> **Las contribuciones pueden ser agregadas a la colección de librerías estándar si demuestran ser útiles y son requeridas para muchos proyectos.**

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.
A continuación, analizaremos en detalle el proceso de instalación de una nueva librería en Arduino.

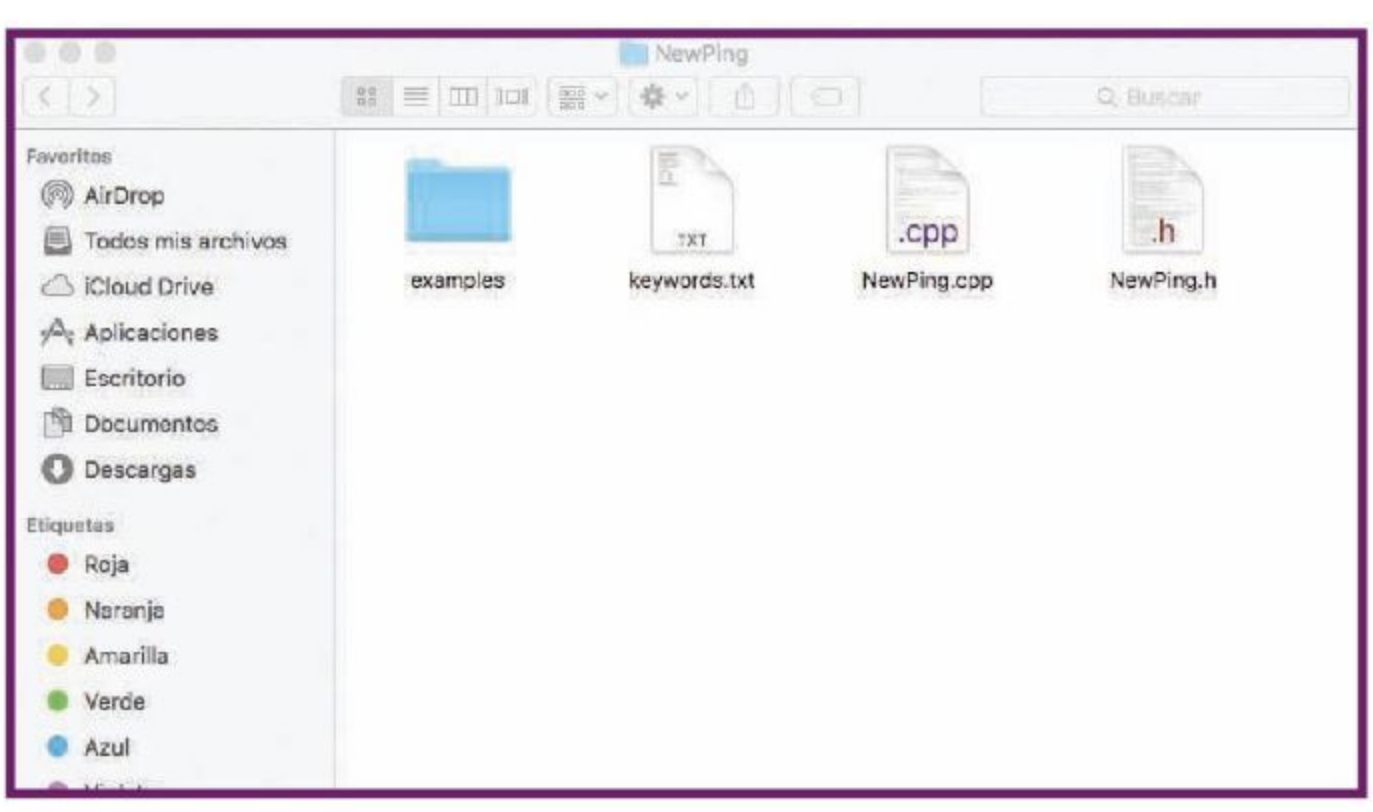
Instalar y utilizar una librería



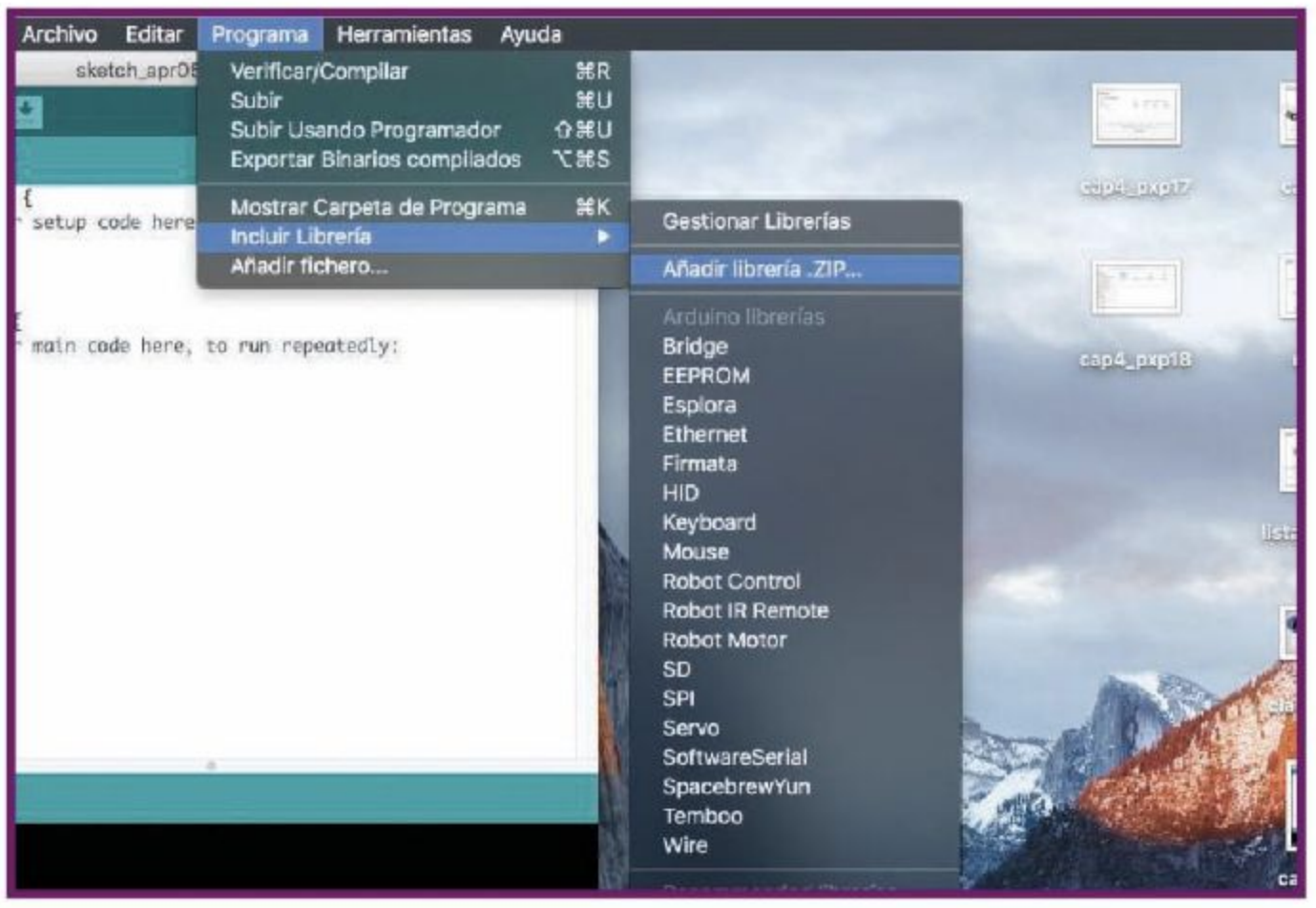
1 En primer lugar, ubicamos la librería que necesitamos en Internet; para este ejemplo utilizaremos NewPing, que nos ayudará en el manejo de varios sensores ultrasónicos. Esta librería en particular se encuentra en la dirección **http://playground.arduino.cc/Code/NewPing**. Hacemos clic sobre **Download**, luego presionamos sobre el enlace **DownloadNewPinglibrary** y descargcamos la librería a la computadora.



2 Para algunas librerías, encontraremos diferentes opciones de descarga; en este ejemplo se ve que se encuentran disponibles las versiones 1.7 y 1.8 de **NewPing Library**, seleccionamos la más actualizada.



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



4 Iniciamos el IDE de Arduino y desplegamos las opciones que se encuentran en el menú **Programa**, seleccionamos **Incluir librería** y luego hacemos clic sobre **Añadir librería .ZIP**. Antes de realizar este paso, tengamos en cuenta el lugar donde descargamos la librería.

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

6 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 podemos agregar esta línea de código en forma automática si desplegamos el menú **Programa**, hacemos clic en **Incluir librería** y posteriormente seleccionamos la librería deseada.

Si el método anterior no funciona, intentaremos agregar la librería en forma manual, para lo cual 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.

Por otra parte, 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:

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

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 las disponibles, y también instalarlas y actualizarlas mediante unos pocos clics del mouse.**

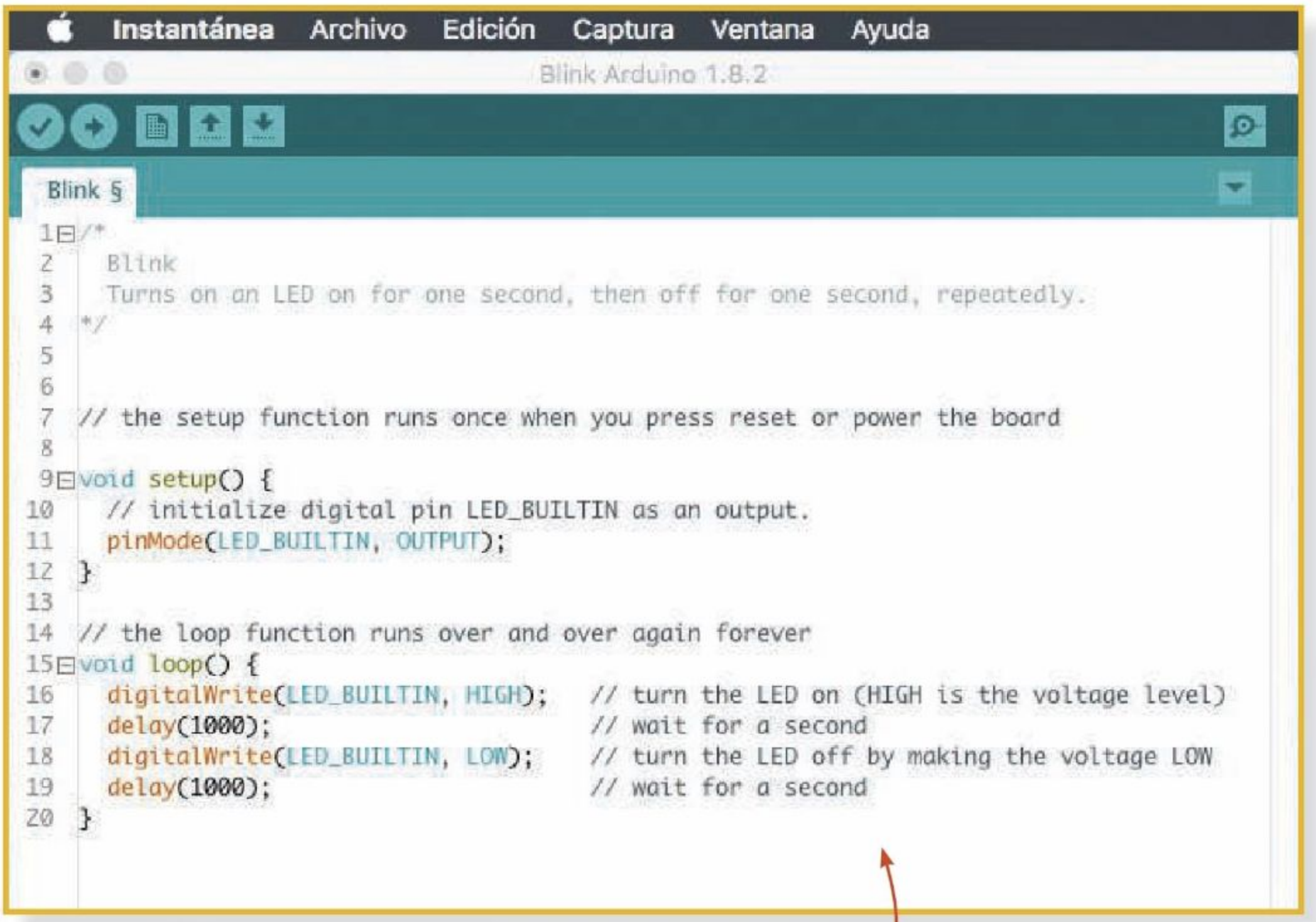
El Gestor de librerías es otra forma de instalar librerías en Arduino; se encuentra disponible en el menú Programa/Incluir librería/Gestionar librerías.

Estructura de un sketch

Un programa para Arduino se denomina sketch y posee la extensión .ino. Un proyecto puede estar compuesto por varios archivos, pero todos deben estar alojados en la misma carpeta donde está el archivo principal. Aquí veremos la estructura básica de un sketch.

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()**.

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.



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

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

Comentarios

En un sketch podemos encontrar líneas de comentarios, es decir, 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. Por otra parte, 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, que veremos en detalle más adelante.

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.

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.**

Aspectos importantes

Existen muchos aspectos relevantes a la hora de escribir código, aquí conoceremos algunos de ellos.

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.

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);
}
```

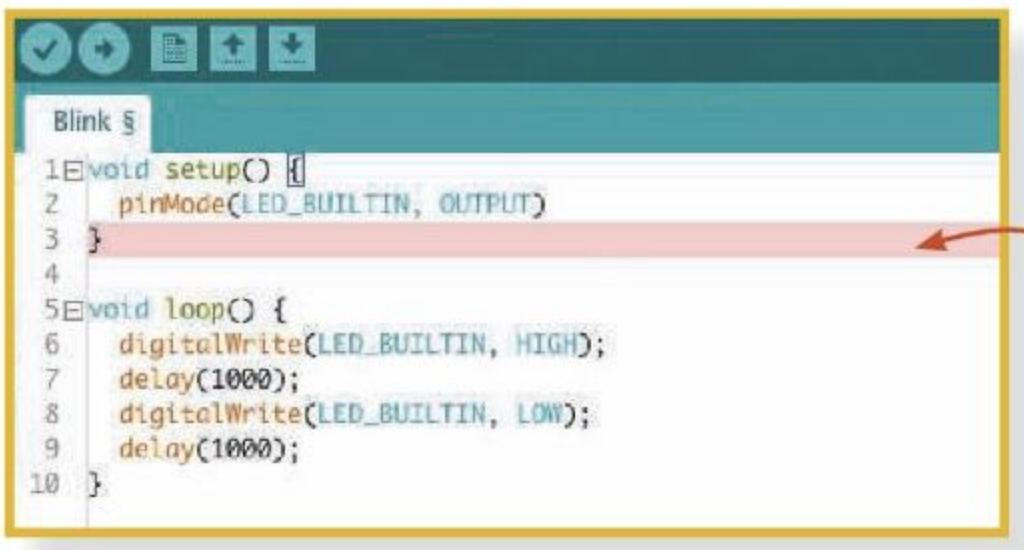
Tabulaciones

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 otro momento. De esta manera, las tabulaciones facilitan la tarea de leer código escrito y permiten 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);
}
```



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.

RedUSERS

COMUNIDAD DE TECNOLOGIA

TODA LA ACTUALIDAD IT

WEB | NEWSLETTERS | REDES SOCIALES

redusers.com

www.facebook.com/redusers
www.twitter.com/redusers
www.instagram.com/reduserscom/

Funciones y parámetros

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:

```
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 esto 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, razón por la cual los paréntesis se escriben vacíos.

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 tener que rescribirlos todas las veces que los necesitemos.

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 que 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 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; y función de** micros(), **que devuelve el número de microsegundos del tiempo.**

Variables

En palabras sencillas, una variable sirve para almacenar datos, que pueden ser consultados y utilizados desde diversos segmentos del sketch. Para usar un dato específico, accedemos a él mediante el nombre de la variable.

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;
```

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**, mientras que en la variable **h** almacenamos el resultado de sumar las variables **f** y **g**.

Constantes

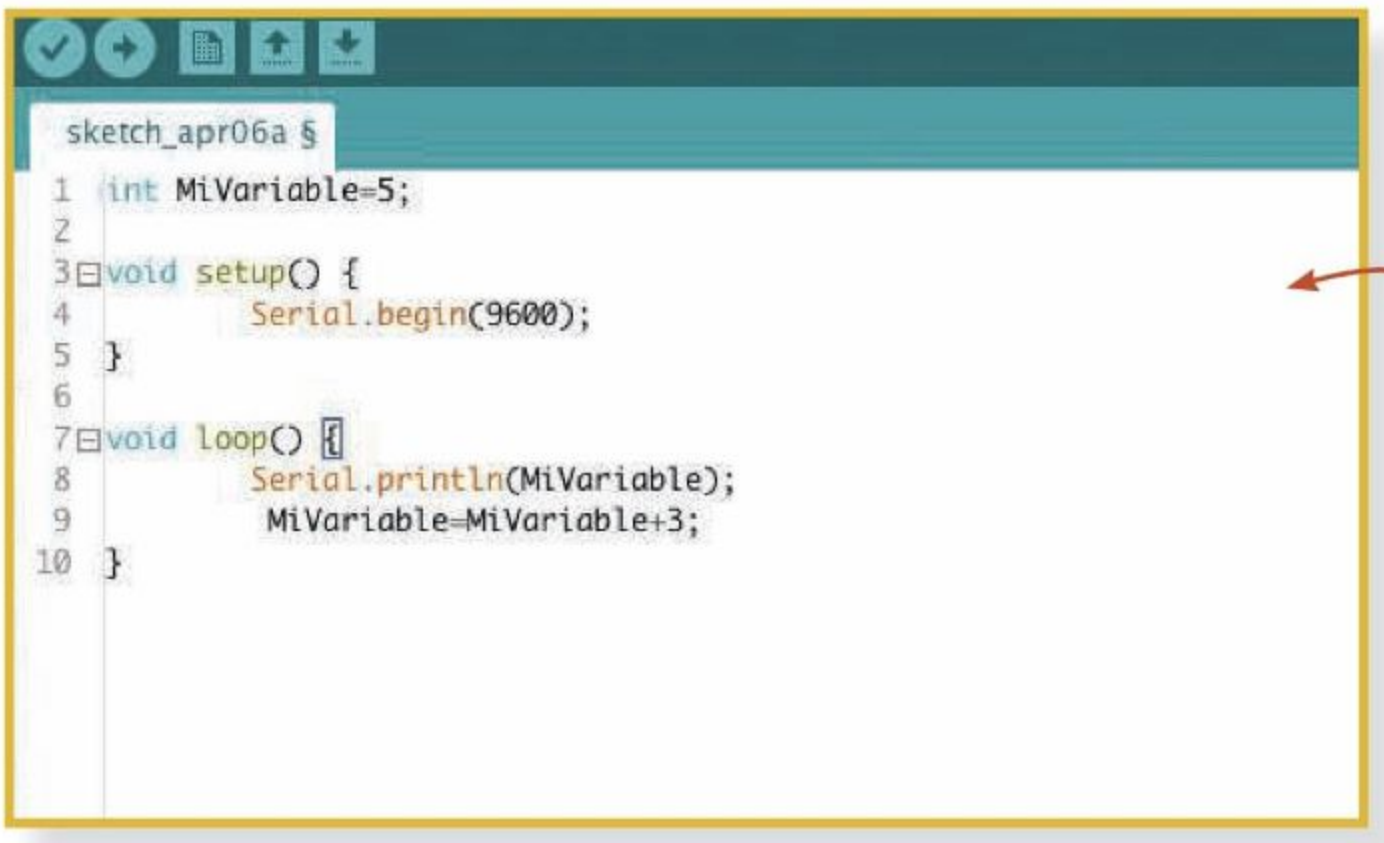
Podemos entender a las constantes como variables de solo lectura, es decir, no podemos modificar el valor que les fue asignado; si intentamos hacerlo, el compilador nos mostrará un error. La definición de una constante se hace de forma similar a una variable, pero debemos preceder su declaración con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

Ámbitos

Las variables pueden ser globales o locales, y 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:

```
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 que su ámbito es global y podemos utilizarla 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 **5**.

Por otra parte, una variable es local cuando se declara dentro de un ámbito específico (dentro de corchetes) y, por lo tanto, no podrá usarse en otro 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 que 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:

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

}

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

Declarar variables

Como vimos, la declaración de variables no es una tarea compleja, pero, de igual modo, 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 uno u otro lugar. Por otra parte, también es relevante darle un nombre descriptivo, de manera que será más fácil reconocerla en códigos extensos; por ejemplo, es mejor una variable denominada SensorDeDistancia **que una llamada** VariableDos.

RedUSERS PREMIUM

- Nuestros expertos comparten su saber y experiencia
- Calidad y cantidad por una mínima cuota mensual
- Cientos de publicaciones con los temas que más interesan
- Siempre, donde vayas. On Line – Off Line. En cualquier dispositivo
- Al menos 1 novedad semanal. Lee todo lo que desees sin límites

SUSCRÍBETE

usershop.redusers.com

+54-11-4110-8700

usershop@redusers.com

Datos y operadores

Dos conceptos importantes 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 forma que obtendremos resultados.

En primer lugar, analizaremos los diferentes tipos de datos con los que podemos trabajar. Este tema se encuentra íntimamente relacionado con las variables que conocimos anteriormente, 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 siguiente tabla:

TIPOS DE DATOS COMÚNMENTE UTILIZADOS 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.
unsignedint	**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.

En programación, un tipo de dato es un atributo de los datos que indica la clase de datos con que se va a trabajar. Esto incluye imponer restricciones en los datos, como qué valores pueden tomar y qué operaciones se pueden realizar.

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 OFRECIDOS POR EL CORE DE ARDUINO, Y ALGUNOS EJEMPLOS ESPECÍFICOS JUNTO A SU DESCRIPCIÓN

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 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.

Un operador es un elemento de programa que se aplica a uno o varios operandos en una expresión o instrucción. Se trata de un símbolo que indica al compilador que se lleven a cabo ciertas manipulaciones matemáticas o lógicas.

Tipos de operadores

Existen distintos tipos de operadores; a continuación analizaremos sus características para entender en qué situaciones conviene utilizarlos.

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, tendremos que 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**.

Los operadores lógicos o booleanos se utilizan usualmente como una forma de comparar dos expresiones y devuelven como resultado un VERDADERO o FALSO dependiendo del operador específico que estemos utilizando.

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**.

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, que 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.

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.

Estructuras de control

Las estructuras de control son elementos básicos que todo programador debe conocer. Nos permiten programar la toma de decisiones basándonos en los datos que se encuentran disponibles en un momento determinado. Para tomar estas decisiones, elegiremos como base los tests 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 verdadero o true
}
```

Por ejemplo, podríamos desarrollar un sketch que se encargará 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 verdadero 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 lo cual integraremos **elseif**.
La función de esta modificación es permitirnos detallar el control que ocurre en el bloque **if**.
Gracias a **elseif**, podemos introducir 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 tests 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, y 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 declaramos la variable v sin un valor inicial y luego 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.

Bucles

Los bucles o loops son capaces de realizar una tarea en forma repetitiva hasta que esta se considere completada; 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.

Inicialización

Cuando trabajamos en un bucle for**, debemos tener en cuenta que la inicialización solo sucederá una vez, cuando el bucle comience, aunque el test se realiza cada vez que el bucle itera o se ejecuta. Si el resultado es verdadero, el bloque de código entre llaves se ejecutará y el valor del contador se incrementará (++) o decrementará (–), dependiendo de lo que hayamos indicado en la primera línea del bucle. Las rutinas seguirán ejecutándose hasta que el resultado del test sea** false**, en nuestro ejemplo, hasta que** i **alcance el valor** 100.

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 while nos permite realizar múltiples iteraciones basándonos en el resultado de una expresión lógica que puede tener como resultado un valor verdadero o falso.

El bucle **do while** se diferencia porque testea la expresión después de ejecutar el bucle; de esta forma, podemos asegurarnos 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.

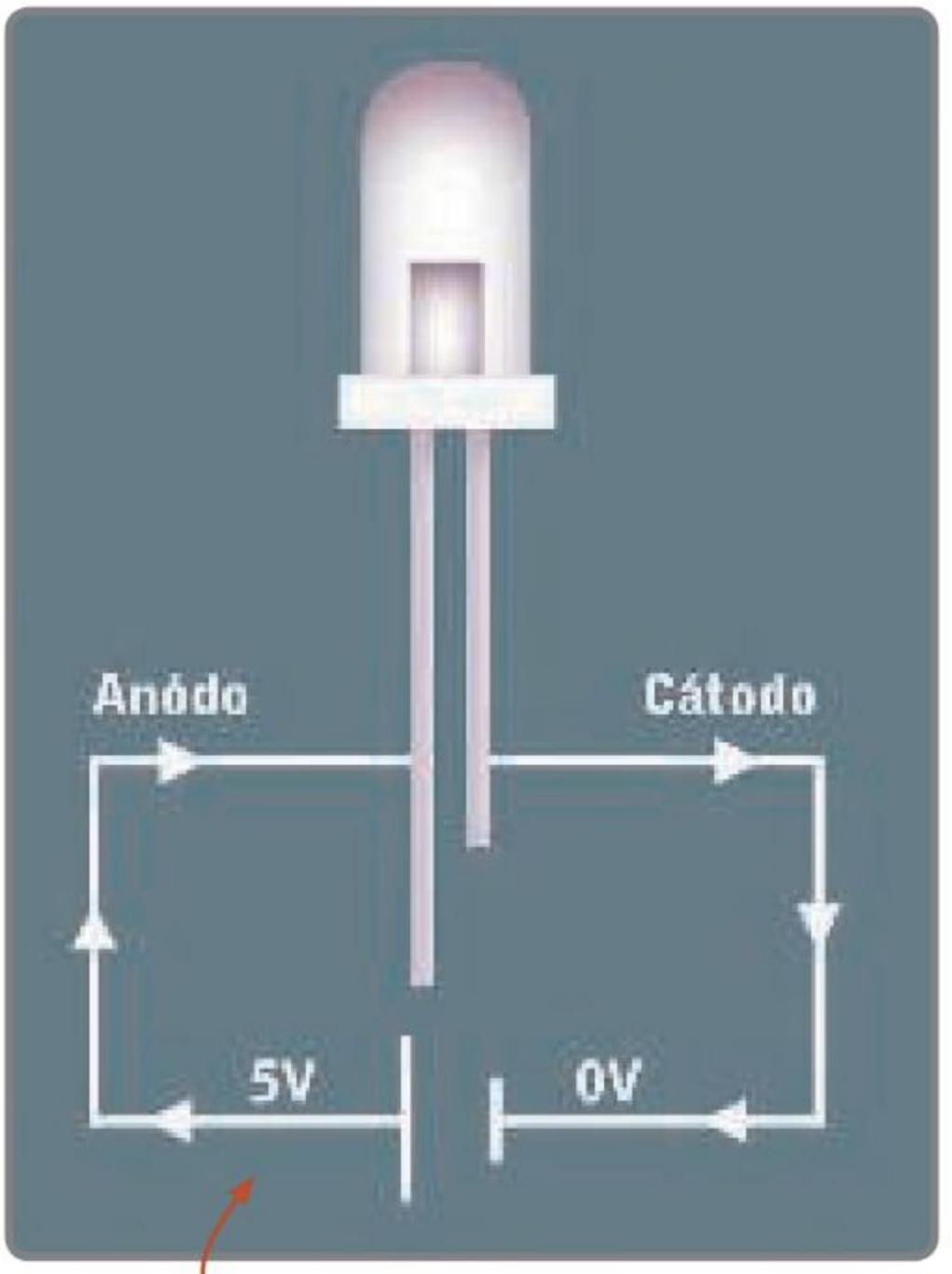
- **Goto**: se trata de una sentencia que nos permite marcar el lugar del código al que realizaremos un salto. La posición desde la que saltamos se almacena en la pila del programa para que sea posible regresar.

- **Return**: es una sentencia que se utiliza para volver de un salto realizado con goto.

LEDs y Arduino

Los LEDs son 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.

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, una 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 negativo.



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

Resistencias

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, y por esta razón es necesario que miremos sus características técnicas.

Ley de Ohm aplicada

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

Para calcular el voltaje: **V = I X R**
Para calcular la intensidad: **I = V / R**
Para calcular la resistencia: **R = V / I**

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 esto aplicamos la ley de Ohm y, como necesitamos dos de las magnitudes, tenemos lo siguiente:

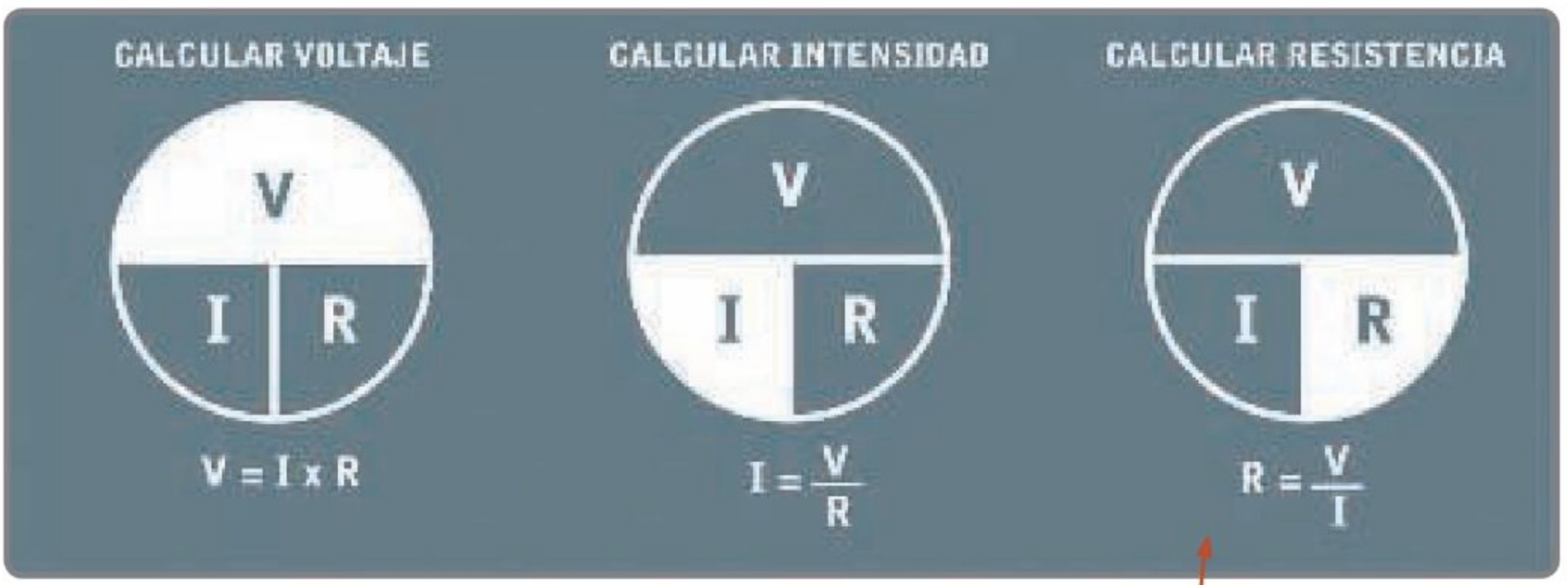
- Voltaje: los pines de la placa suministran 5 V.
- Intensidad: podemos obtenerlo en la hoja de especificaciones o en la web oficial de Arduino; para la placa Arduino UNO se trata de 20 mA.

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.



En esta imagen vemos un resumen de las magnitudes que es posible calcular mediante la ley de Ohm.

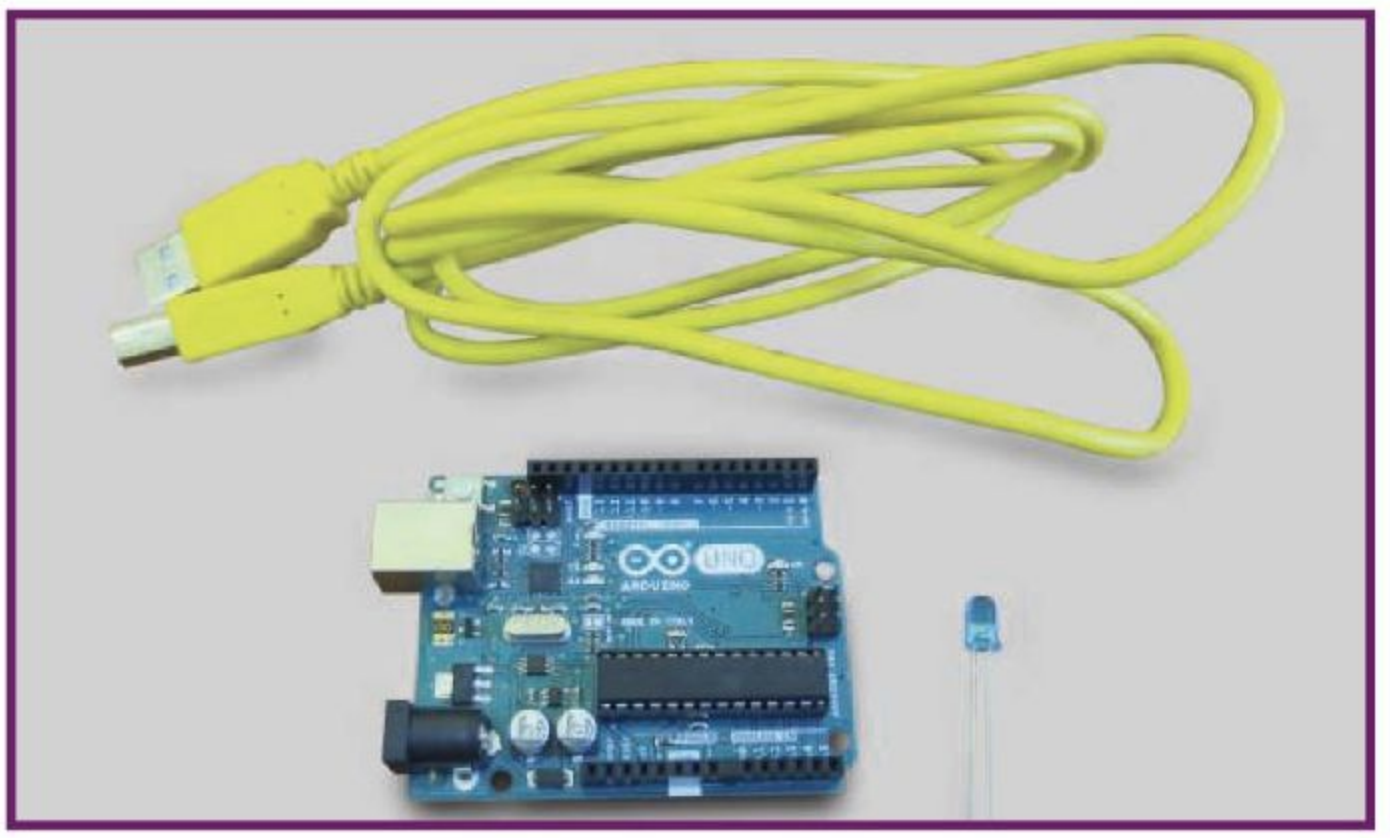
Con estos conceptos claros, pongamos manos a la obra.

Conexión básica de un LED

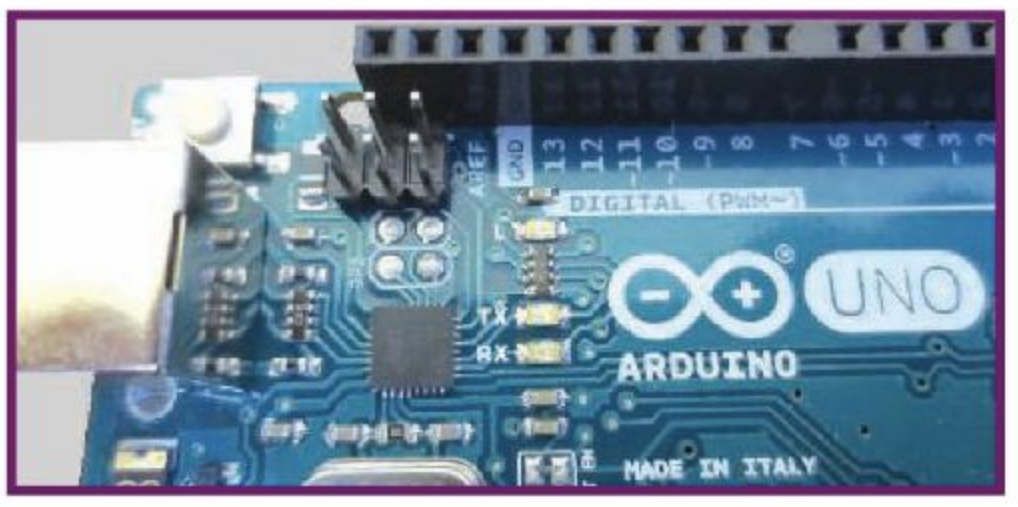
Ya conocimos la forma de cargar un ejemplo básico en nuestra placa Arduino UNO; para esto hicimos uso de Blink, uno de los códigos que se encuentran disponibles en el IDE. En esta ocasión trabajaremos con este código ya cargado en la placa, y lo profundizaremos y modificaremos para obtener diferentes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

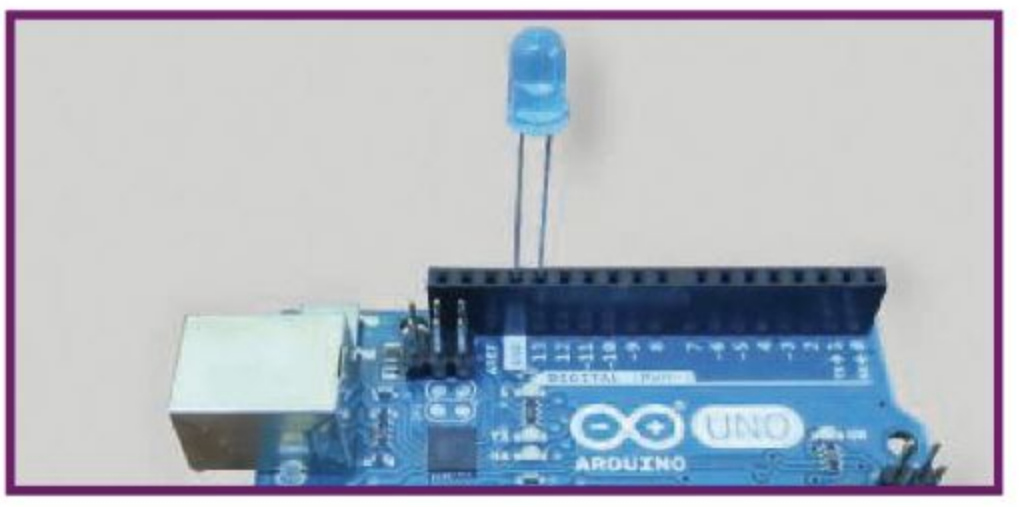
Conectar un LED directamente



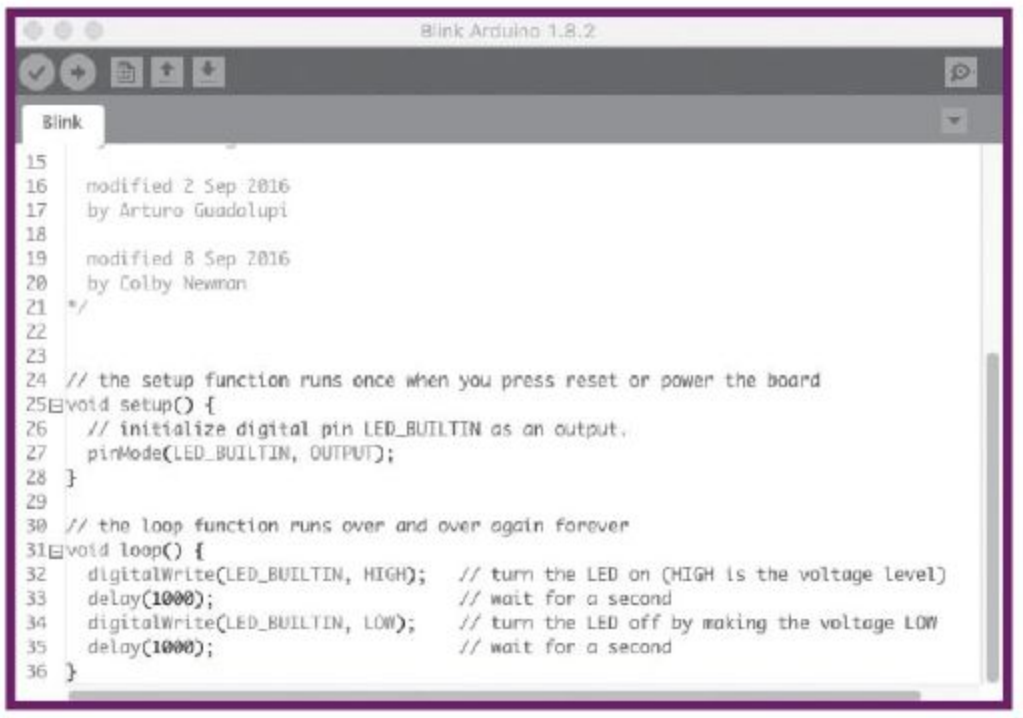
1 Como ya se completaron los pasos necesarios para conocer la placa Arduino UNO, se instaló el IDE y se conoció la forma correcta de cargar un sketch en la placa, en esta ocasión se revisará cómo se conecta un LED en forma directa. Necesitaremos la placa Arduino UNO, un LED y el cable USB para conectarla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



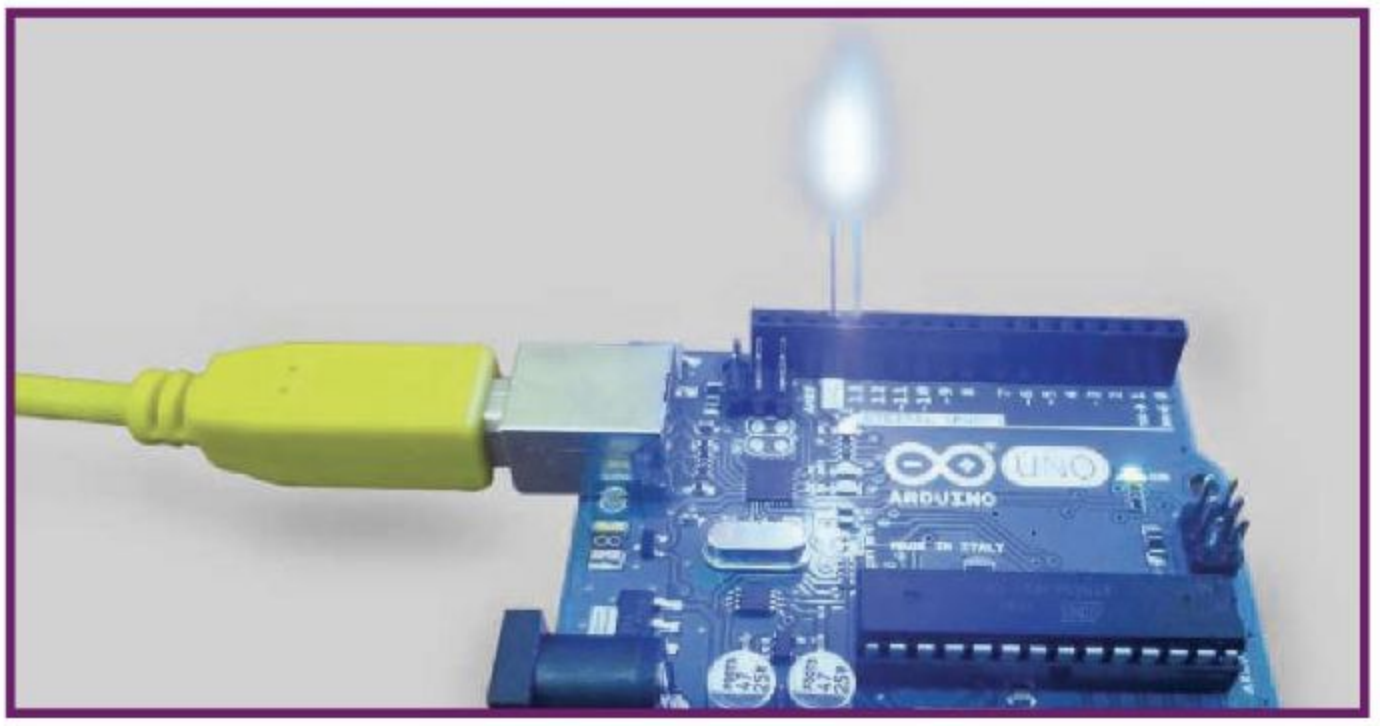
2 Antes veremos la forma de conectar un LED mediante el protoboard, esta vez, en forma directa. En este punto es necesario mencionar que la placa ya dispone de un LED incorporado, por lo que es posible testear código aun si no se conecta un LED externo. Este LED se encuentra indicado con la letra L.



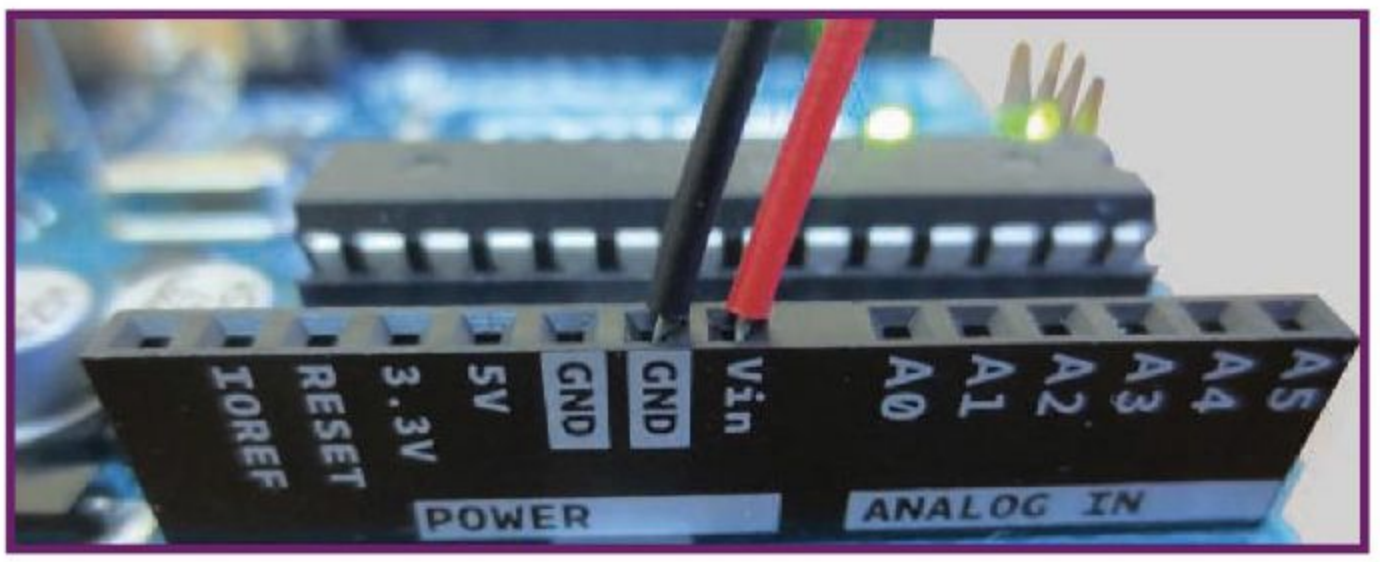
3 Para realizar la conexión de un LED en forma directa, conectamos la pata más larga o positivo al pin digital 13, y la pata más corta, al ping GND. Tengamos en cuenta que la placa ya incorpora una resistencia en el pin 13, por lo tanto, no se dañará el LED conectado.



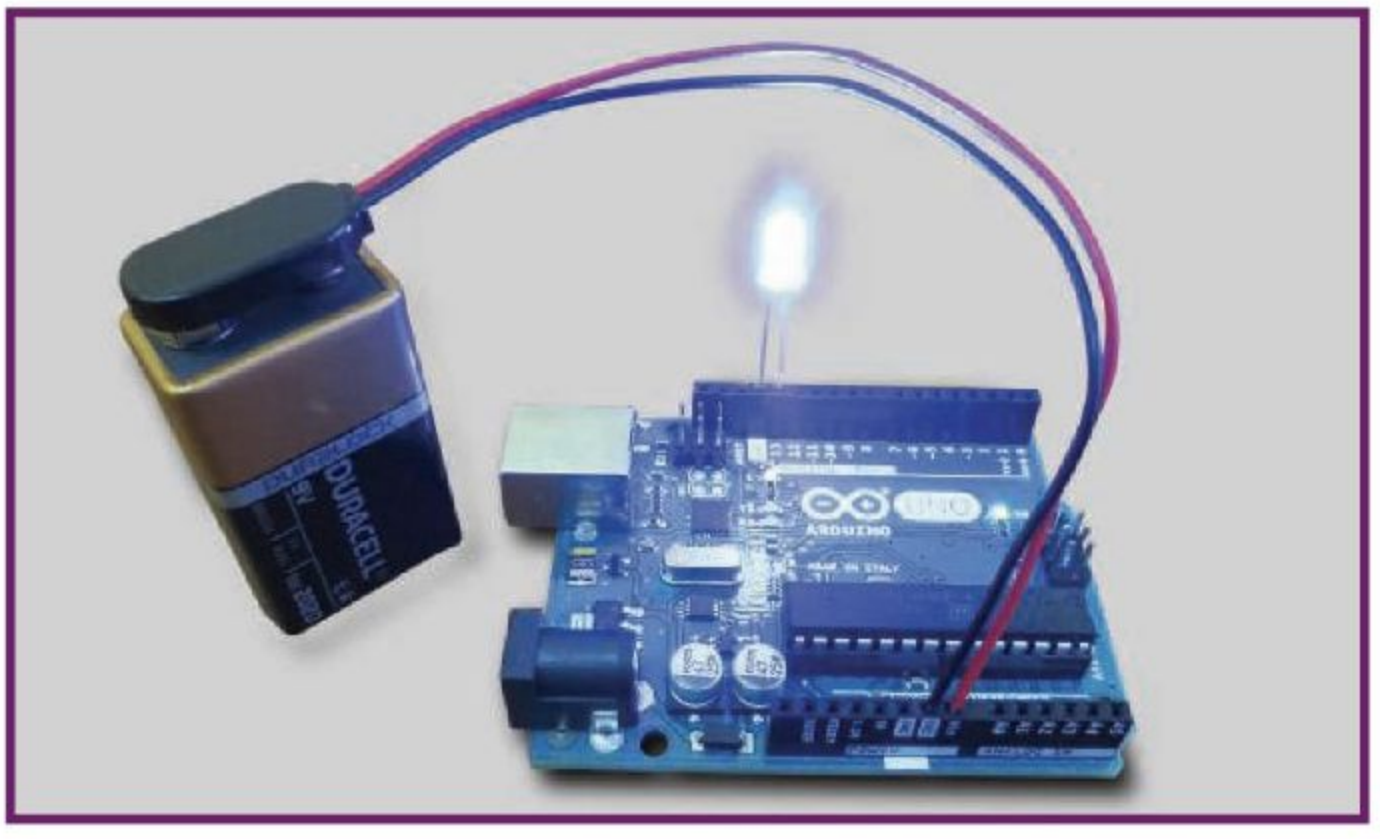
4 Ejecutamos el IDE de Arduino y cargamos el ejemplo Blink; se trata del mismo código que se presentó antes, por ahora no realizamos modificaciones en el código.



5 Conectamos la placa a la computadora mediante el cable USB y cargamos el sketch en la placa Arduino UNO. De inmediato veremos que el LED parpadea, según las indicaciones del sketch Blink.



6 Ahora desconectamos 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, utilizamos los pines GND y Vin.



7 Con la batería conectada a la placa Arduino, se seguirá ejecutando el sketch que guardamos; de esta forma, el LED conectado directamente se encenderá y apagará según el código, 1 segundo encendido y 1 segundo apagado.

Podemos simplificar este código para hacer uso de lo mínimo que nos permita encender el LED:

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

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.

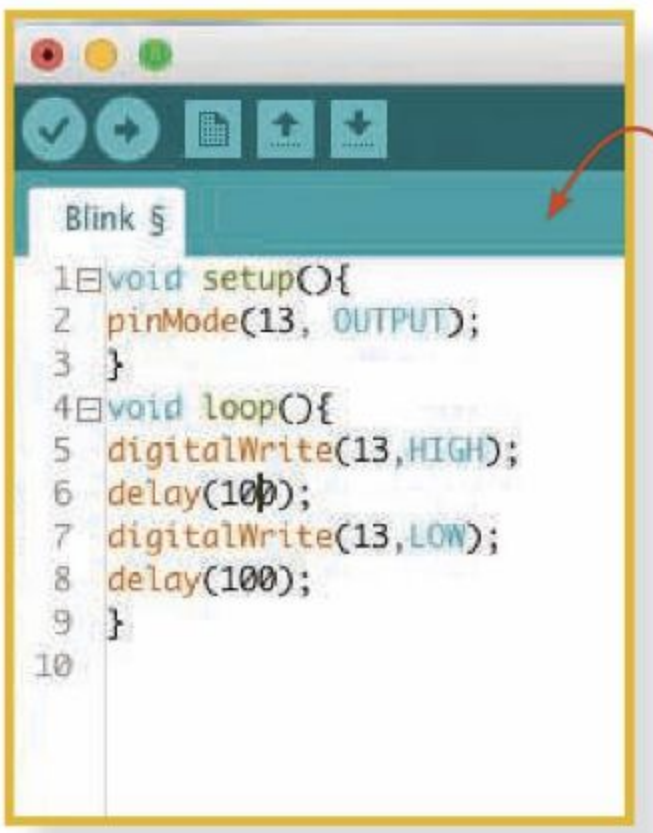
Como sabemos, en **setup()** es posible establecer el uso que le daremos al microcontrolador o iniciar las variables que necesitamos en el sketch. Por otra parte, en **loop()** podemos establecer los procesos que se ejecutarán en forma repetitiva.
De esta forma, en este código de ejemplo usamos **setup()** 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 el 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).

Si utilizamos un pin distinto del 13, será necesario conectar una resistencia de protección, para no dañar el LED conectado.

¡AQUÍ PUEDES ENCONTRARLO!

USERSHOP

Simple, rápido y eficiente

- TODOS LOS PRODUCTOS USERS Y USERSLife, INCLUSO LOS AGOTADOS
- RETIRO EN LOCAL. ENVIO A SUCURSAL O A DOMICILIO
- MULTIPLES MEDIOS DE PAGO
- OFERTAS EXCLUSIVAS

¡TE ESPERAMOS!

usershop.redusers.com

+54-11-4110-8700

usershop@redusers.com

¿Cómo controlar el LED?

Ya sabemos cómo encender un LED conectado a Arduino y, también, cómo controlar su parpadeo. 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.

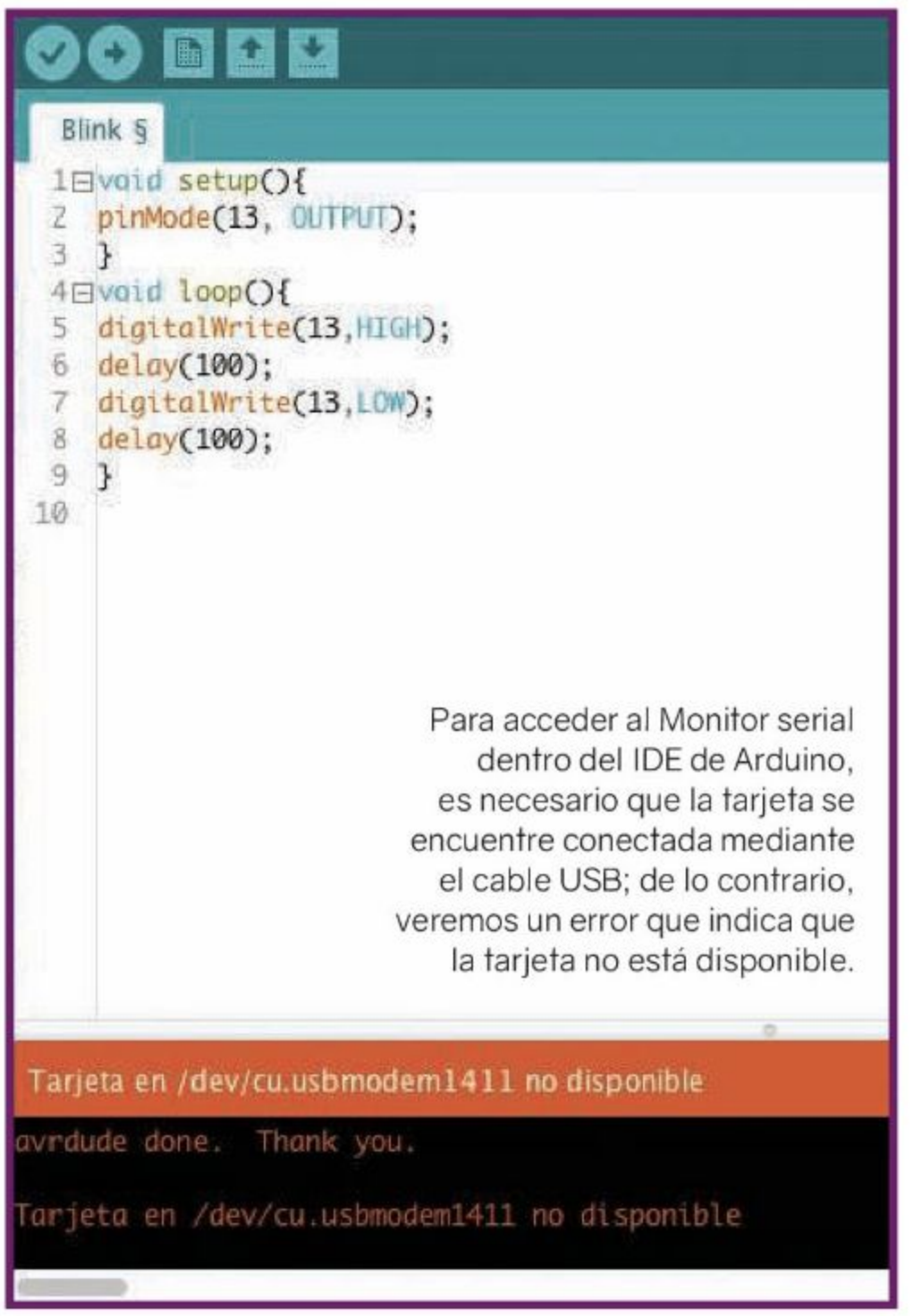
Para este ejemplo seguiremos utilizando el LED que conectamos directamente a la placa Arduino UNO, mediante el pin 13.
Pero, antes de continuar con las instrucciones necesarias, debemos detenernos para conocer el **Monitor serial**. Se trata de una herramienta ofrecida por el IDE de Arduino, que nos 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, razón por la cual debemos utilizarlo para controlar el LED.

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 e símbolos por segundos) 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.



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.

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.

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. Para comenzar, declararemos el uso del pin 13:

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

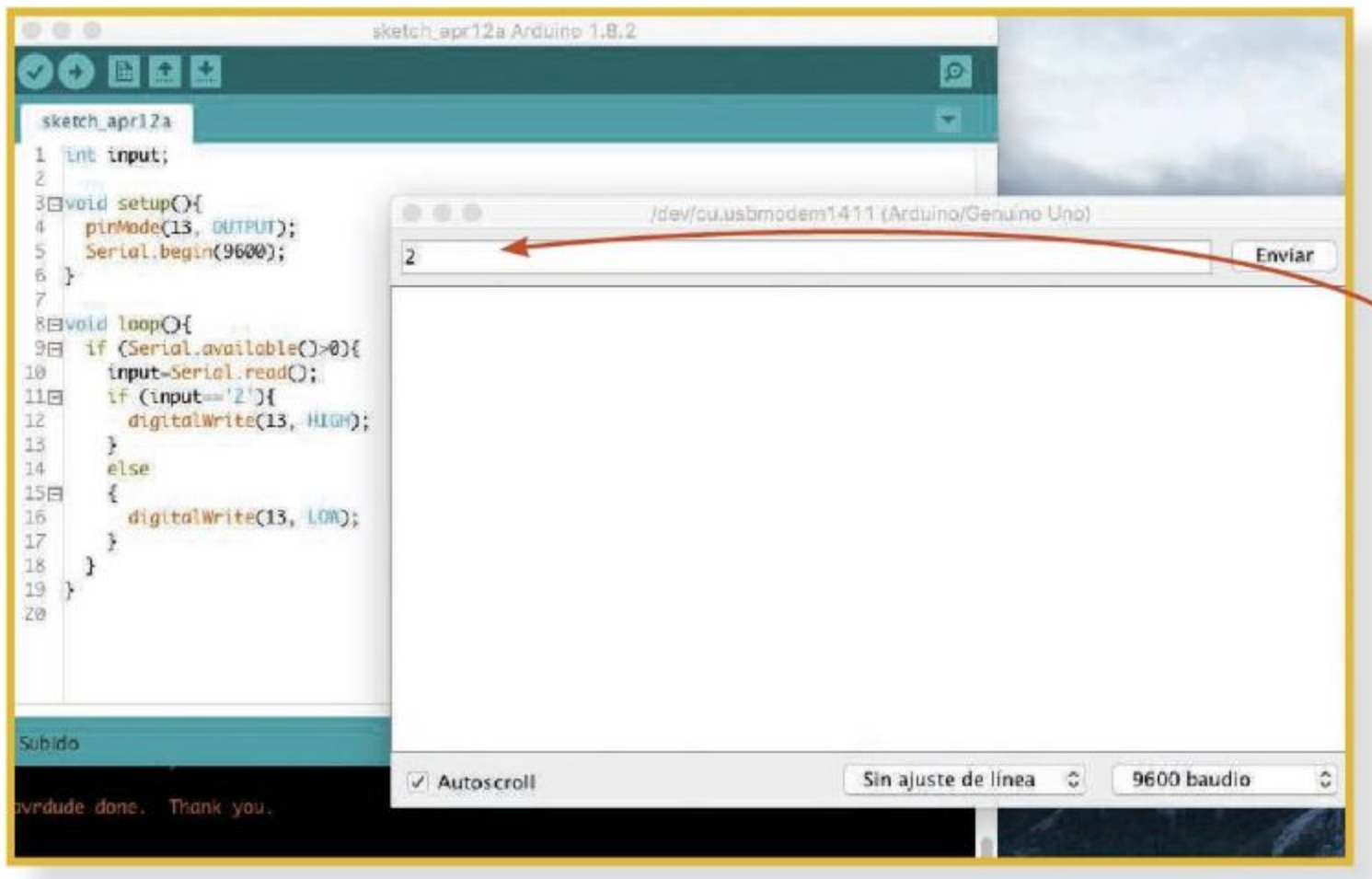
Luego, mediante un **if else** verificamos el valor de **input**. En este ejemplo, encenderemos el LED si el valor es **2**; de lo contrario, lo apagaremos:

```
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.

Agregar 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 manera:

```
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:

```
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. Después especificamos las instrucciones que se ejecutarán mientras la condición sea 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 usamos para encender un solo LED de modo 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++**.

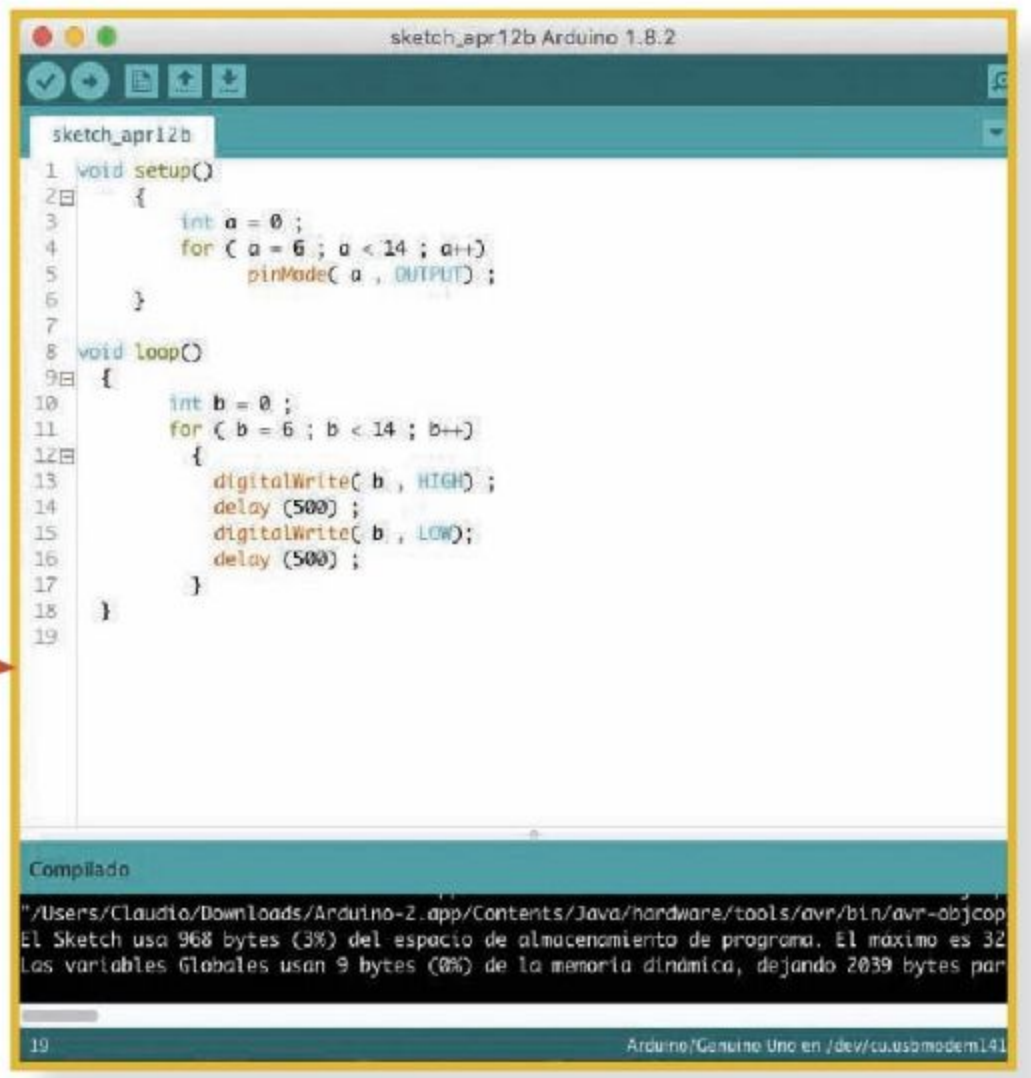
Dentro del ciclo, encontramos las instrucciones que serán ejecutadas en cada iteración:

```
// 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 que el resultado será que todos los LEDs conectados se encenderán y se apagarán. El 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.

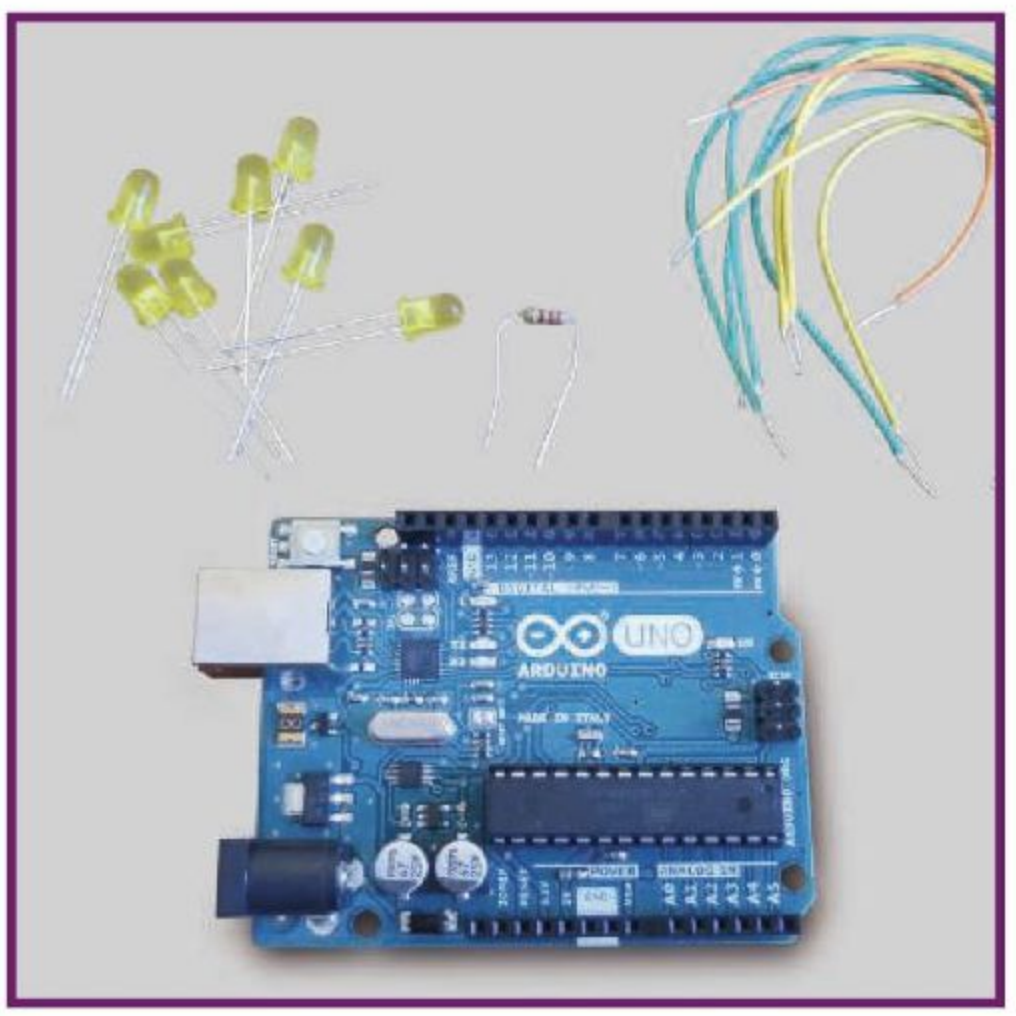


Luego de escribir el código, hacemos clic en **Verificar/Compilar**, que se encuentra en el menú **Programa**. Como vemos en esta imagen, la compilación ha sido correcta.

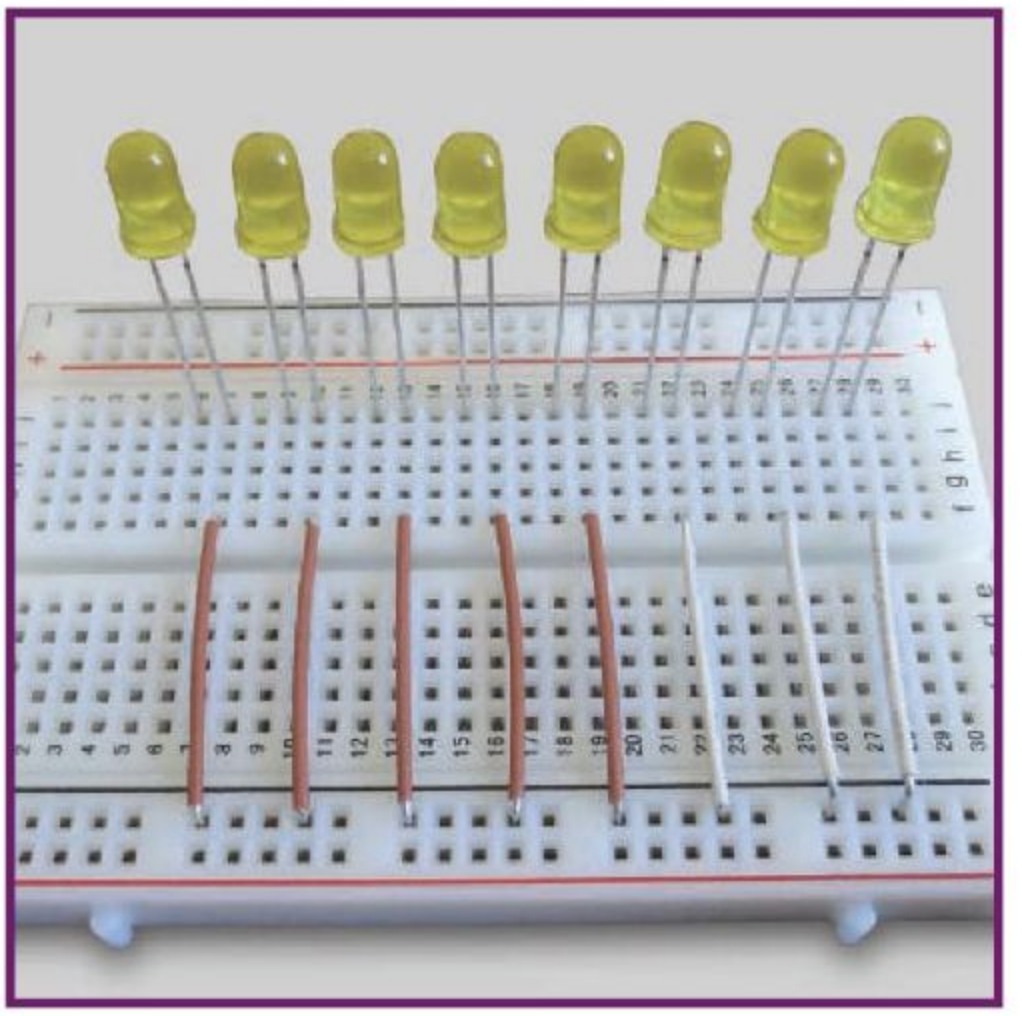
Conectar el circuito

Ahora es necesario armar el circuito, para lo cual necesitaremos la placa Arduino, un protoboard, cables de puente, ocho LEDs y una resistencia, además de seguir las instrucciones que se presentan a continuación.

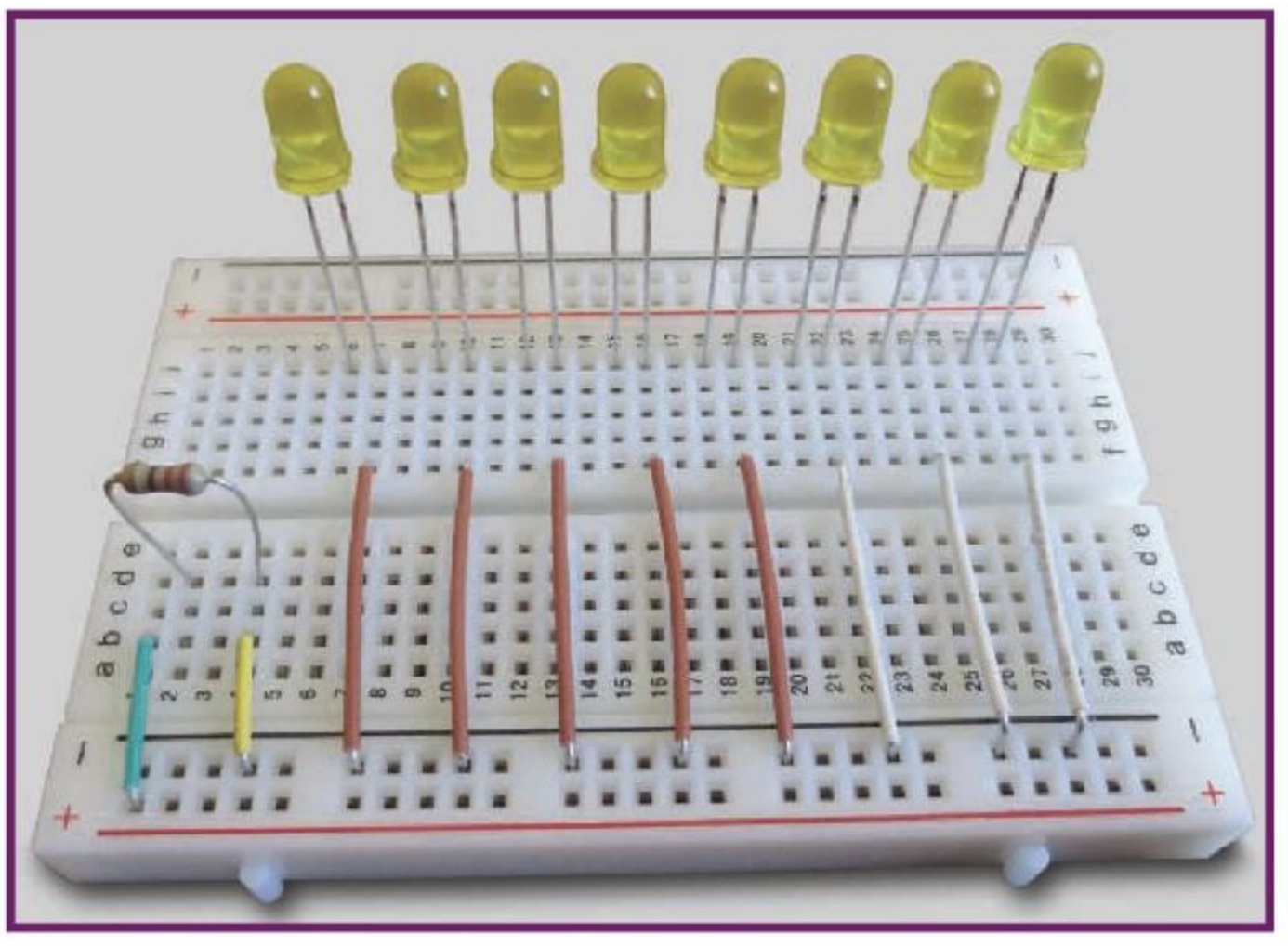
Conectar varios LEDs



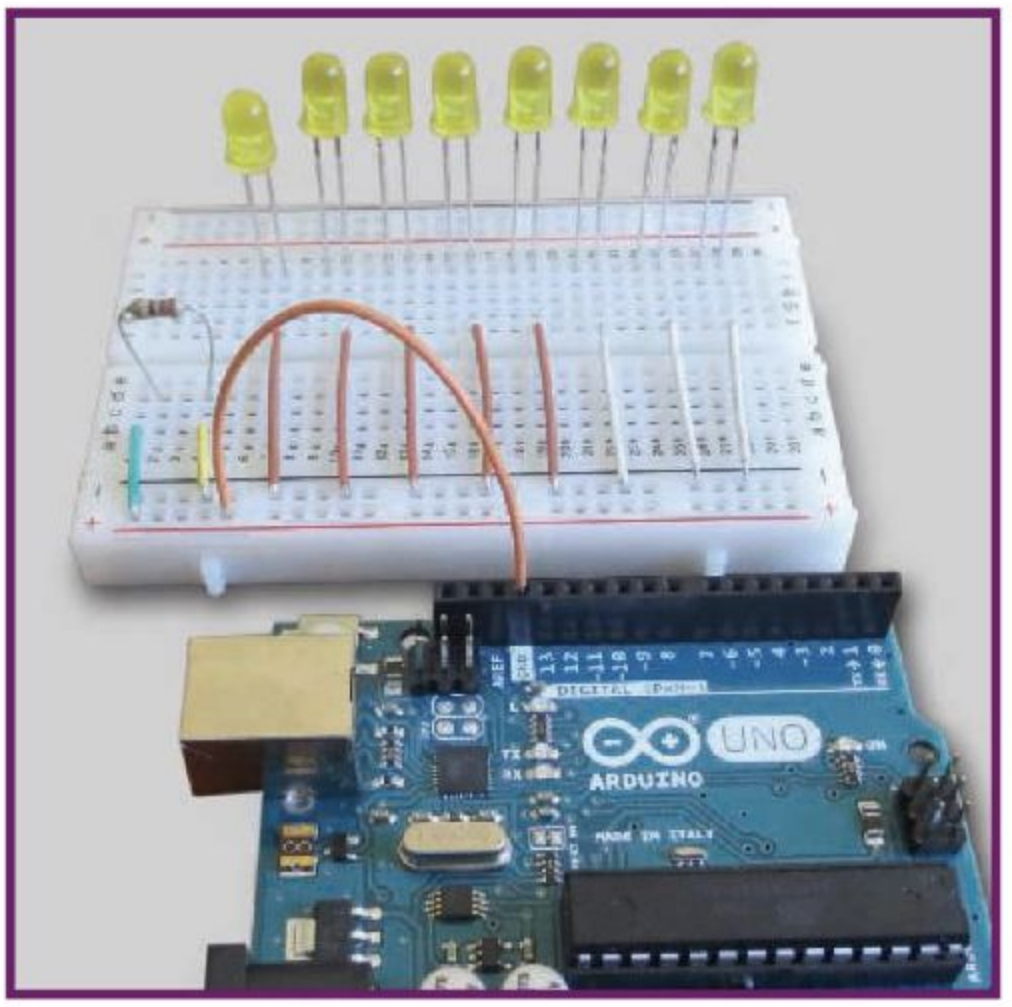
1 Para comenzar, reunimos los materiales necesarios para llevar a cabo este procedimiento: protoboard, cables de puente, LEDs, placa Arduino y cable USB.



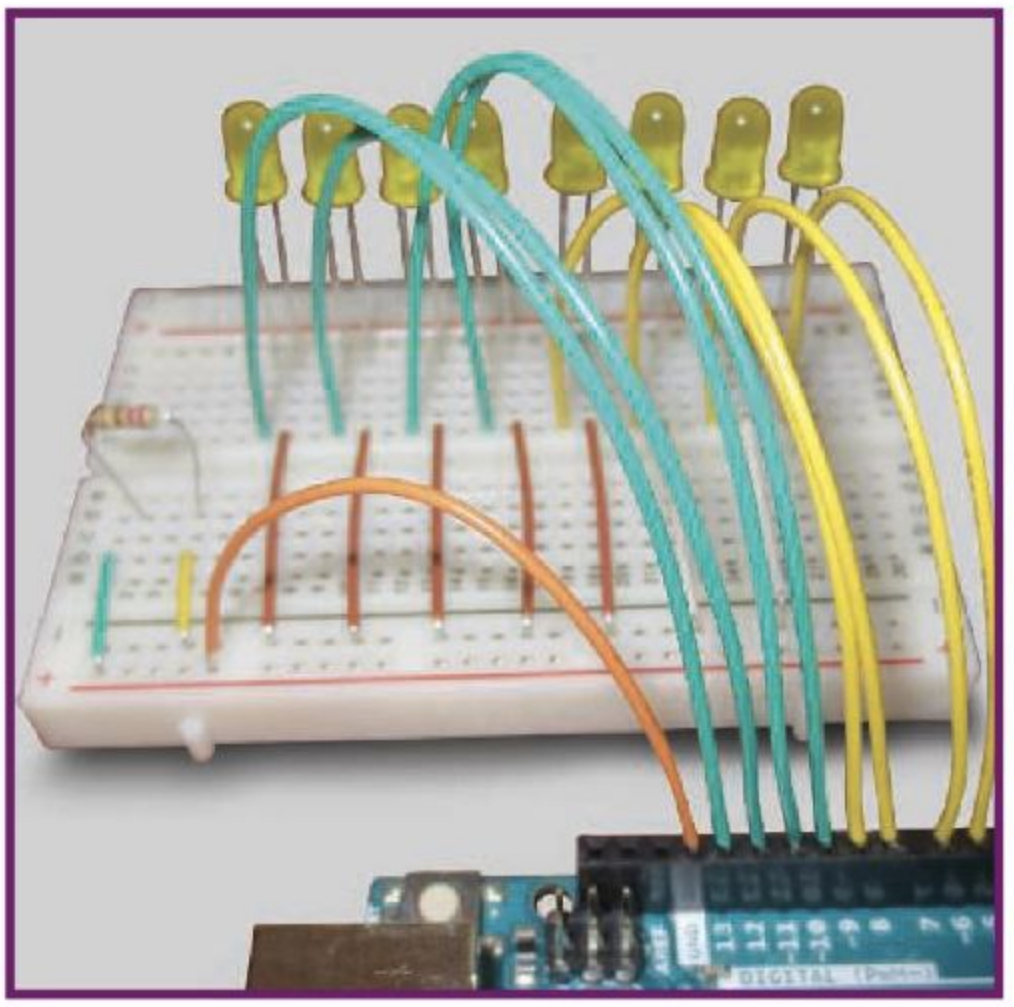
2 Conectamos los LEDs al protoboard, luego utilizamos cables de puente para unir el extremo negativo de cada LED con la línea negativa del protoboard.



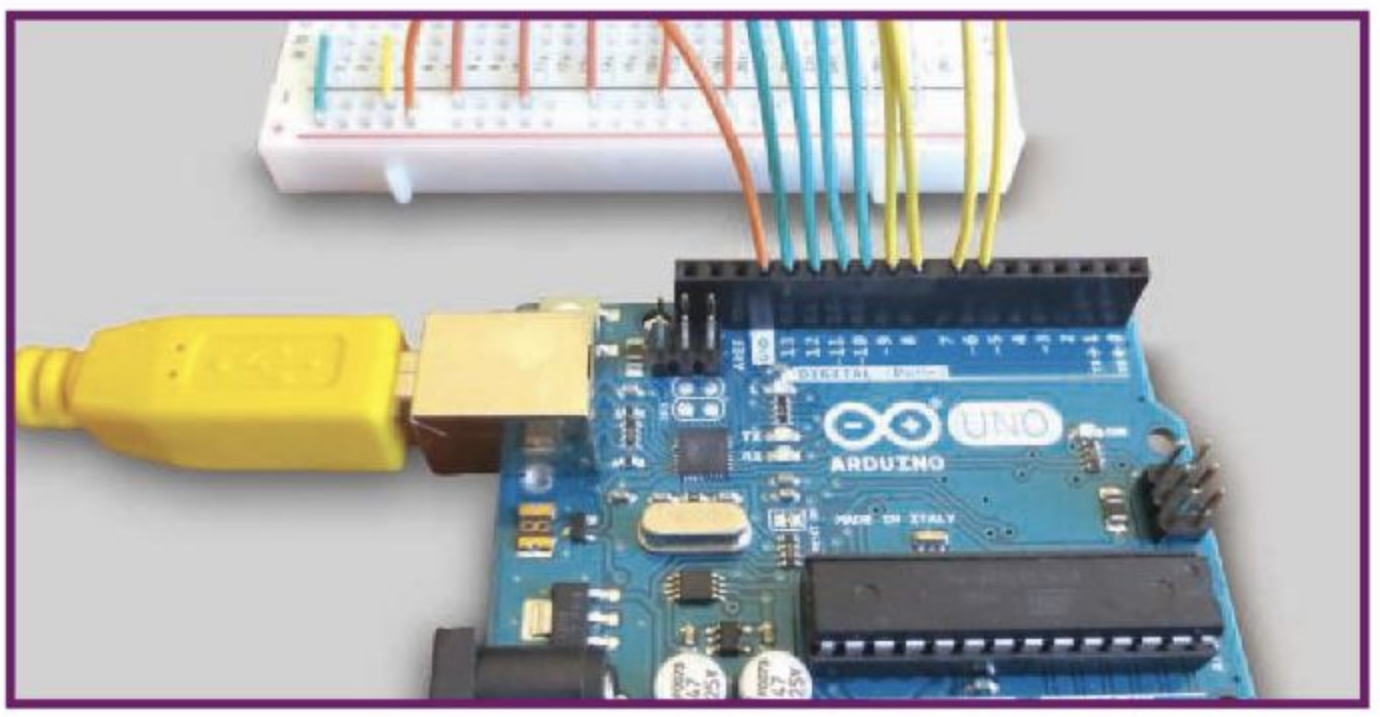
3 Ahora agregamos una resistencia. A continuación, unimos cada uno de sus extremos con las líneas negativa y positiva que se encuentran en uno de los extremos del protoboard.



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



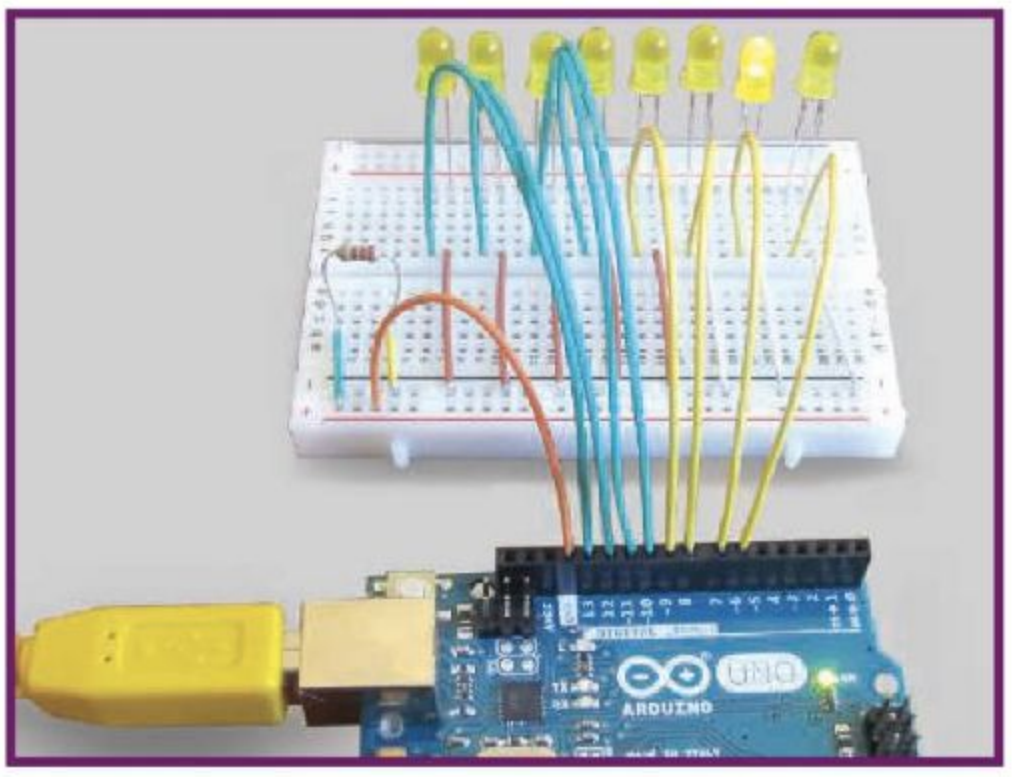
5 Para continuar, utilizamos los cables de puente necesarios para conectar el extremo positivo de cada LED con los pines 6 al 13 de la placa Arduino.



6 Con todas las conexiones ya realizadas, conectamos la placa Arduino a la computadora, para lo cual usamos el cable USB.

```
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
```

7 Subimos el sketch a la placa Arduino; en este punto verificamos que la compilación sea correcta y solucionamos cualquier posible error de sintaxis que pudiera aparecer.



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

```
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   }
```

9 Para lograr otros efectos, podemos tocar el código y subirlo nuevamente a la placa Arduino. Por ejemplo, para que el juego de luces vaya más rápido, reemplazamos las líneas **delay(500)** por **delay(10)**. Cargamos el nuevo sketch y vemos 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 LED 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 se repite la secuencia, pero se irán encendiendo los LEDs que en el primer ciclo no lo hicieron.

Seis LEDs en secuencia

En esta sección 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 desde el centro hasta el exterior; en cada iteración se encienden los LEDs de igual color.

Ubicaremos los LEDs de manera 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 es posible 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 es tiempo de trabajar sobre **loop()**. En esta ocasión, trabajaremos con dos esquemas de código: uno para controlar los primeros tres LEDs y otro para controlar los 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.

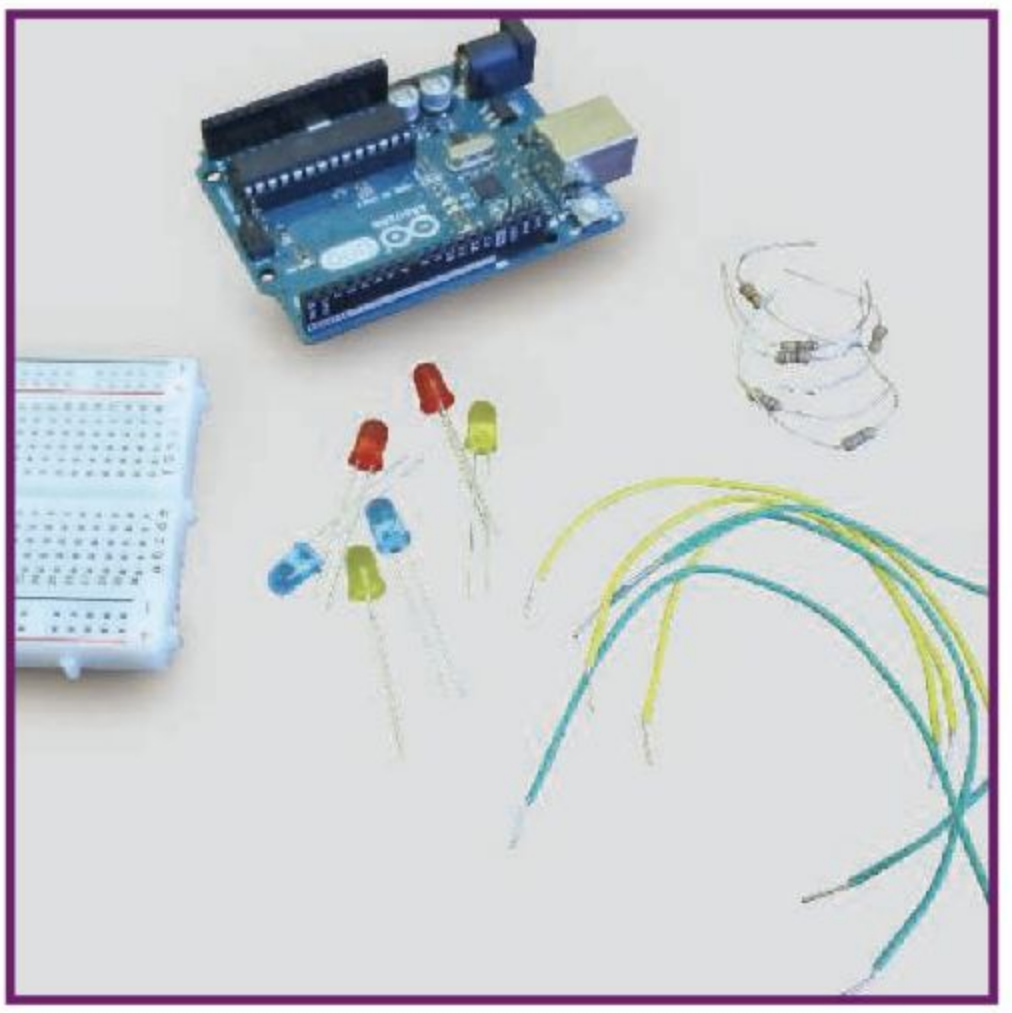
```
        paso = 0;
for (intLed = 6; Led> 3; Led--)
           {
digitalWrite(Led, HIGH);
digitalWrite(Led + 2 * paso + 1,
HIGH);
delay(200);
digitalWrite(Led, LOW);
digitalWrite(Led + 2 * paso + 1, LOW);
delay(200);
        paso++;
           }
```

Con esto, nuestro código quedará de la siguiente forma:

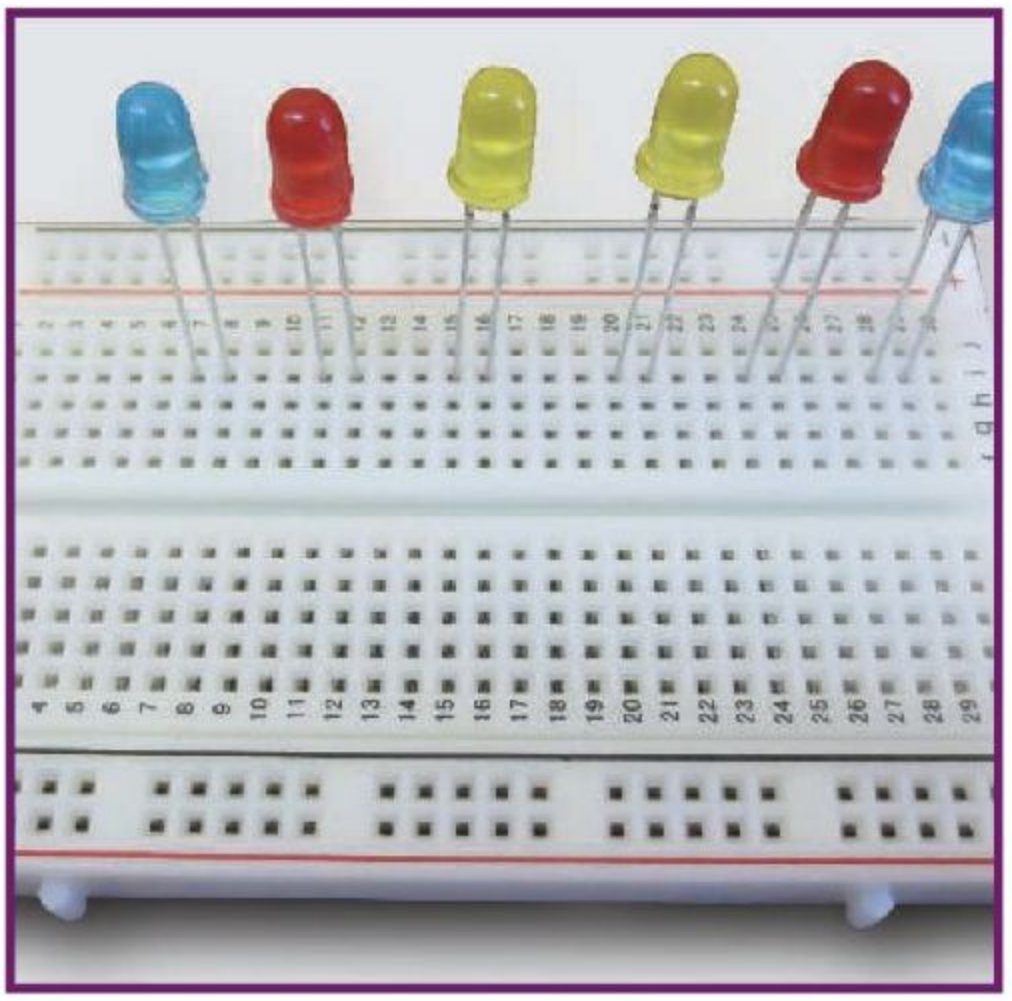
```
voidsetup()
     {
intLed = 0 ;
for ( Led = 4 ; Led< 10 ; Led++)
pinMode( Led , OUTPUT) ;
     }

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.

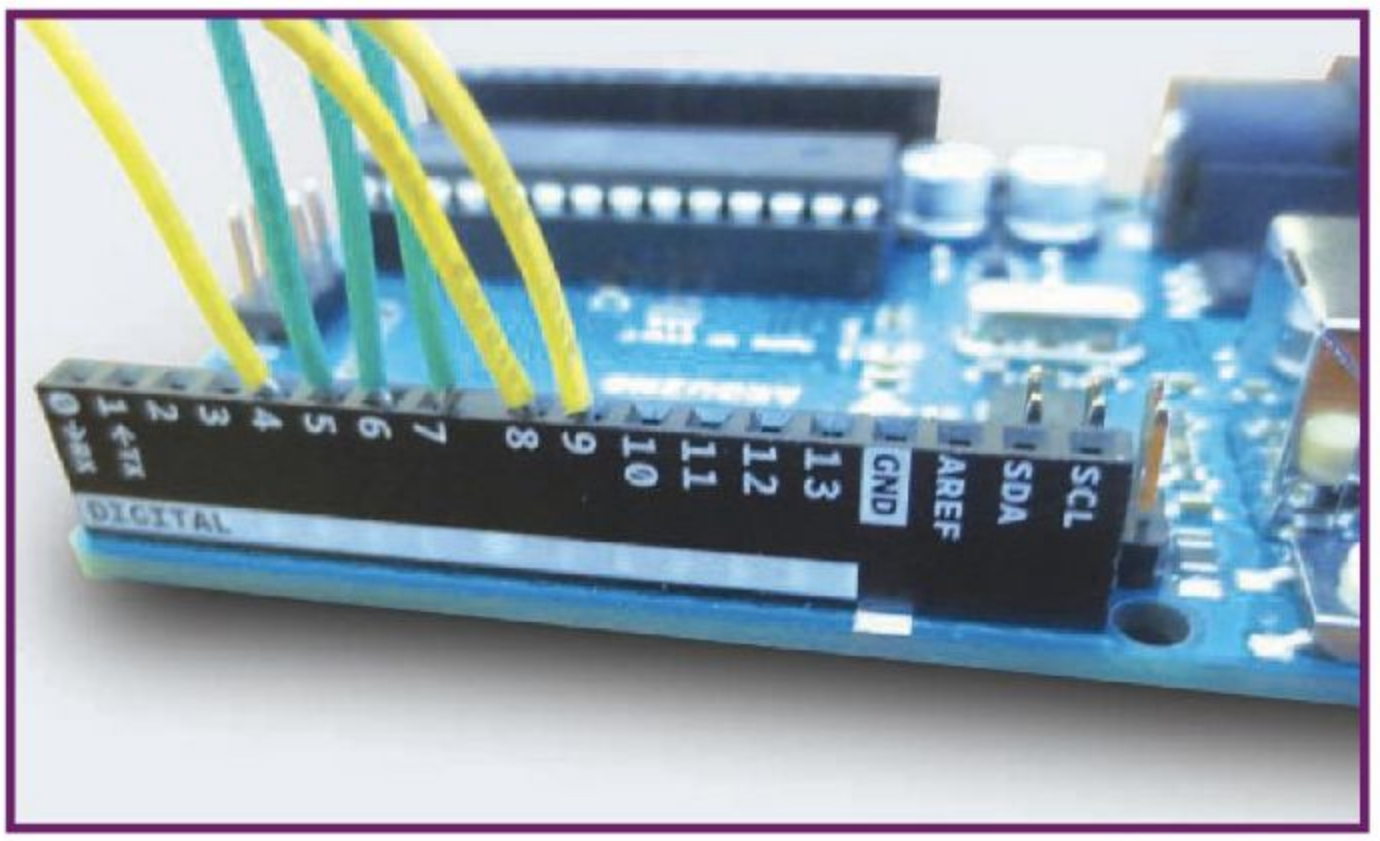
Circuito para encender 6 LEDs



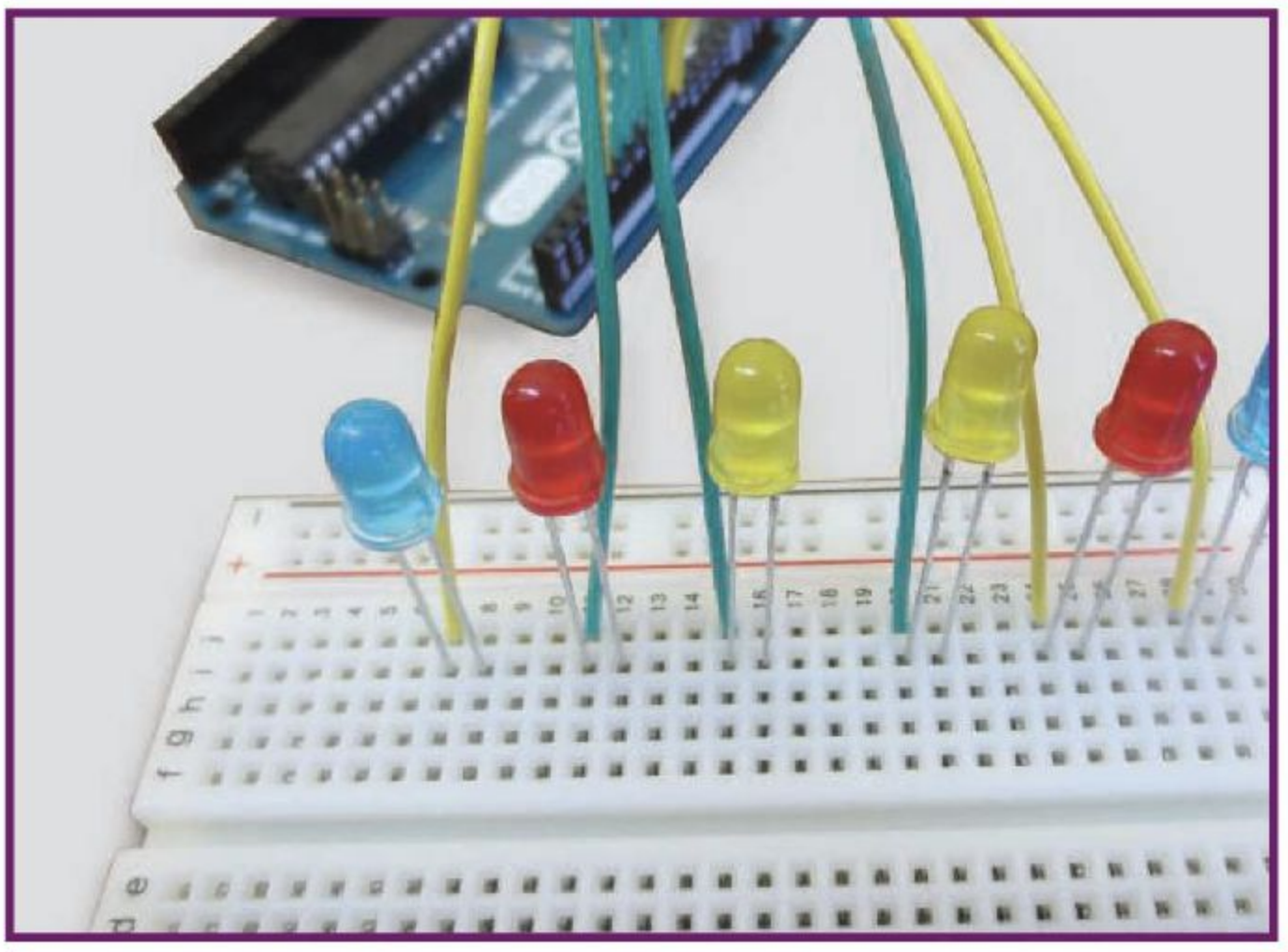
1 Para comenzar, reunimos los elementos necesarios para armar este circuito. En este caso se trata de 6 LEDs en tres colores diferentes, 6 resistencias, cables de puente, Arduino UNO y protoboard.



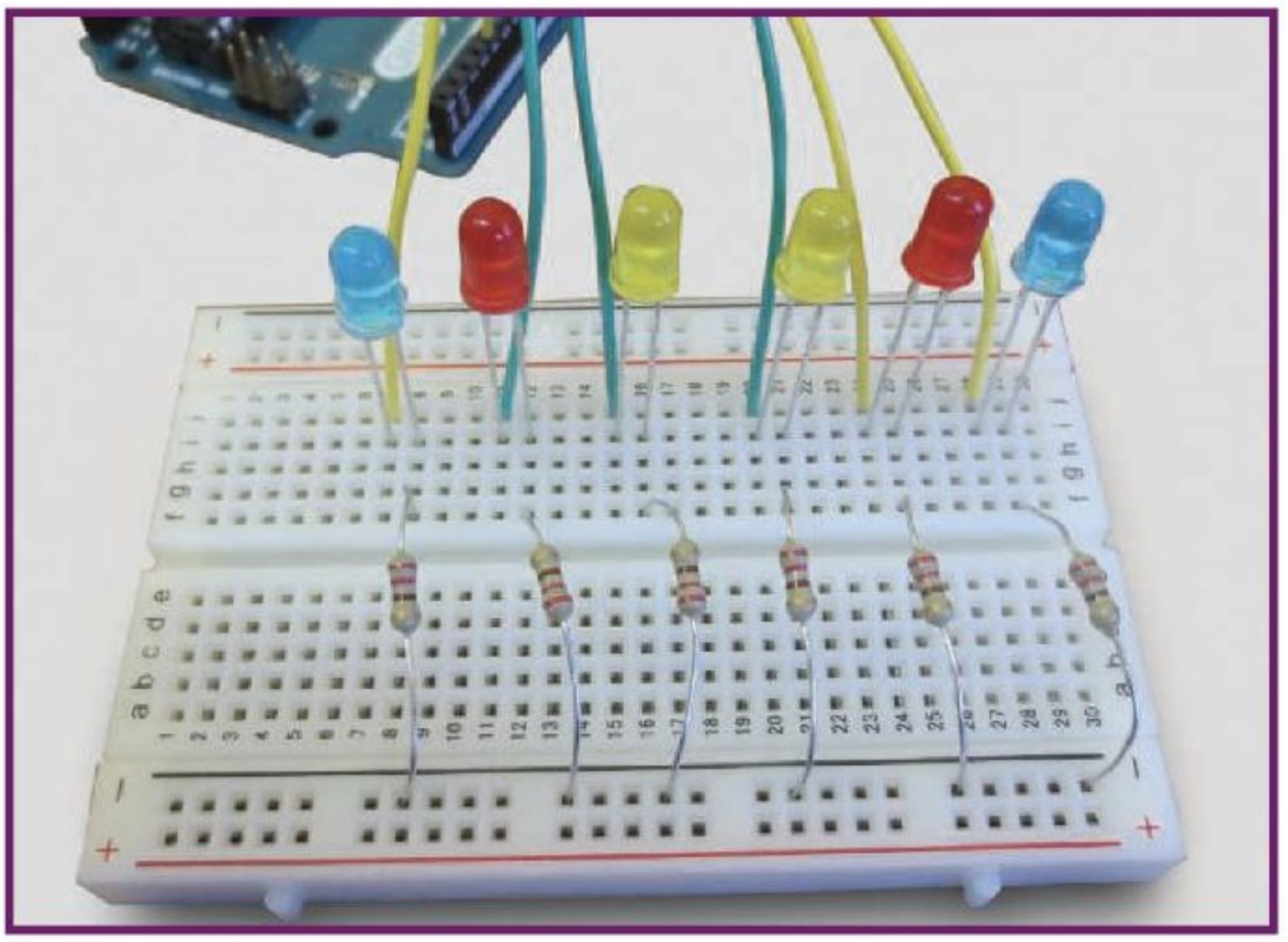
2 Conectamos los LEDs en el protoboard, deberán quedar conectados en línea en el siguiente orden: azul, rojo, amarillo, amarillo, rojo, azul. Por comodidad, podemos dejar algunos espacios entre cada LED.



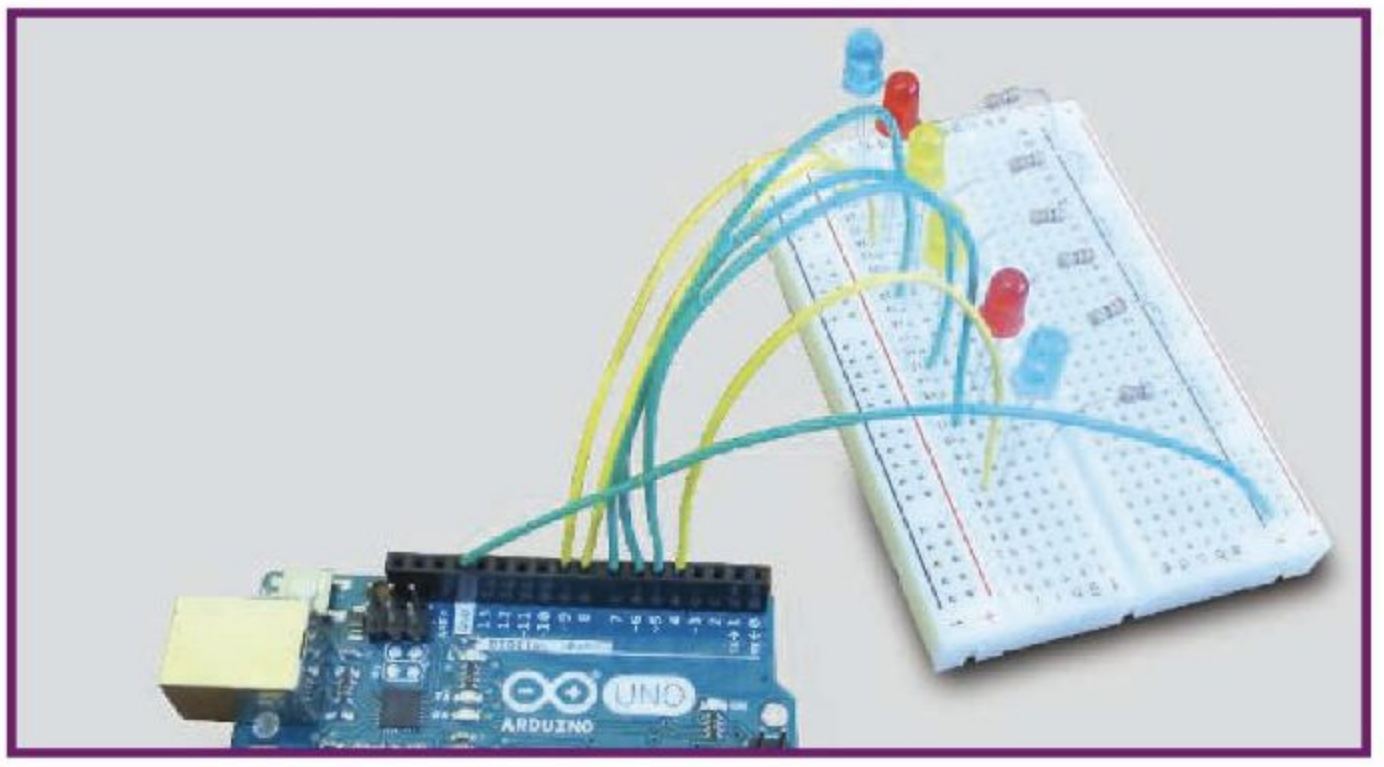
3 Como se declaró el uso de los pines 4 al 9, conectamos cables de puente a los pines adecuados en la tarjeta Arduino UNO.



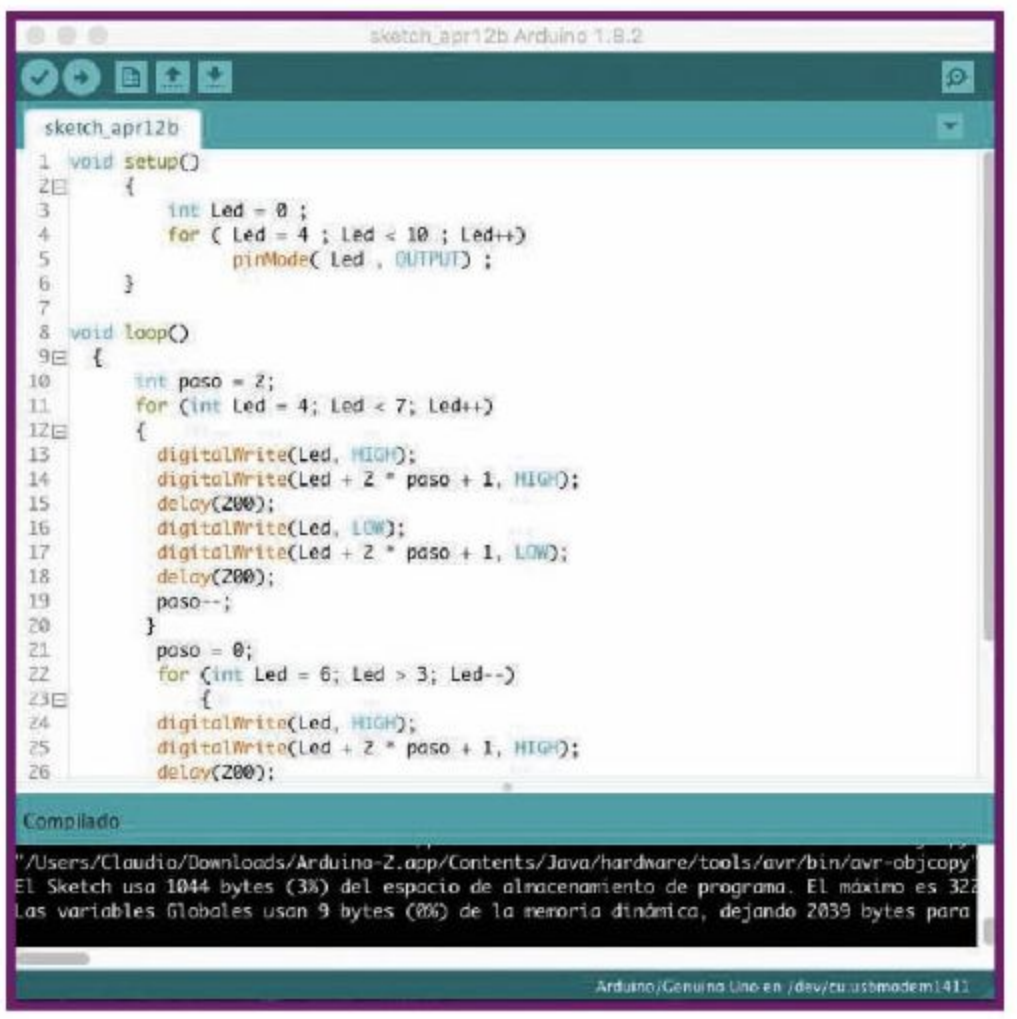
4 Para continuar, conectamos los cables de puente a los LEDs que ubicamos en el protoboard. Comenzamos por el pin 4, que irá conectado al primer LED azul, el pin 5 que irá conectado al primer LED rojo y el pin 6, que irá conectado al primer LED amarillo. Posteriormente, los pines 7, 8 y 9 deberán conectarse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



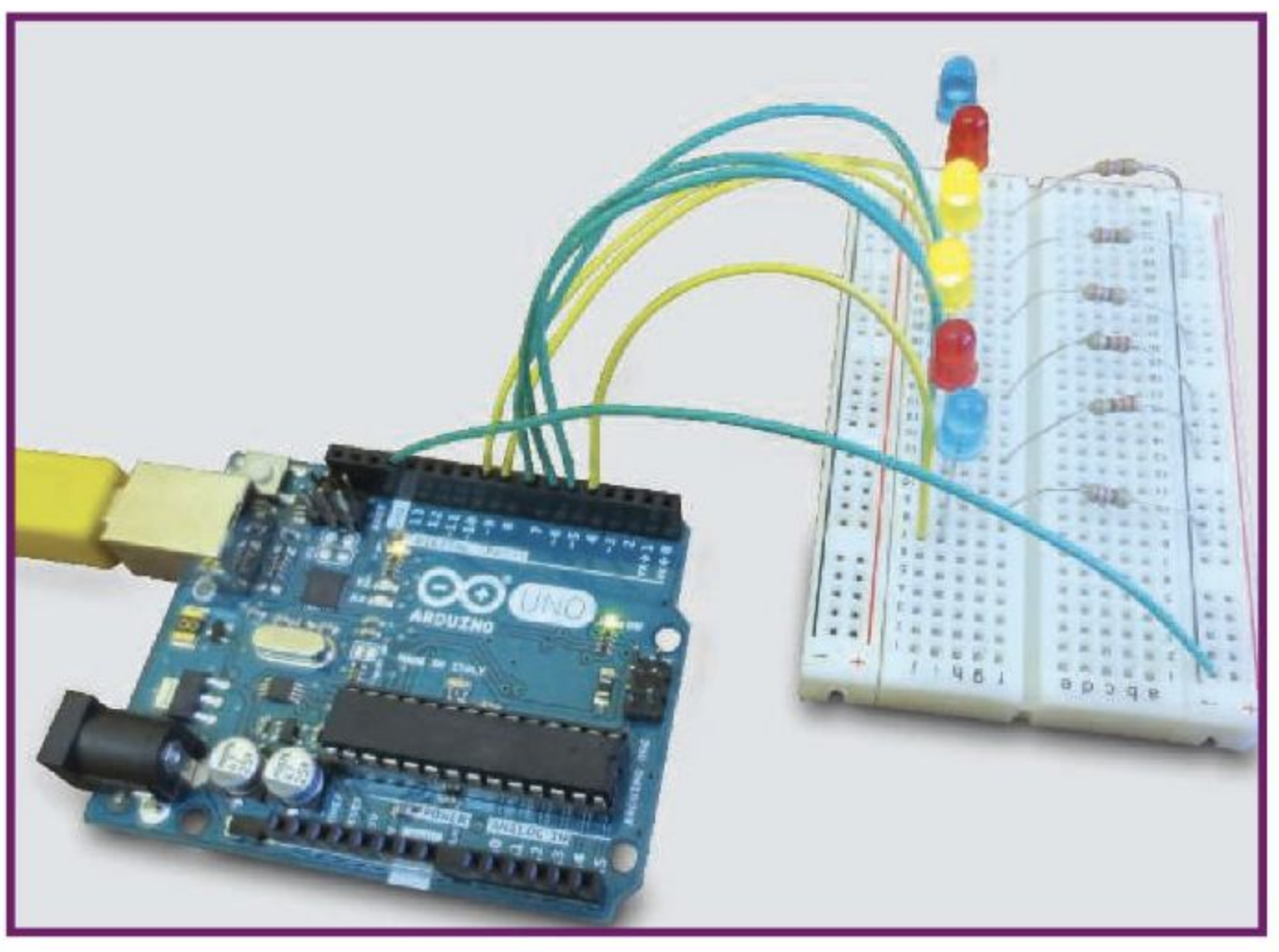
5 Conectamos las resistencias para unir la línea negativa de la parte inferior del protoboard con los LEDs. Utilizamos una resistencia para cada LED.



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



7 Con las conexiones ya realizadas, debemos compilar el sketch que desarrollamos antes. Verificamos que la compilación se realice sin problemas; de lo contrario, buscamos y solucionamos los posibles errores que puedan presentarse.



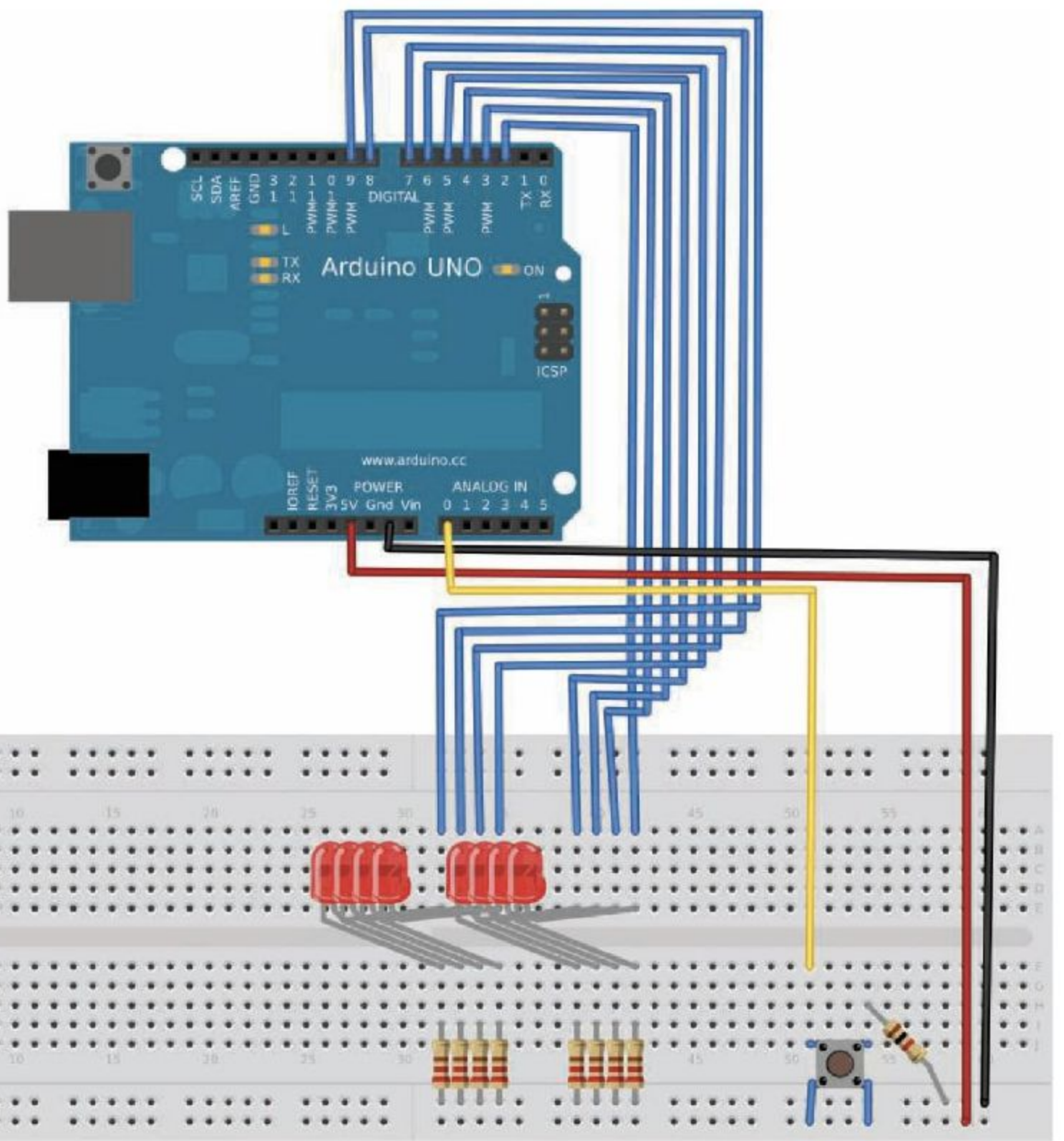
8 Conectamos la placa Arduino UNO a la computadora mediante el cable USB y cargamos el sketch mediante la opción **Subir**, que se encuentra en la barra superior de opciones. Con el sketch cargado, veremos que los LEDs se encenderán en secuencia: los dos azules, luego los dos rojos y finalmente los dos amarillos.

Secuencia de 8 LEDs

Para complejizar un poco lo que venimos haciendo con Arduino, trabajaremos con un grupo de 8 LEDs, pero en esta ocasión los 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, 8 LEDs, 8 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.

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();
}

}
```

Con una resistencia de 150Ω estamos dentro del margen recomendado por el Forward Voltage de la hoja técnica de un LED. Si queremos que nuestro LED brille más, podemos cambiar el valor de esta resistencia, por ejemplo llegar hasta 3 V, que es lo máximo recomendado.

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.

Lograremos diferentes variaciones teniendo como base la creación de distintas secuencias de encendido, a las que accederemos mediante un pulsador o en forma automática.

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.

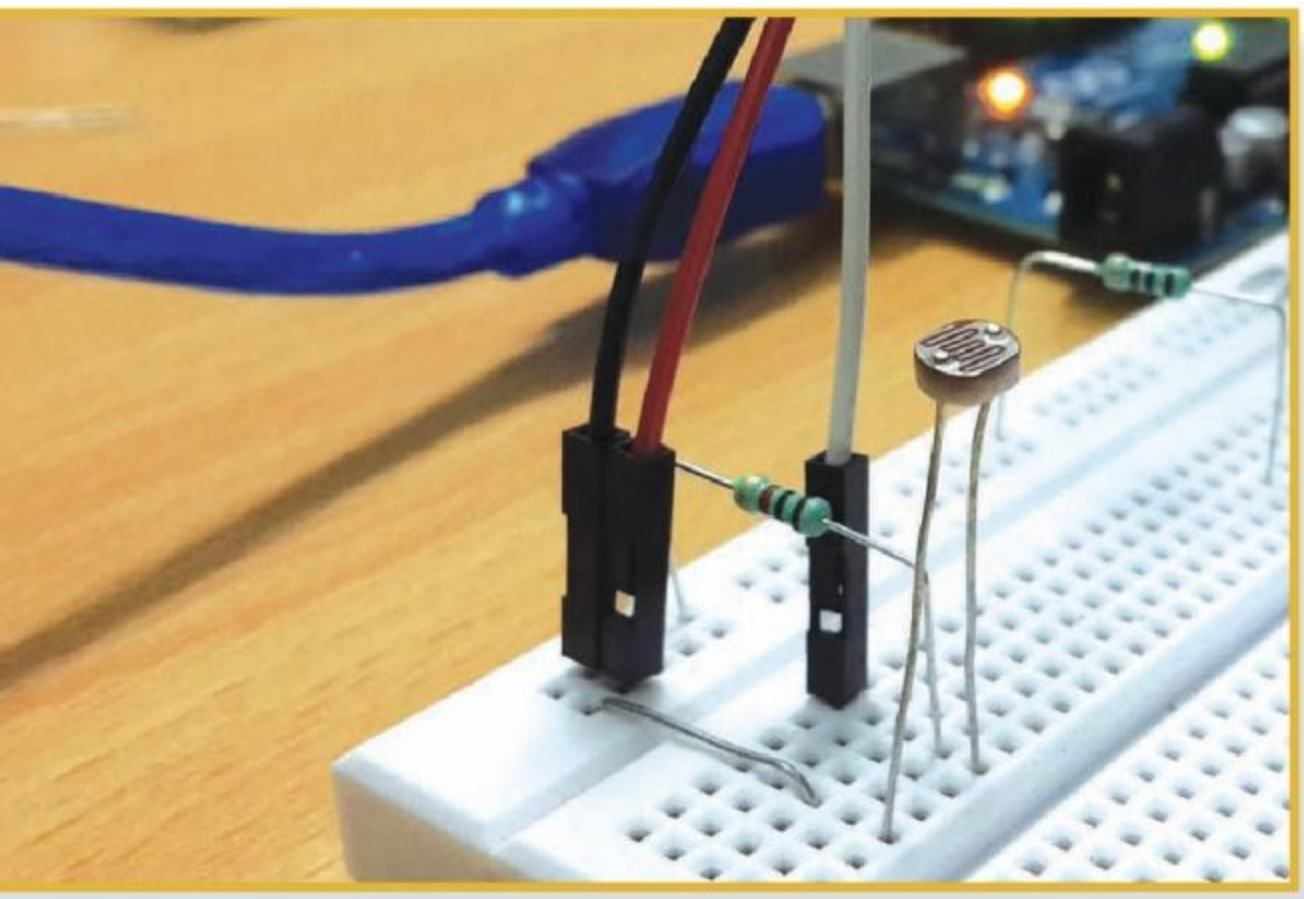
¿Qué es una fotorresistencia?

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.

Ya tenemos los conocimientos básicos que nos permitirán enfrentar un proyecto algo más complejo. Sabemos crear nuestros primeros sketches 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 sobre los componentes que pueden ser utilizados, podemos realizar un proyecto como el que explicamos a continuación.

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 o exterior.



DATOS CORRESPONDIENTES A LAS FOTORRESISTENCIAS DE LA FAMILIA GL55

MODELO	VOLTAJE (V)
GL5516	150
GL5528	150
GL5537-1	150
GL5537-2	150
GL5539	150
GL5549	150

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, que 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. Su valor típico es de **0,5** a **0,8**.
Si deseamos 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 presentamos los datos en la tabla al pie de página.

En todo caso, debido a las variaciones propias del proceso de construcción de estos componentes, no podemos utilizar esta información de manera absoluta. La recomendación es efectuar un proceso de calibración preliminar; así, enfrentaremos la posibilidad de que pequeñas diferencias entre componentes den como resultado grandes variaciones en el momento de efectuar la medición.

Variaciones

Existen muchas fotorresistencias diferentes; sus variaciones dependen no solo 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 únicamente serán sensibles a las radiaciones infrarrojas.

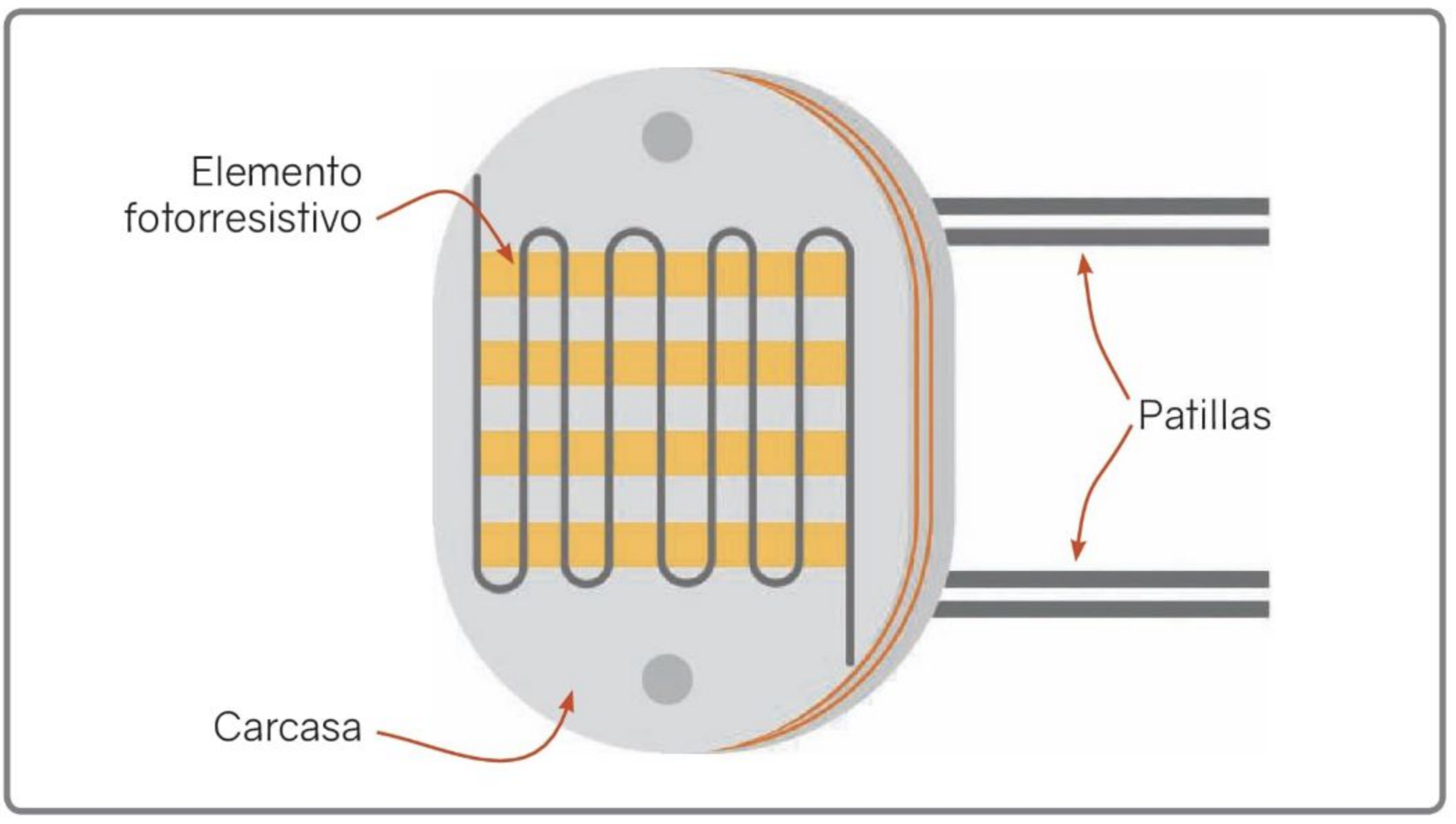
	TEMPERATURA (°C)	PICO ESPECTRAL (NM)	RESISTENCIA OSCURIDAD (KΩ)	RESISTENCIA LUZ BRILLANTE (KΩ)	GAMMA	TIEMPO RESPUESTA (MS)
	-30°+70°	540	5-10	500	0,5	30
	-30°+70°	540	10-20	1000	0,6	25
	-30°+70°	540	20-30	2000	0,6	25
	-30°+70°	540	30-50	3000	0,7	25
	-30°+70°	540	50-100	5000	0,8	25
	-30°+70°	540	100-200	10000	0,9	25

¿Cómo aprovechar una fotorresistencia?

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, razón por la cual lo utilizaremos en conjunto con las entradas analógicas de la placa Arduino, para obtener una estimación del nivel del 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 pasarán a la banda de conducción, lo que disminuirá la resistencia que presenta el componente o fotorresistor.
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.

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 las principales partes de una fotorresistencia.



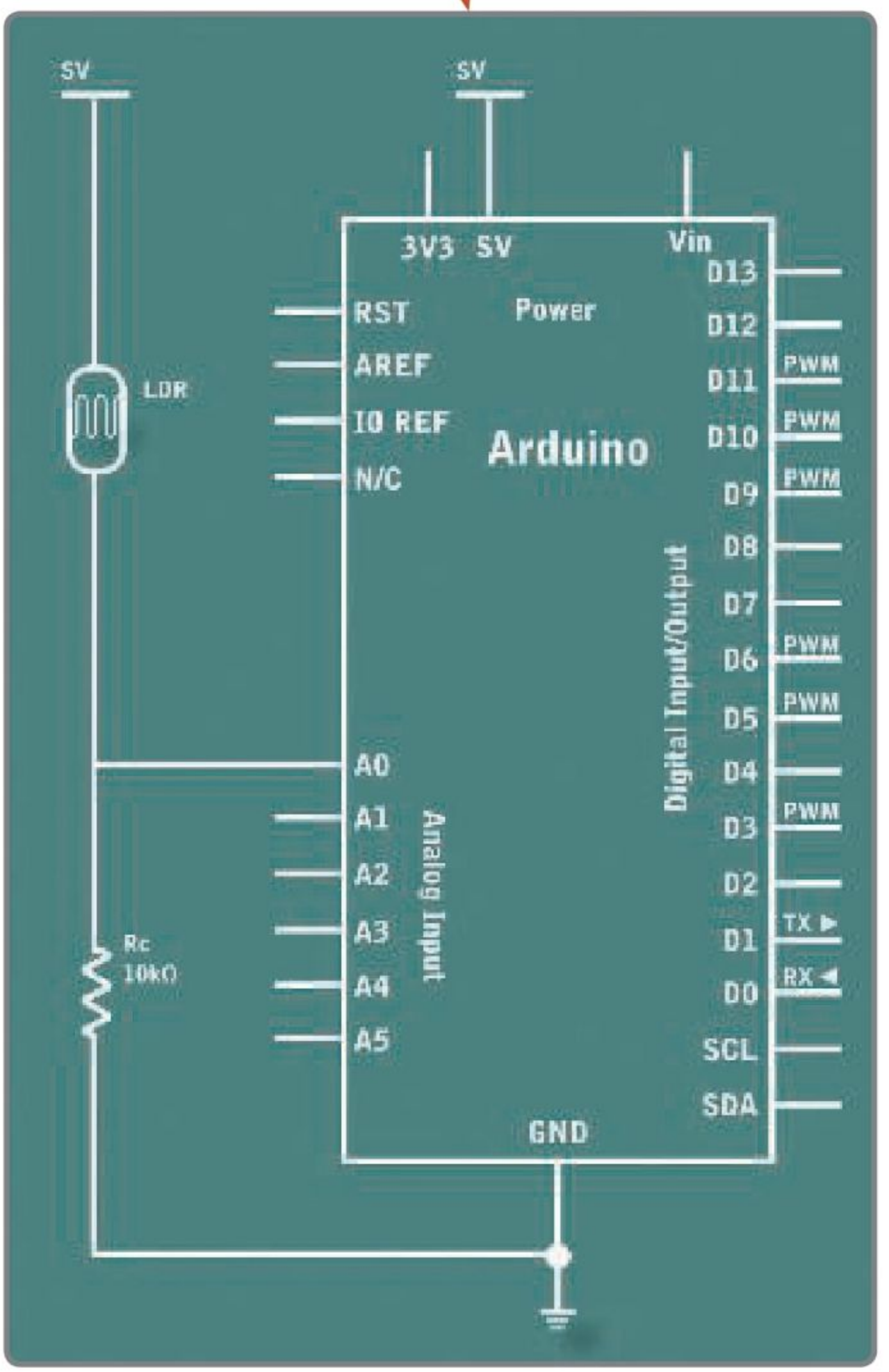
Debemos tener en cuenta que la variación de la resistencia en un fotorresistor es bastante lenta; por este motivo, no es un elemento eficiente cuando deseamos medir variaciones de luz rápidas, por ejemplo, las que podría producir una fuente de luz artificial que se encuentra alimentada por corriente 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 luz rápida; en cambio, si deseamos medir la intensidad de la luz ambiental, tanto en interiores como en exteriores, seguramente 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. Para lograrlo, conectaremos este sensor a una entrada analógica de la placa Arduino mediante un protoboard. Luego, podremos jugar con el código para, por ejemplo, hacer reaccionar un LED cuando el nivel de luz exceda un límite establecido o para acceder a información sobre la lectura del nivel de iluminación que es detectado por el sensor LDR.

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.



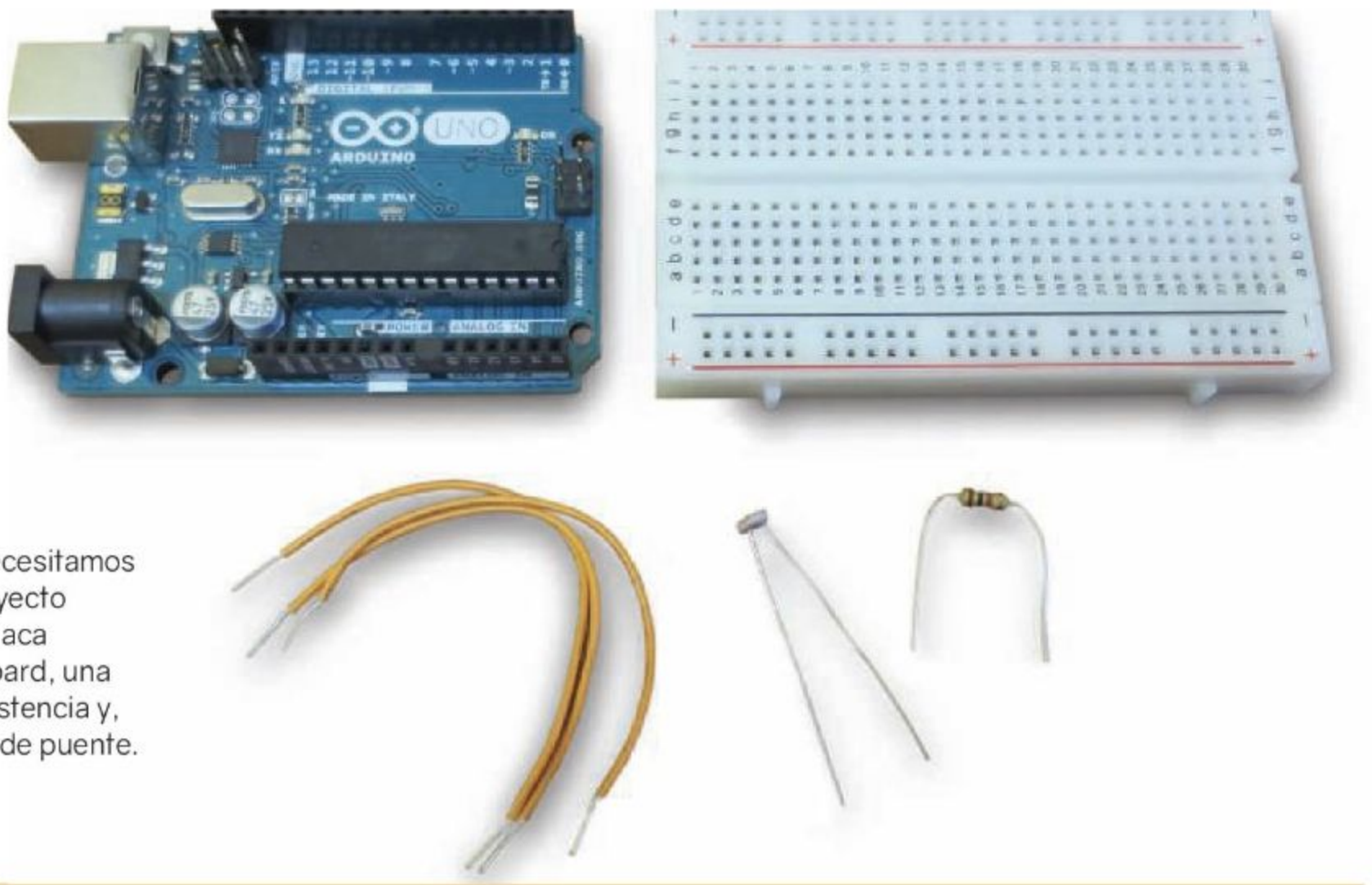
Componentes necesarios

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. Este será el componente principal, pero además necesitaremos otros, que vamos a definir a continuación.

Como siempre, contaremos con una placa Arduino. Podemos elegir entre las opciones disponibles, pero siguiendo el trabajo que hemos realizado hasta aquí, 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 lo que también utilizaremos un protoboard. Este elemento nos permitirá conectar en forma precisa todos los componentes involucrados 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.



Los componentes que necesitamos para desarrollar este proyecto son los siguientes: una placa Arduino UNO, un protoboard, una fotorresistencia, una resistencia y, también, algunos cables de puente.

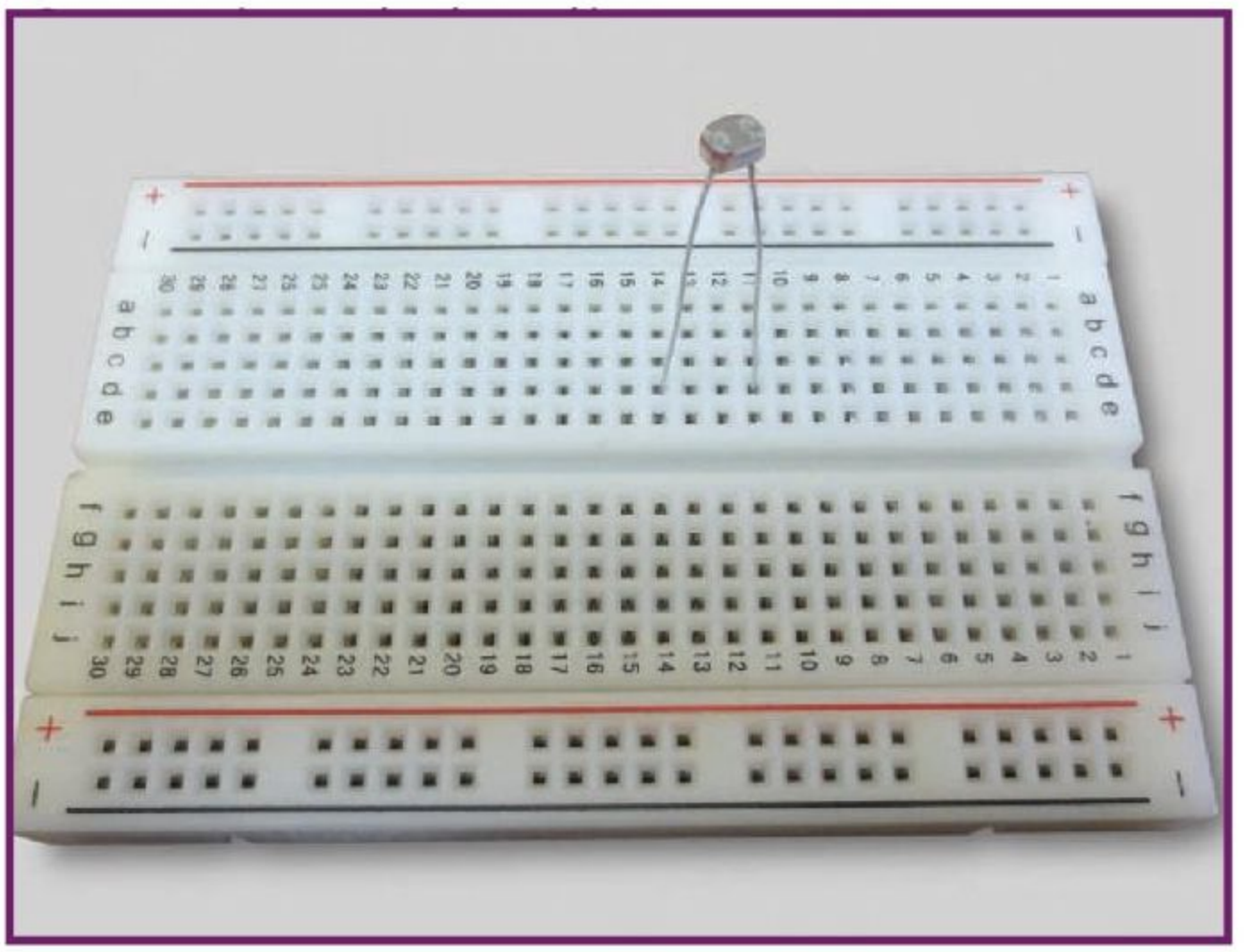
Resultados esperados

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.

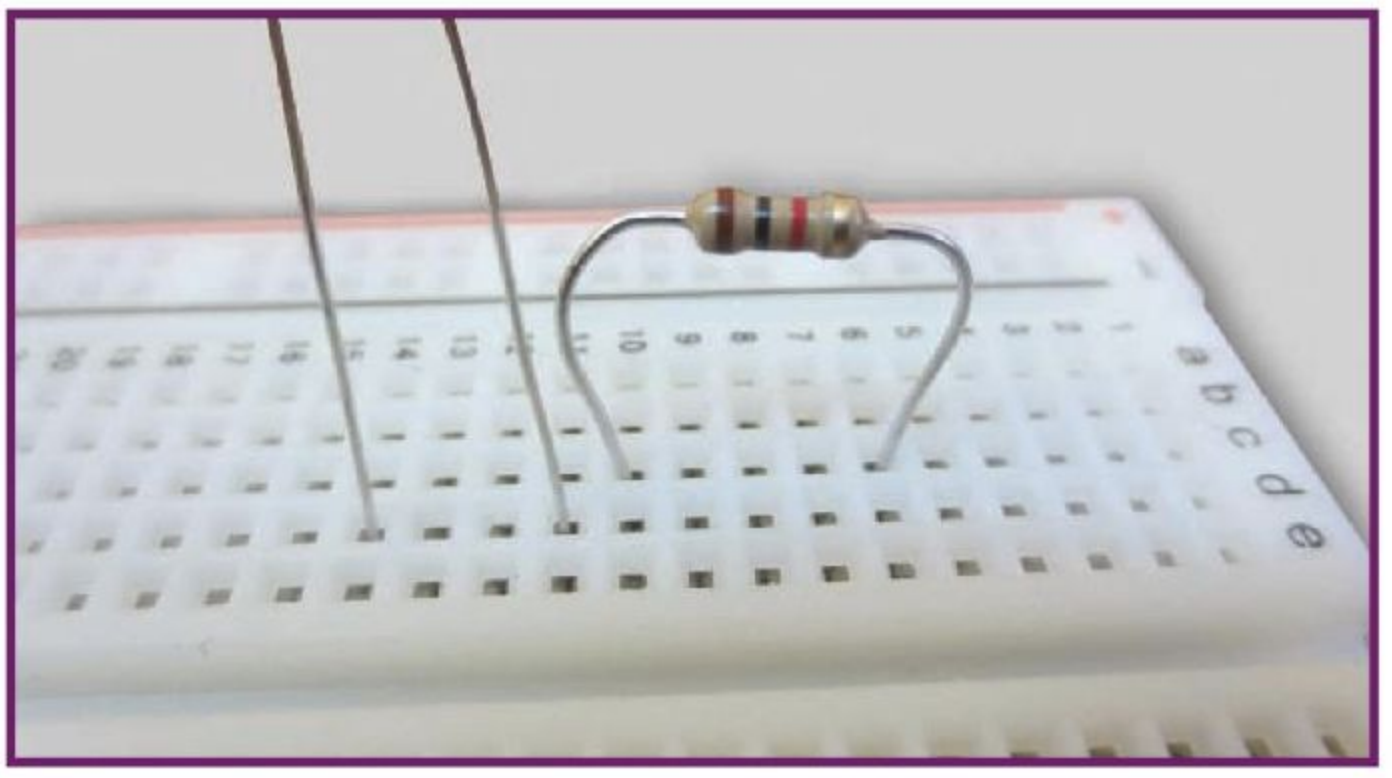
1. Crear un sketch que nos permita activar un LED solo cuando la intensidad de la luz llegue a un nivel que definiremos con anticipación.
2. Implementar el código necesario para hacer parpadear un LED mientras el sensor LDR conectado reciba la cantidad suficiente de luz.
3. Implementar el sketch adecuado 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 detectada por el sensor LDR.

Armar el 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 a continuación.

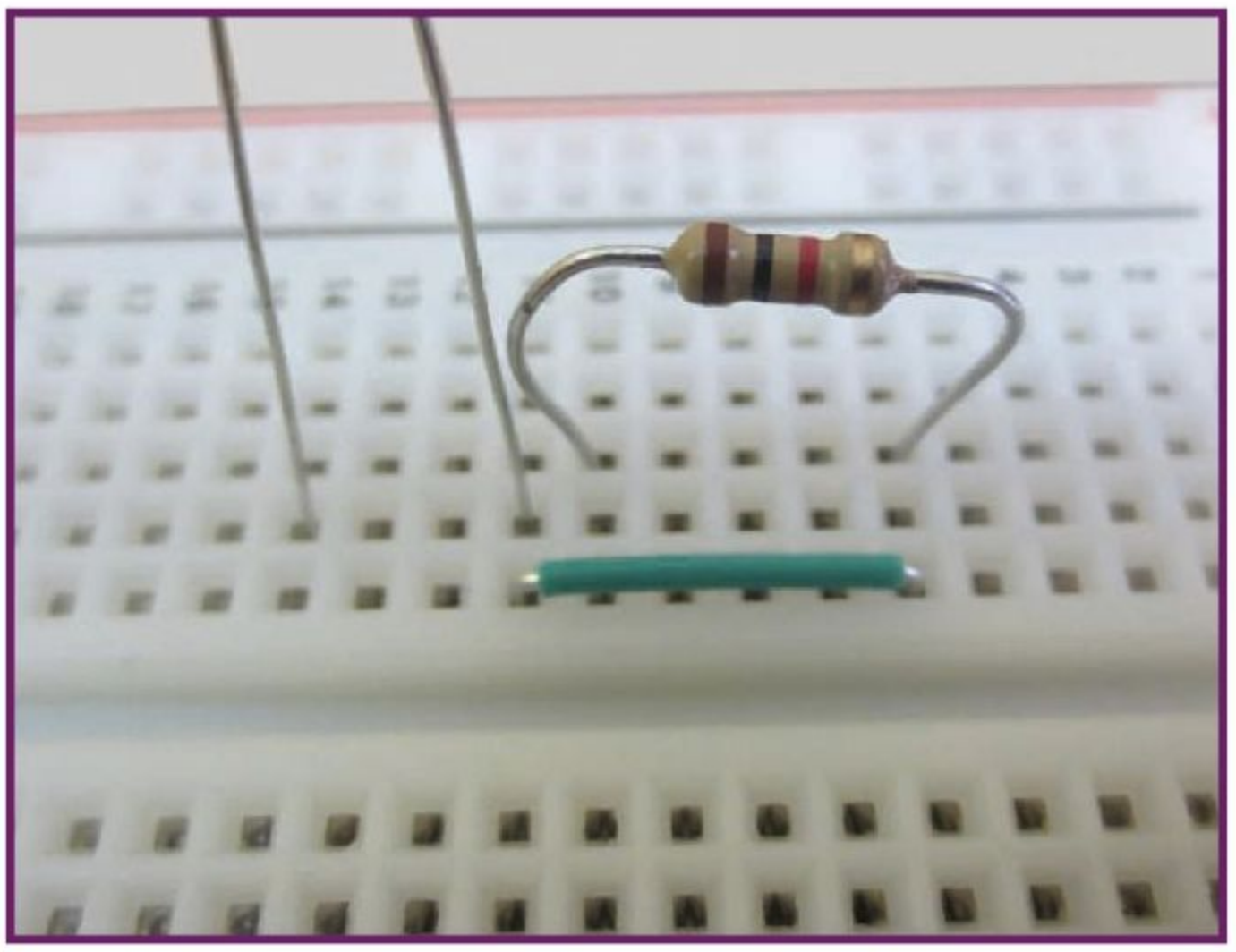


1 Para comenzar, conectamos la fotorresistencia a la placa Arduino UNO; debemos hacerlo en un espacio central, que nos permita trabajar con comodidad con los demás elementos que es necesario incluir en el circuito.

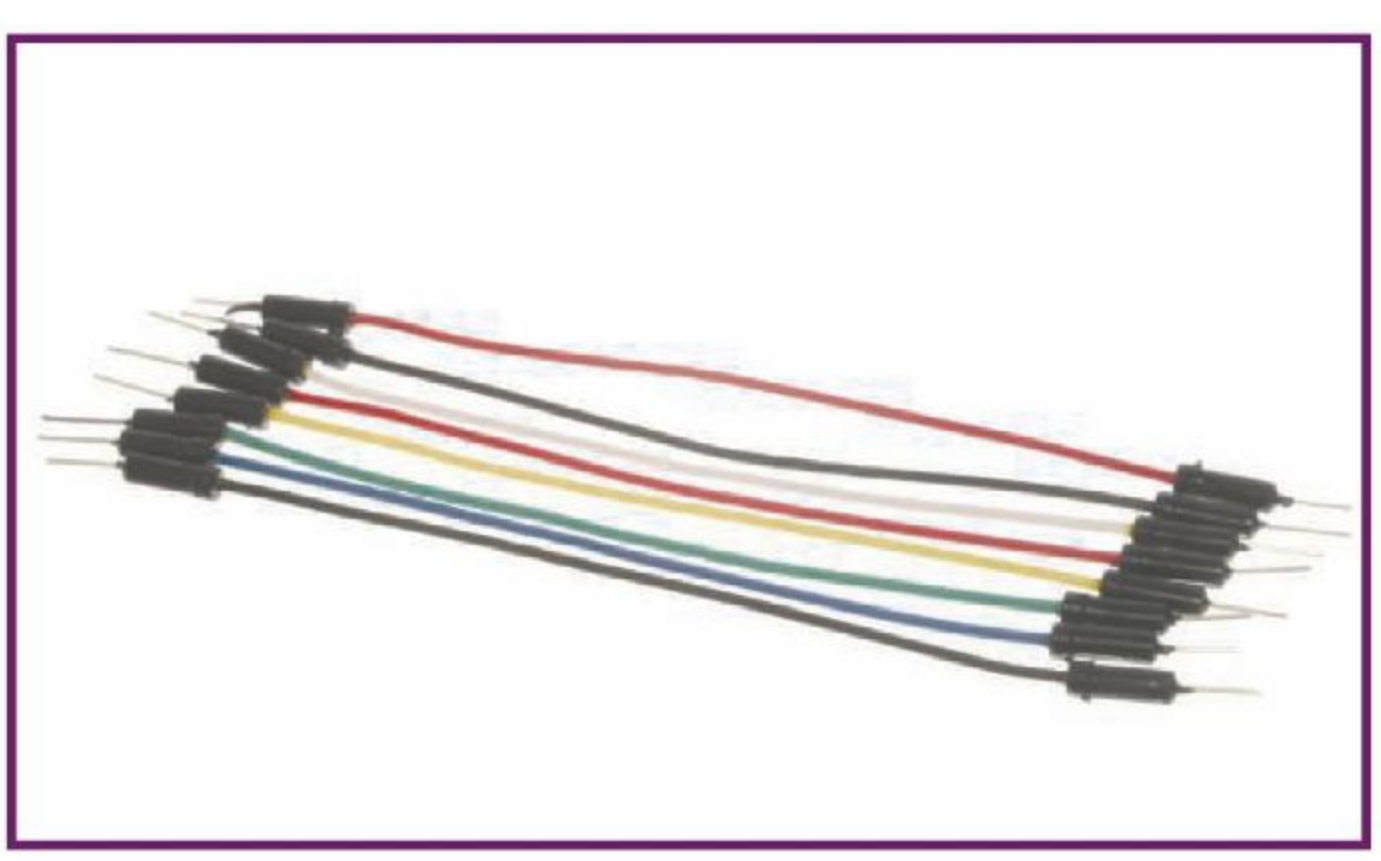


2 Agregamos una resistencia al circuito, conectándola justo en la línea siguiente a la fotorresistencia; esto es importante pues después las uniremos mediante un cable de puente.

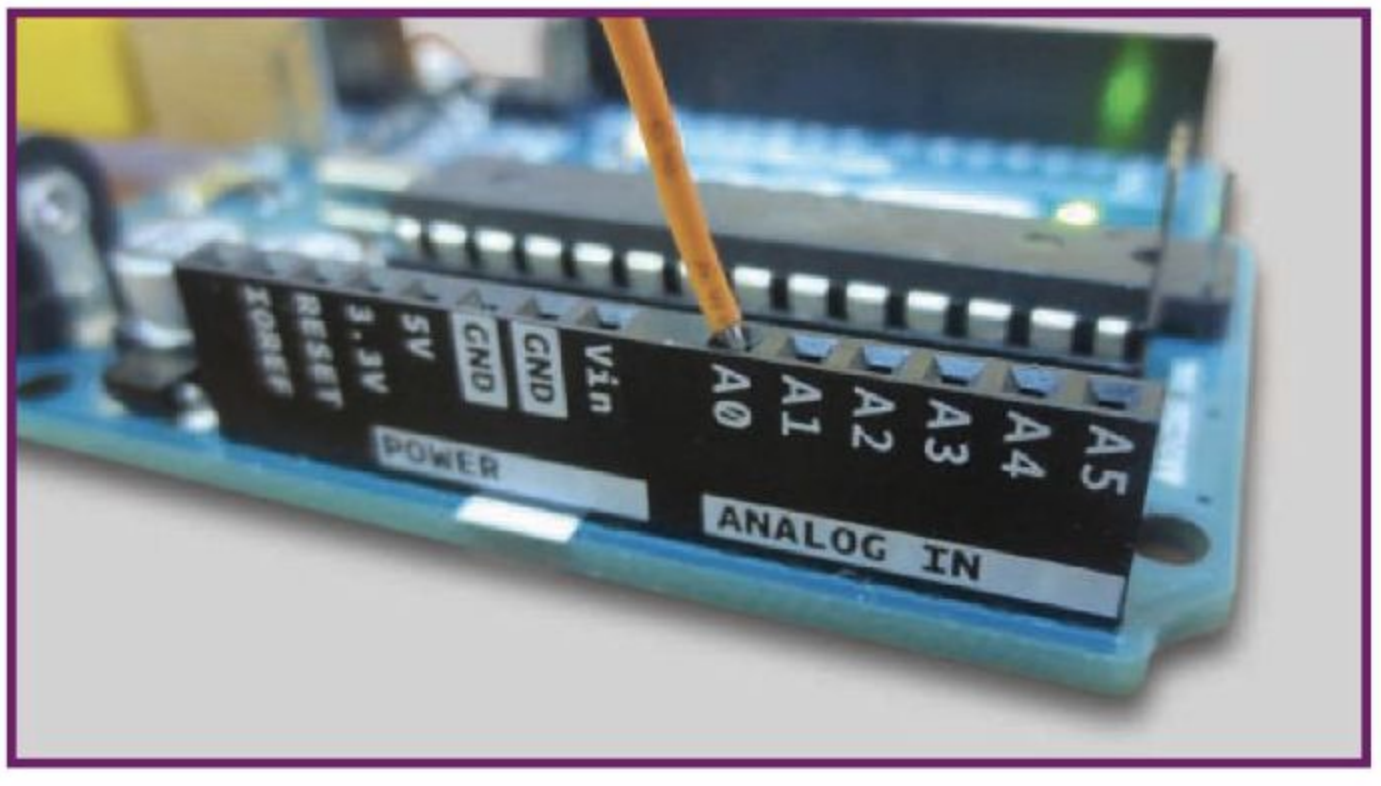
La fotorresistencia LDR es un componente cuya resistencia varía sensiblemente con la cantidad de luz percibida, pero la relación entre la intensidad lumínica y el valor de la resistencia no es lineal.



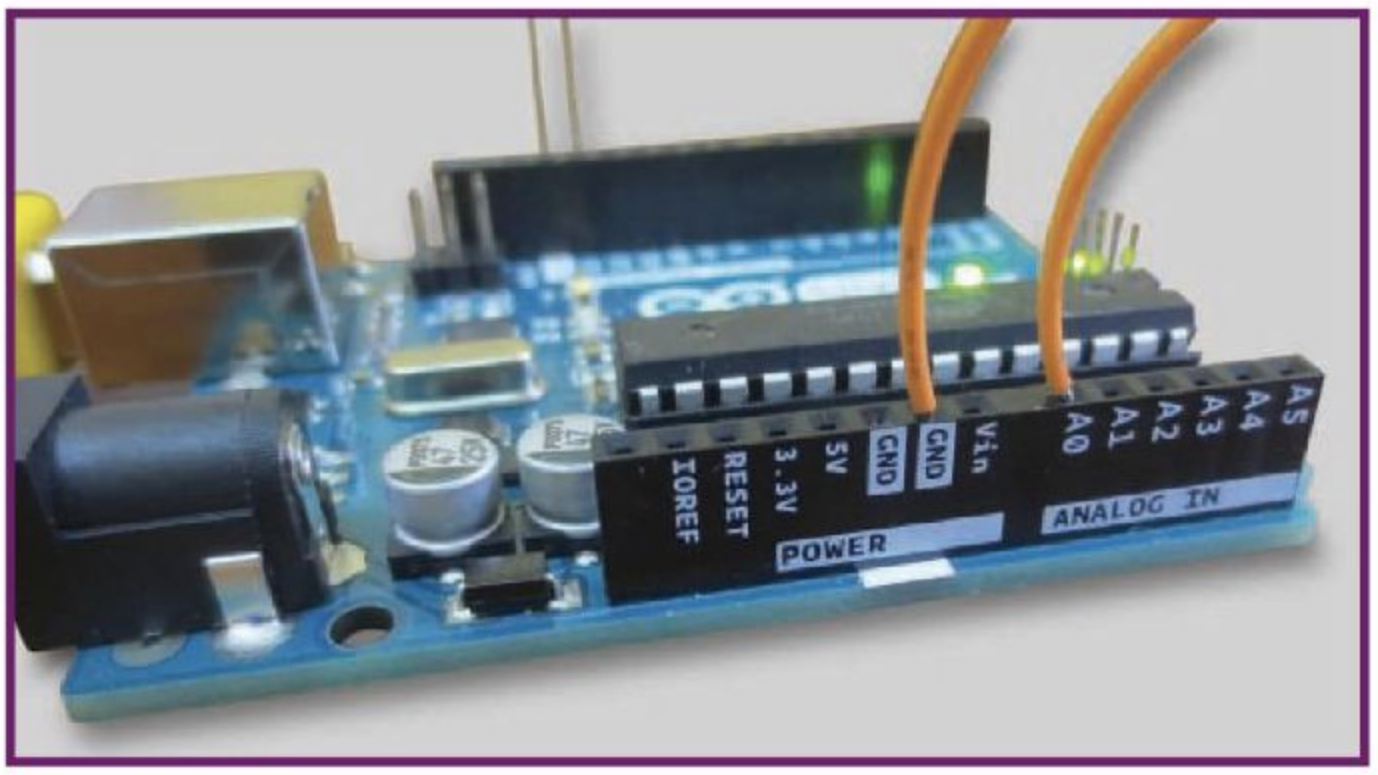
3 Ahora unimos la resistencia y la fotorresistencia, para lo cual usamos un cable de puente que una el extremo derecho de un elemento con el extremo derecho del otro, tal como se ve en la imagen de ejemplo.



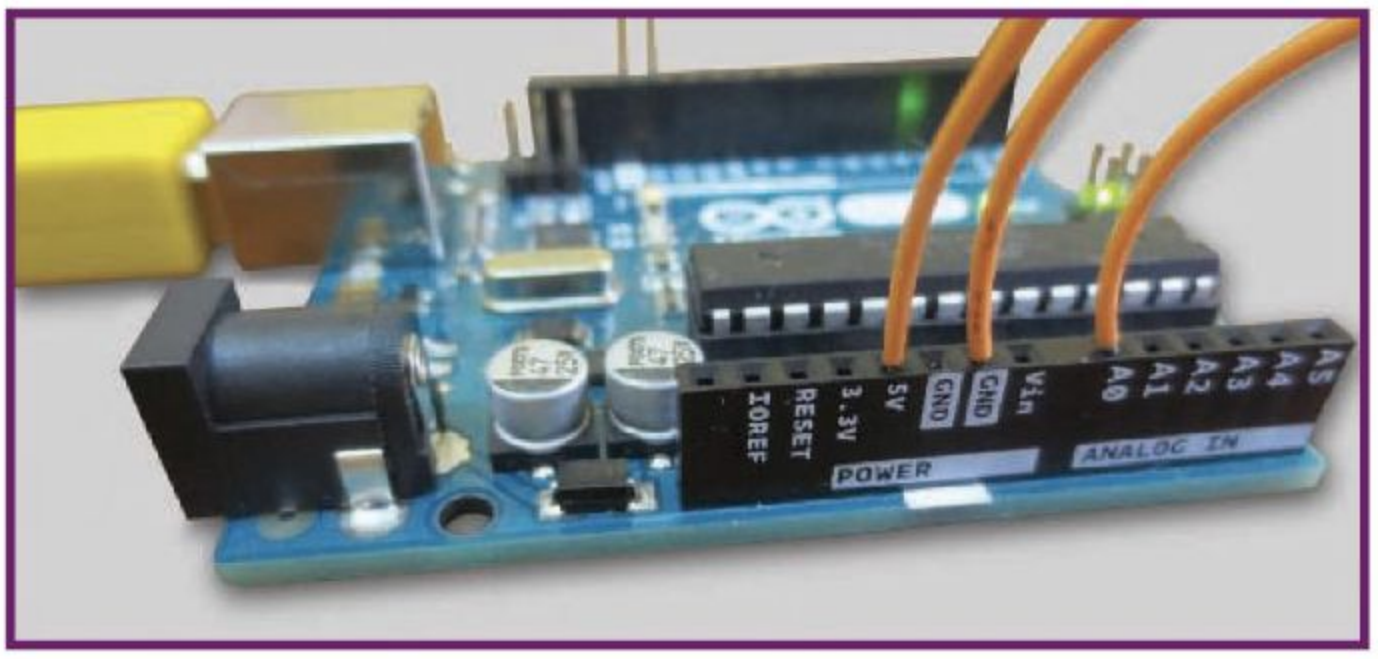
4 En este punto tenemos listas las conexiones necesarias en el protoboard, y debemos conectarlo a la placa Arduino, pero eso será en un paso posterior. En este momento, preparamos los cables de puente que usaremos para conectar Arduino con el protoboard.



5 En primer lugar, conectamos el pin A0 de la placa Arduino UNO con un extremo de la resistencia. Para esto, debemos conectar el cable de puente en la placa y, luego, en la línea que corresponde a uno de los extremos de la resistencia.



6 Conectamos 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.



7 El tercer cable de puente proporcionará el voltaje necesario al circuito; para esto, conectamos uno de sus extremos al pin 5 V de la placa Arduino UNO, el otro extremo debe ser conectado al extremo que queda libre en la fotorresistencia. Para conectarlo, utilizamos uno de los puntos de conexión que corresponde a la línea donde se encuentra conectada la fotorresistencia.

8 En este punto ya tenemos el circuito creado; como podemos notar, solo hemos utilizado unos pocos componentes. Para terminar será necesario conectar la placa Arduino UNO a la computadora. Como siempre, tendremos que utilizar el cable USB adecuado. Efectuamos la conexión con la PC y verificamos que la placa se encuentre energizada, observando los LEDs incorporados en Arduino UNO.

Ya tenemos el circuito creado, se trata de la disposición básica que nos servirá para probar diversos sketches que nos permitirán trabajar con la fotorresistencia. Es decir, trabajaremos teniendo en cuenta la cantidad de luz que recibe este sensor.

Primera versión del código

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.

Resultado 1

Para concretar este resultado, recordemos el primer planteamiento:
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 el **LED**; de lo contrario, lo apagaremos. El código 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 definido 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 lo cual 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.

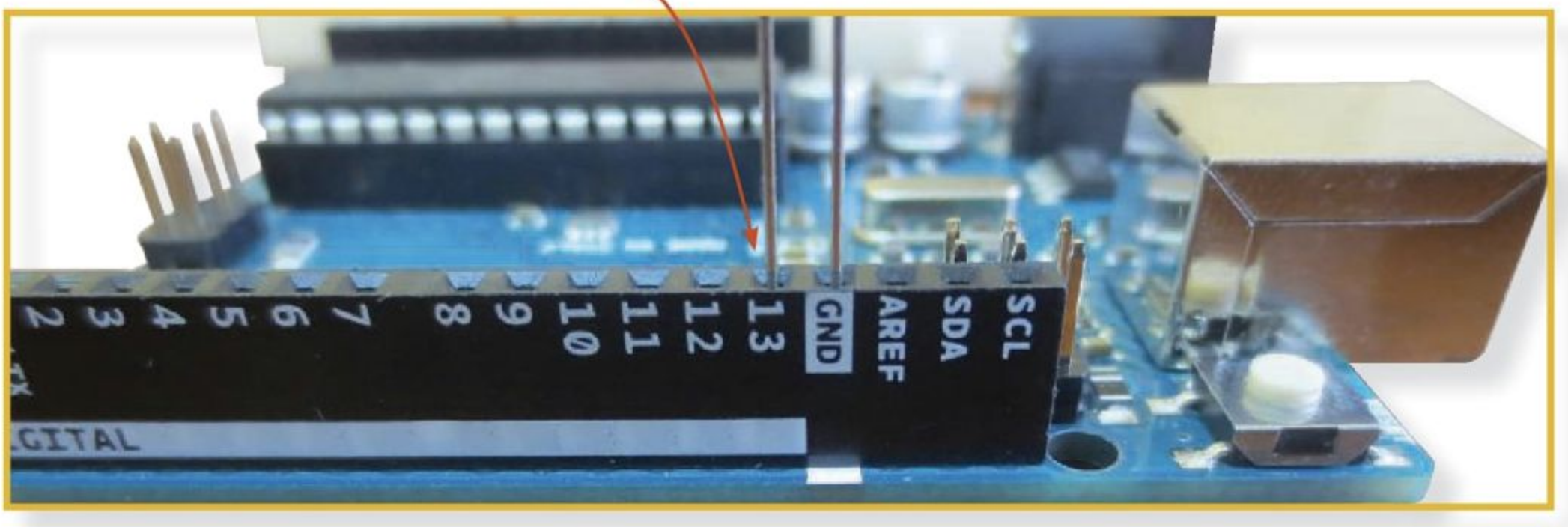
Ya con el sketch compilado, lo cargamos en la placa Arduino, mediante un clic en la opción **Subir**, que se encuentra en la barra superior de herramientas dentro del IDE. Cuando la carga del sketch haya finalizado, podremos verificar que el LED **L** de la placa se encienda en presencia de un nivel 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;
```

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, sino que también lo hemos compilado, y no encontramos errores.

Otra modificación interesante que hará posible ver de manera 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.



Segunda versión del código

Para completar esta parte podemos basarnos en el código que escribimos en la sección anterior y realizarle algunas modificaciones.

Resultado 2

Para continuar con el proyecto que nos enseña a utilizar una fotorresistencia, recordemos el segundo planteamiento que nos impusimos:
Implementar el código necesario para hacer parpadear un LED mientras el sensor LDR conectado reciba la cantidad suficiente de luz.

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);
}
```

Un LED es un dispositivo diodo emisor de luz que se usa como indicador en muchos dispositivos y en iluminación.
Debemos tener en cuenta que un LED comienza a funcionar aproximadamente con 2 voltios.

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 es de **150**, el cambio estará en que esta vez no solo encenderemos y apagaremos el LED adecuado, sino que 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 se muestra a continuación:

```
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);
 }
}
```

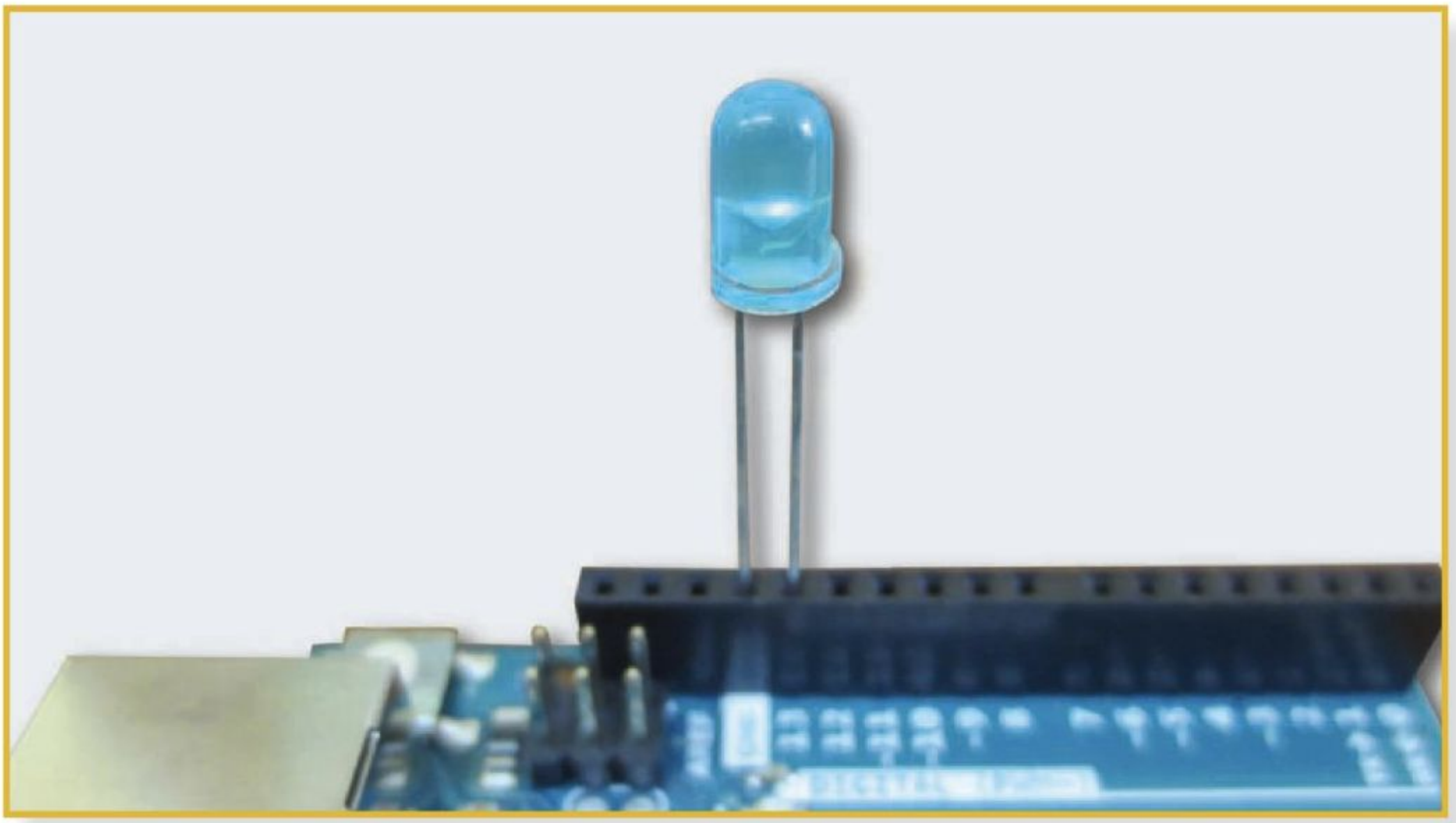
Una vez que hayamos compilado y cargado el sketch en la placa Arduino UNO, podremos verificar que el LED integrado parpadea mientras la fotorresistencia obtenga un nivel mayor de luz al indicado en la constante valor. Si cubrimos la fotorresistencia con la mano, el LED se apagará.

En este sketch podemos tocar algunos valores para obtener diferentes comportamientos. En primer lugar es posible cambiar la constante almacenada en **valor**, de modo que 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);
 }
```



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 es de 150.

Tercera versión del código

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.

Resultado 3

Recordemos la tercera actividad que nos propusimos realizar en este proyecto:
Implementar el sketch adecuado para leer los valores que corresponden al nivel de iluminación detectado por la fotorresistencia conectada a nuestro circuito.

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;
```

La variación de la resistencia es relativamente lenta, de 20 a 100 ms en función del modelo. Esta lentitud hace que no sea posible registrar variaciones rápidas, como las producidas en fuentes de luz artificiales alimentadas por corriente alterna.

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 precisamos incluir el siguiente código en **loop()**, se trata de los cálculos requeridos para luego mostrar los datos adecuados utilizando el monitor serie:

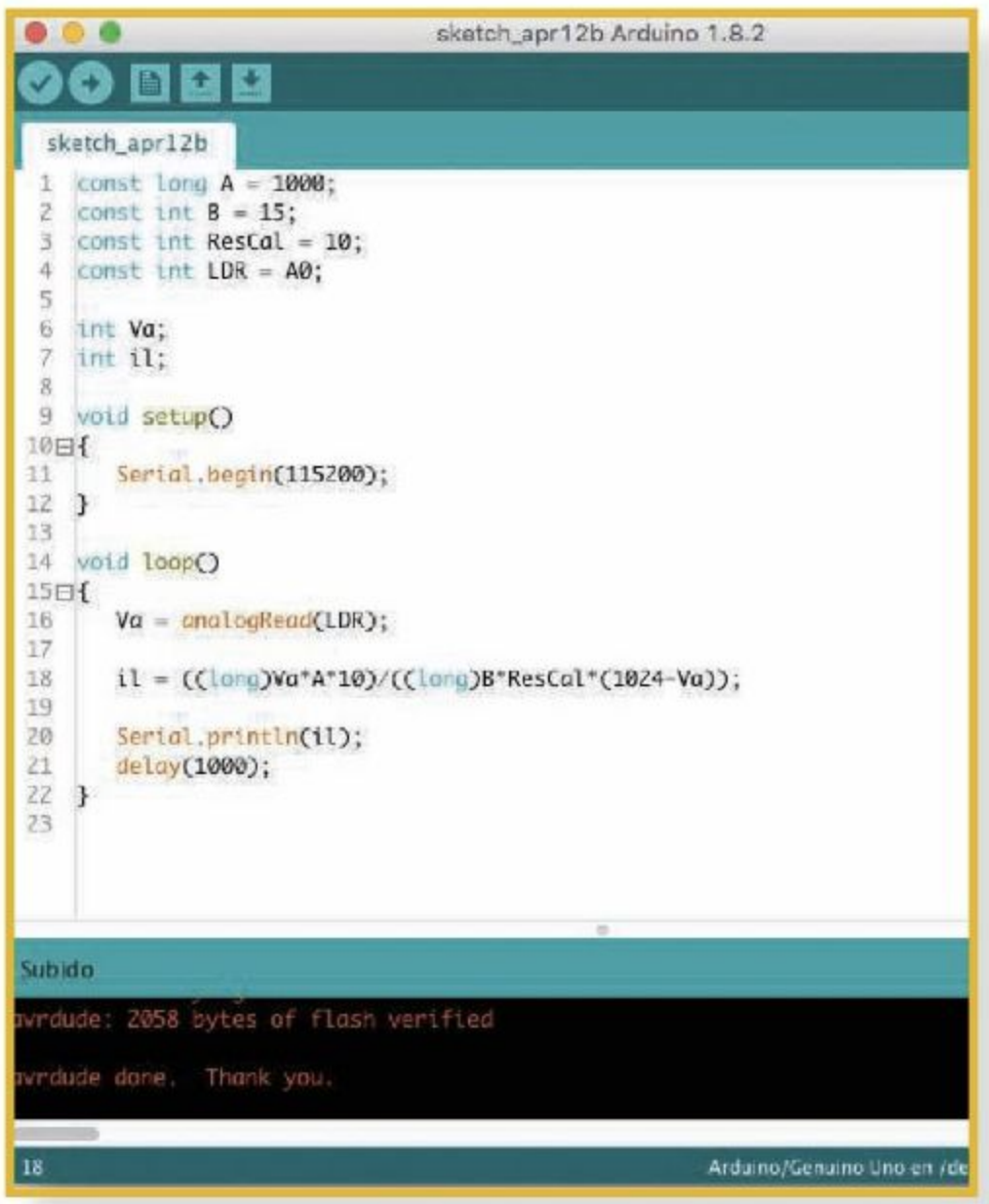
```
voidloop()
{
   Va = analogRead(LDR);

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.

Los fotorresistores no resultan adecuados para proporcionar una medición de la iluminancia, es decir, para servir como luxómetro Esto es debido a su baja precisión y su fuerte dependencia con la temperatura.

Buzzer

Para llevar a cabo este proyecto, necesitamos contar con algunos elementos que hemos presentado y utilizado en secciones 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.
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; su precio no supera el par de dólares.

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, 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 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.

Es posible encontrar los buzzers aplicados en el diseño de alarmas y controles acústicos, que requieran un rango de frecuencia estrecho, por ejemplo, en aparatos domésticos y de medicina.

En el mercado también se encuentran buzzers integrados en módulos.

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 lo cual 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);
}
```

Buzzer activo

Los buzzer activos disponen de un oscilador interno, por lo que únicamente tenemos que alimentar el dispositivo para que se produzca el sonido. El buzzer activo se se identifica por una pegatina y al alimentarlo entre 5V y GND suena a una frecuencia fija.

Proyecto básico

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 varios objetivos:

1. **En primer lugar, conectaremos el buzzer directamente a la placa Arduino UNO y generaremos el código necesario para que este componente emita sonidos.**

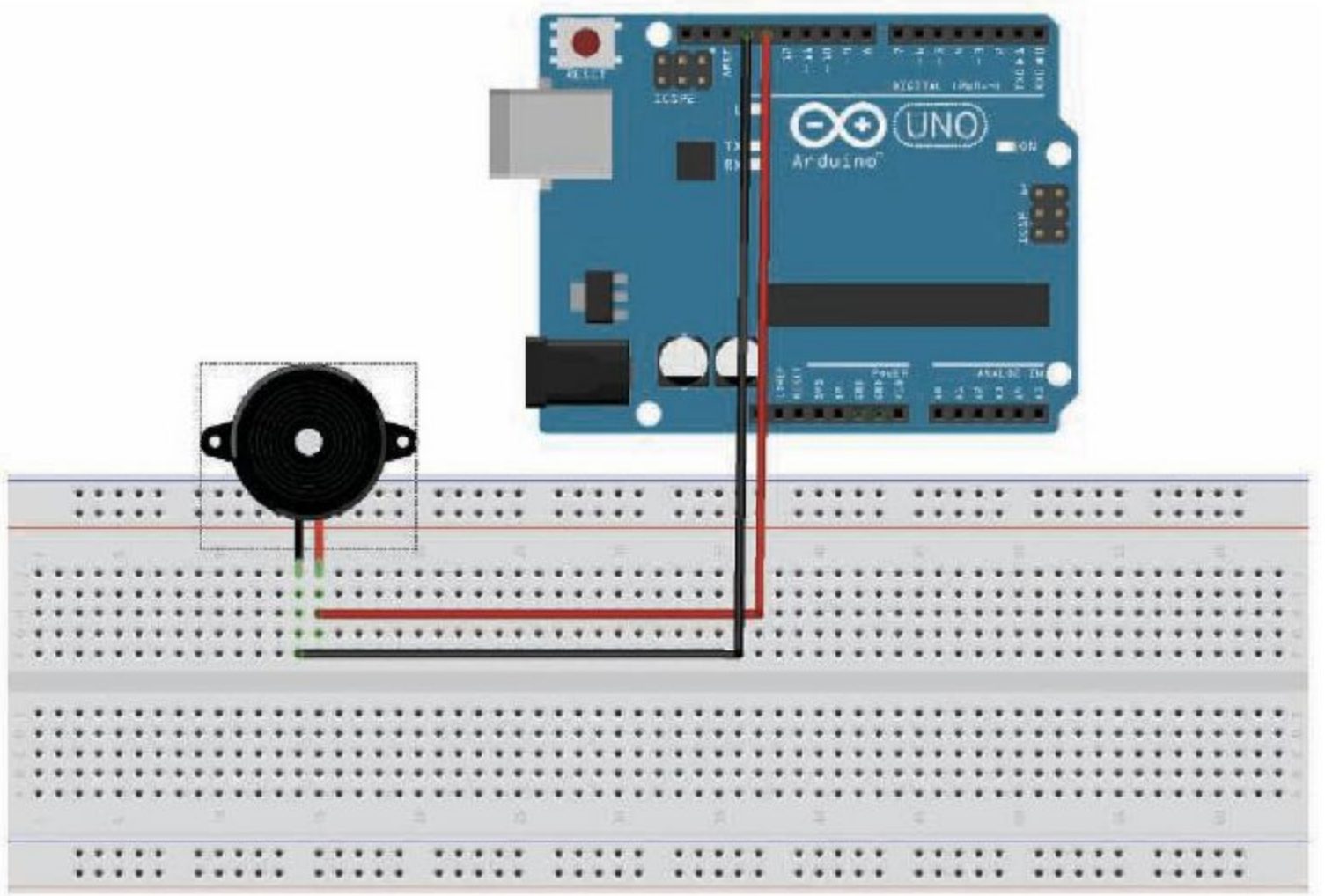
2. **Con el buzzer conectado a la placa Arduino, intentaremos reproducir una serie de sonidos que representen una escala musical.**

3. **Continuando con el mismo circuito original, combinaremos una cantidad de sonidos para que se escuche una melodía reconocible; para esto trabajaremos sobre el sketch básico que resultó de la tarea número dos.**

Manos a la obra

Para las tareas que nos propusimos en este proyecto, dividiremos el trabajo en secciones; en cada una de ellas crearemos el circuito y el código que necesitamos, de modo que iremos completando el proyecto mientras realizamos las tareas propuestas.

Para comenzar a trabajar, recordaremos nuestra primera tarea: “En primer lugar conectaremos el buzzer directamente a la placa Arduino UNO y generaremos el código necesario para que este componente emita sonidos”.



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 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. En primer lugar, 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;
```

Para este sketch no precisamos 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);
```

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. Así, 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);
```

Reproducir archivos de audio

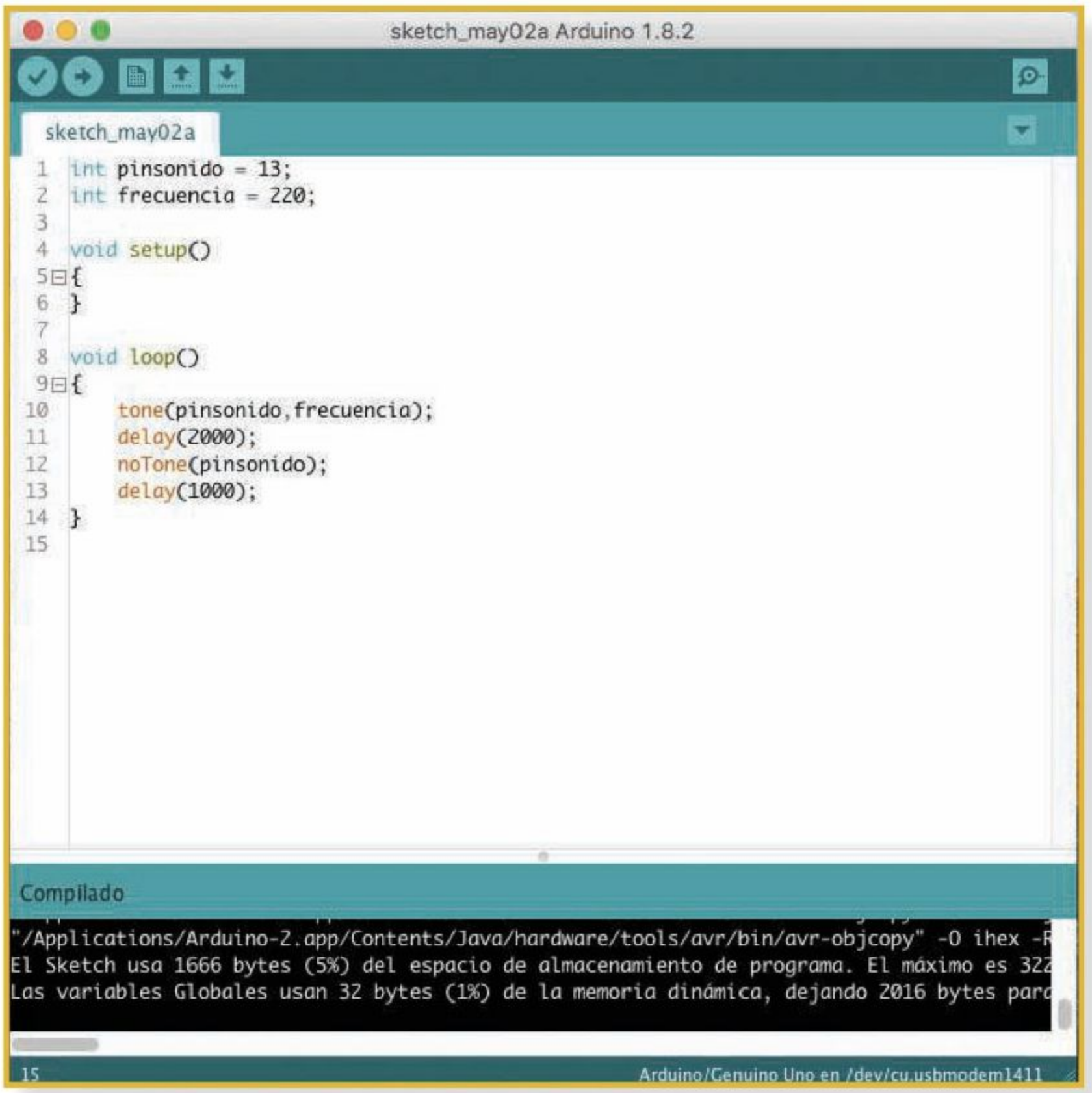
Gracias a la librería SDplayWAV**, es posible reproducir archivos de audio que se encuentren almacenados en una tarjeta SD conectada a la placa Arduino. Esta librería fue desarrollada por David Cuartielles, uno de los desarrolladores de Arduino, y la podemos conseguir visitando la dirección web** http://blushingboy.net/p/SDplayWAV**. La librería se descarga en formato zip y posee un peso de 59,71 KB.**

Con esto, nuestro código completo quedará de la siguiente manera:

```
intpinsonido = 13;
int frecuencia = 220;

voidsetup()
{
}

voidloop()
{
tone(pinsonido,frecuencia);
delay(2000);
noTone(pinsonido);
delay(1000);
}
```

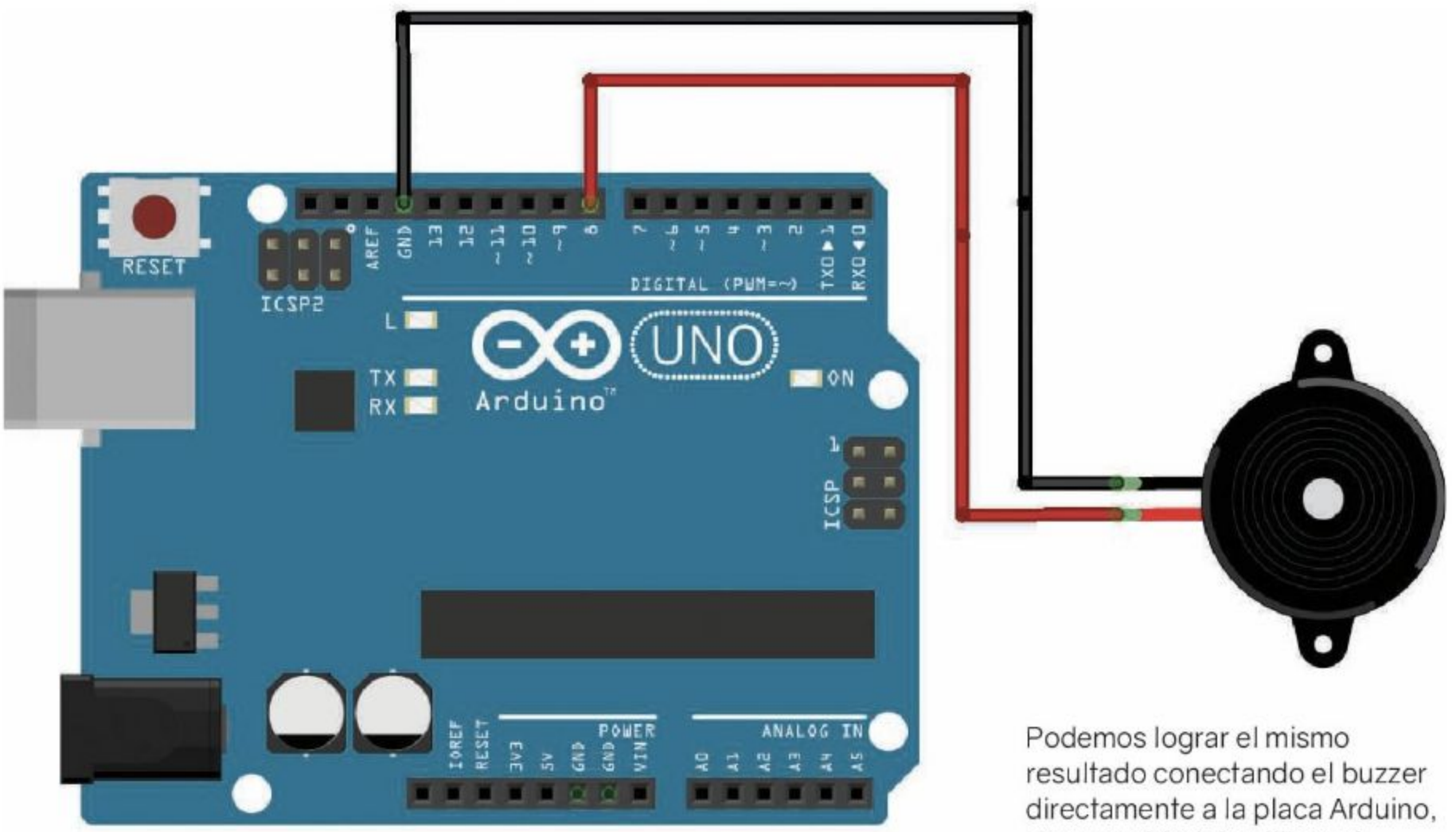
Ahora podemos ejecutar el Arduino IDE y cargar el código propuesto, luego lo compilamos para verificar que no exista ningún error.

La conexión del circuito es sencilla, negativo a GND. De todas maneras hay que tener cuidado pues los buzzers tienen polaridad y debemos asegurarnos de conectarlos correctamente. Si los conectamos al revés, simplemente no sonará, y tendremos que conectar otra vez.



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.

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.
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.



Podemos lograr el mismo resultado conectando el buzzer directamente a la placa Arduino, sin necesidad de utilizar el protoboard como intermediario.

Piezoelectricidad

La piezoelectricidad es un fenómeno que ocurre en determinados cristales que, al ser sometidos a tensiones mecánicas, adquieren una polarización eléctrica; entonces aparece una diferencia de potencial y cargas eléctricas en su superficie que generan una tensión eléctrica.

Reproducir una escala musical y una melodía

Para completar esta segunda tarea, debemos reproducir una escala musical utilizando el buzzer que ya conectamos a la placa Arduino UNO.

Como ya hemos creado el circuito básico, solo nos queda trabajar sobre el código que necesitamos. En primer lugar, 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 este modo, 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.

Para lograr la última tarea, recordemos la propuesta número tres: "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 ya creamos:

```
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 manera:

```
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; así, 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);
```

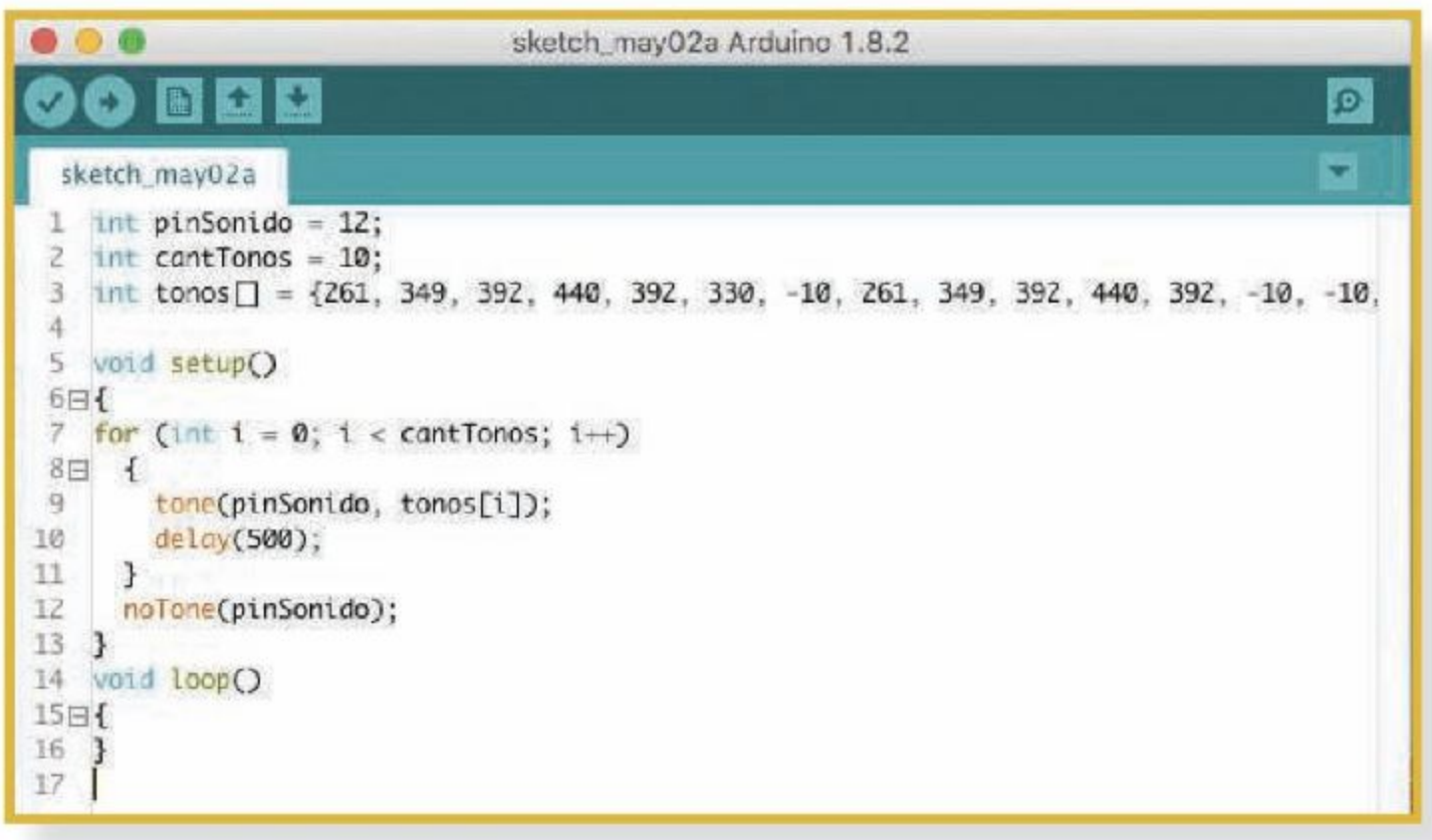
Dejaremos **loop()** vacío, de la siguiente forma:

```
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.

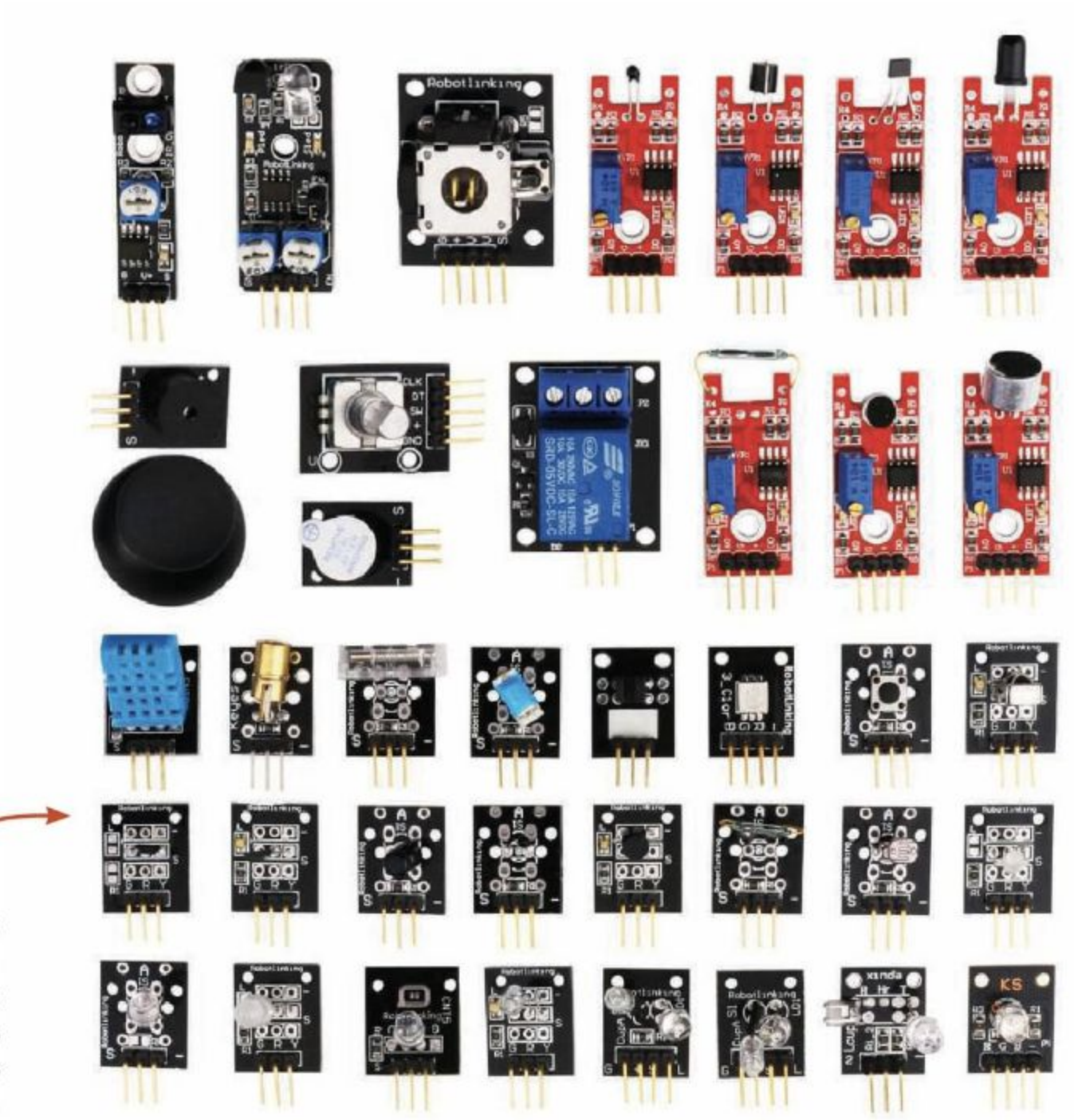


Ya hemos escrito y compilado el sketch en el Arduino IDE; podemos ver que la compilación se realizó sin errores lógicos, por lo que es posible subirlo a la placa Arduino.

¿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**. Son muy diversas, por ejemplo: temperatura, distancia, humedad, movimiento, presión, desplazamiento, pH, entre muchas otras.
El sensor tomará estas variables de instrumentación y las convertirá en magnitudes eléctricas, como resistencia eléctrica, capacidad eléctrica, tensión o corriente, etcétera.

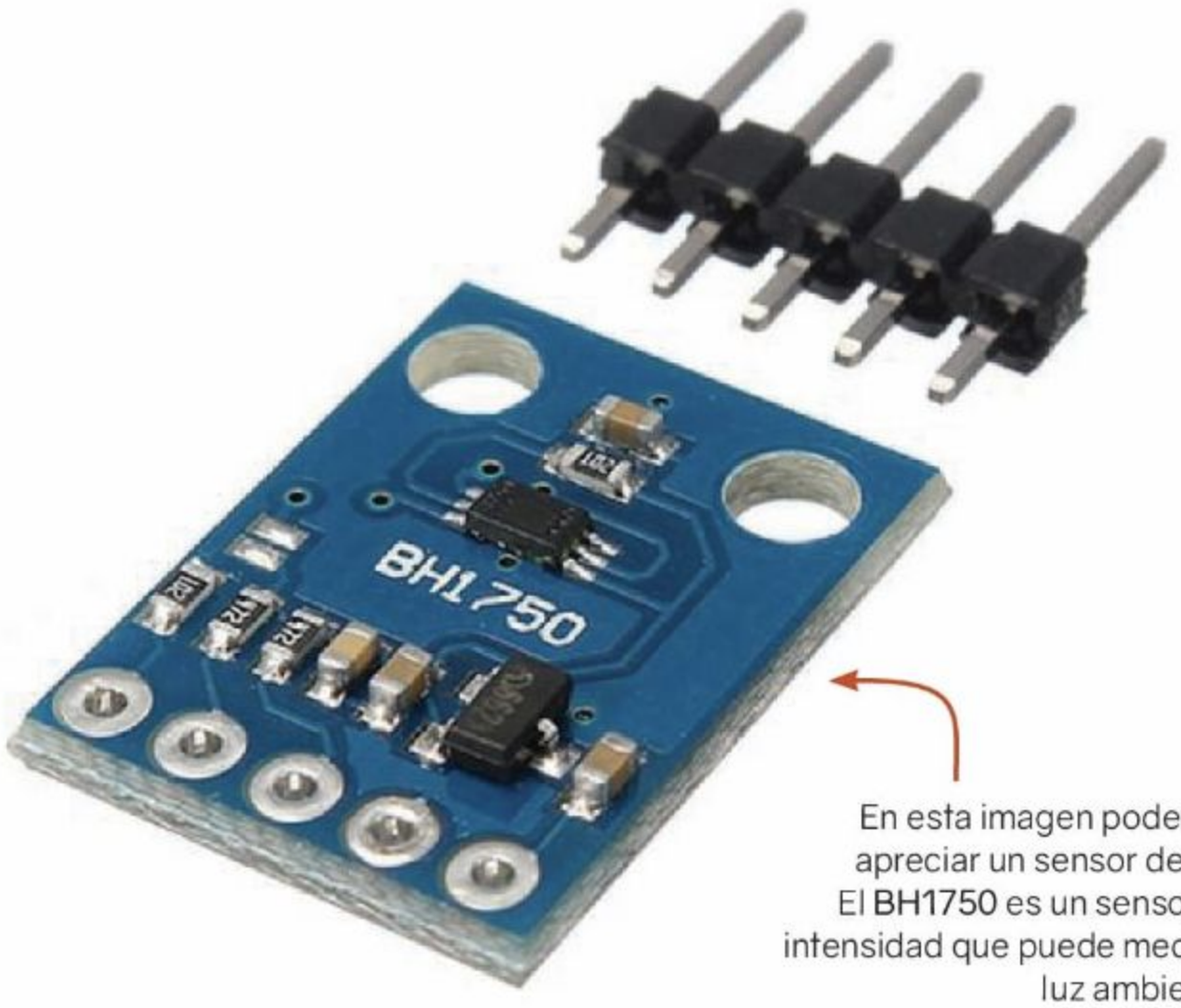


Existe una gran cantidad de sensores, preparados para trabajar con diferentes magnitudes o variables de instrumentación.

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 puedan 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.

Una señal analógica es capaz de tomar cualquier valor en su voltaje. Se trata de una señal que se pueden 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 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, existen magnitudes que nos ofrecerán lecturas de salida que se extienden en una línea donde existen 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 extraen 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.

Entradas en Arduino

Como sabemos, la placa Arduino es capaz no solo de enviar señales, sino que además puede leerlas, para lo cual utiliza entradas tanto analógicas como digitales. El principal objetivo de estas entradas es recibir datos de los sensores conectados a la placa y también comunicarse con los shields, que conoceremos más adelante.

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 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.

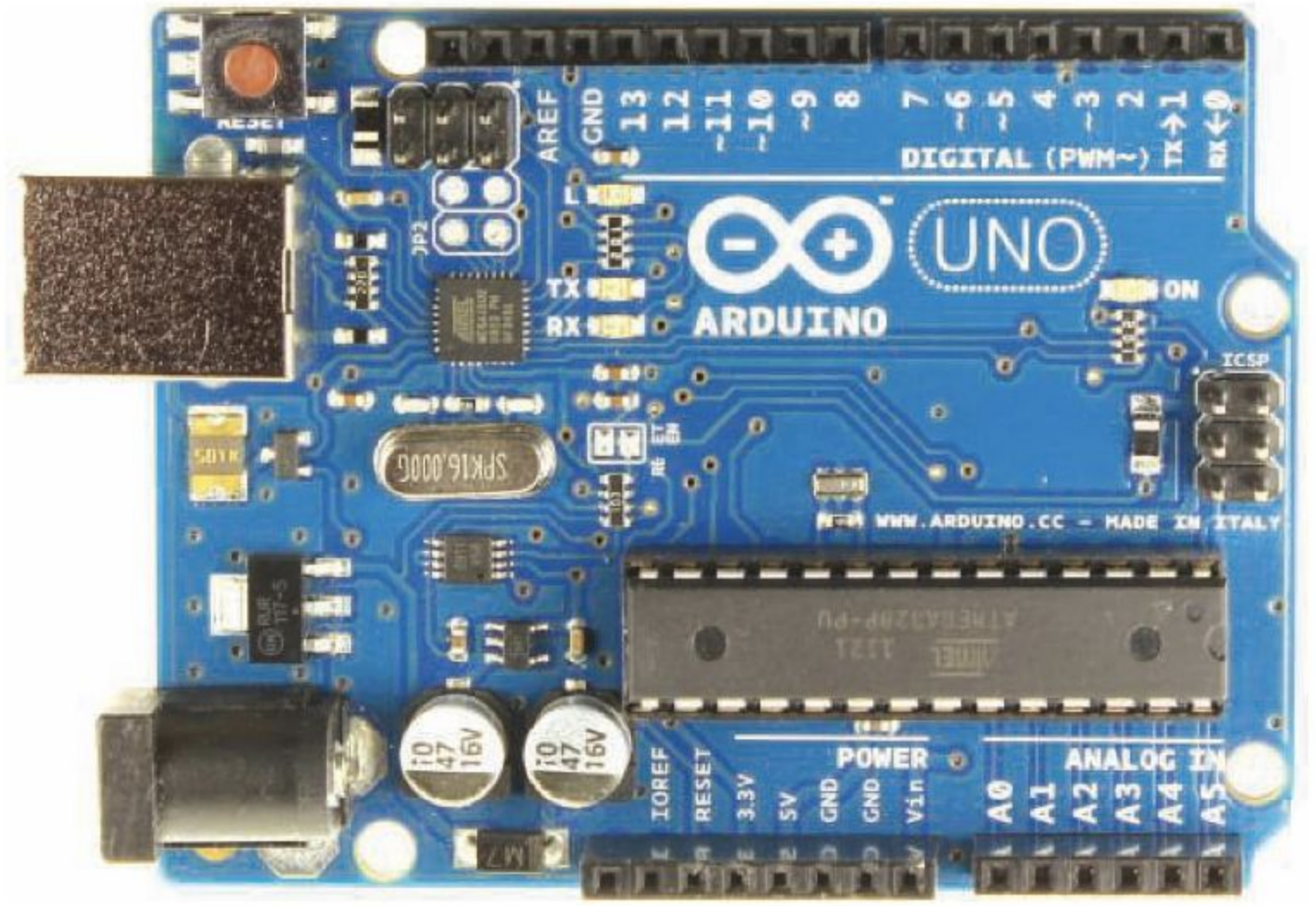
La sintaxis básica para leer la información desde una de estas entradas es la siguiente:

```
leer = analogRead(pin);
```

Por supuesto, debemos reemplazar ciertas partes de este código dependiendo de lo que deseamos. 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**.



La posición adecuada de las entradas analógicas de la placa Arduino UNO son los seis pines rotulados desde el A0 al A5.

Una señal analógica es una magnitud que puede tomar cualquier valor dentro de un intervalo –Vcc y + Vcc.

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. Con una entrada nula, el valor que veremos será cero; con 2,5 V, el valor presentado será 511; para una lectura de 5 V, el valor será 1023.

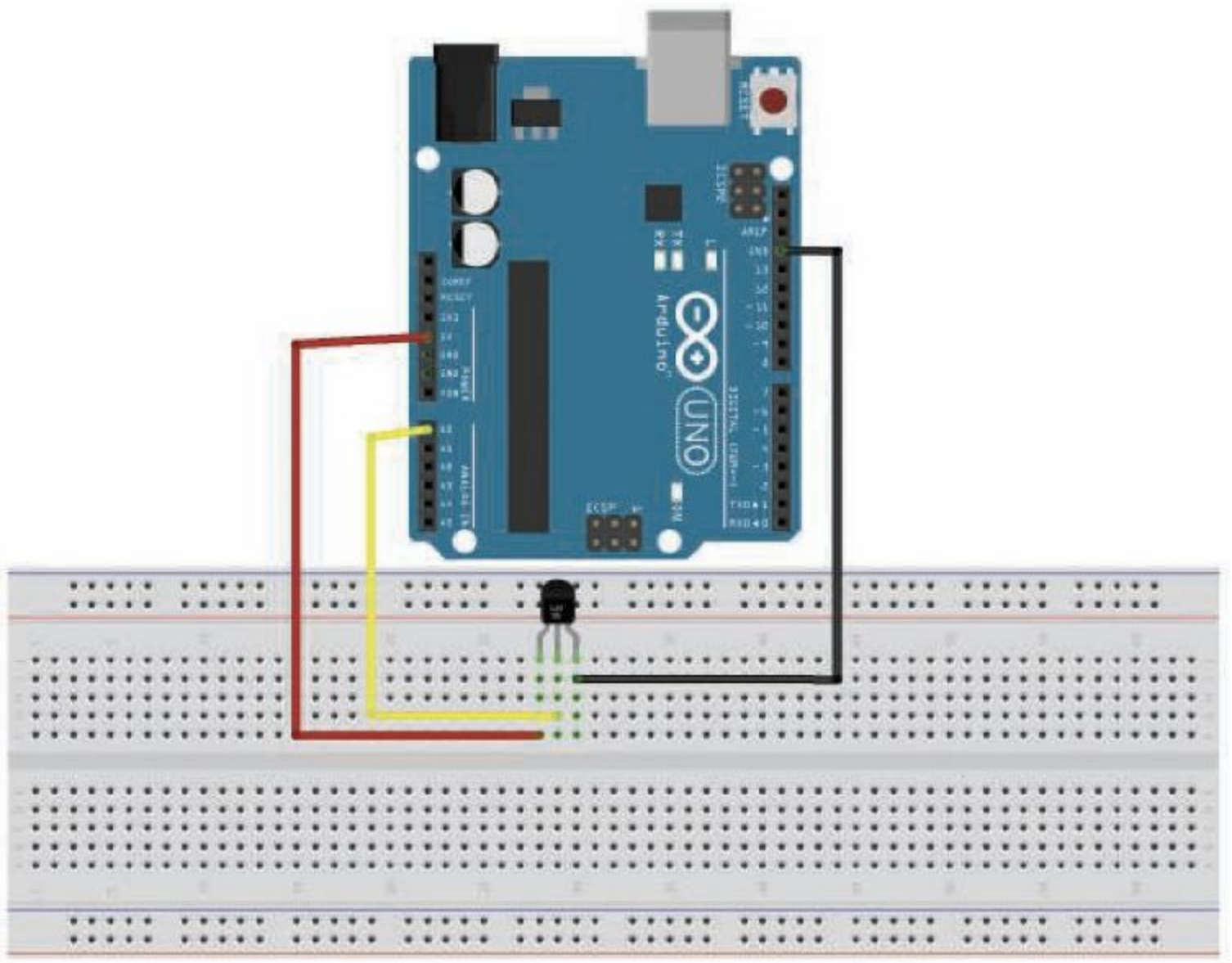
Leer un sensor de temperatura

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 0,5º C mientras que su sensibilidad es de 10 mV/ºC, está calibrado para trabajar con grados Celcius y puede leer valores entre los -50º C y los 150º C.

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 **5 V**, la pata **Vout** se conectará al pin que utilicemos para realizar la lectura, y la pata restante deberá conectarse al pin **GND**.



En este esquema podemos verificar la manera correcta de efectuar las conexiones para, en forma posterior, leer los datos entregados por el sensor LM35.

Una señal analógica de tensión entre 0 V y 5 V podría valer 2,72 V, o cualquier otro valor con cualquier número de decimales.

Una vez que hemos realizado las conexiones adecuadas, trabajaremos en el código, para lo cual necesitaremos iniciar el Arduino IDE.
Para comenzar, declaramos las variables adecuadas, una para almacenar la temperatura y otra para declarar el pin de entrada:

```
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, 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:

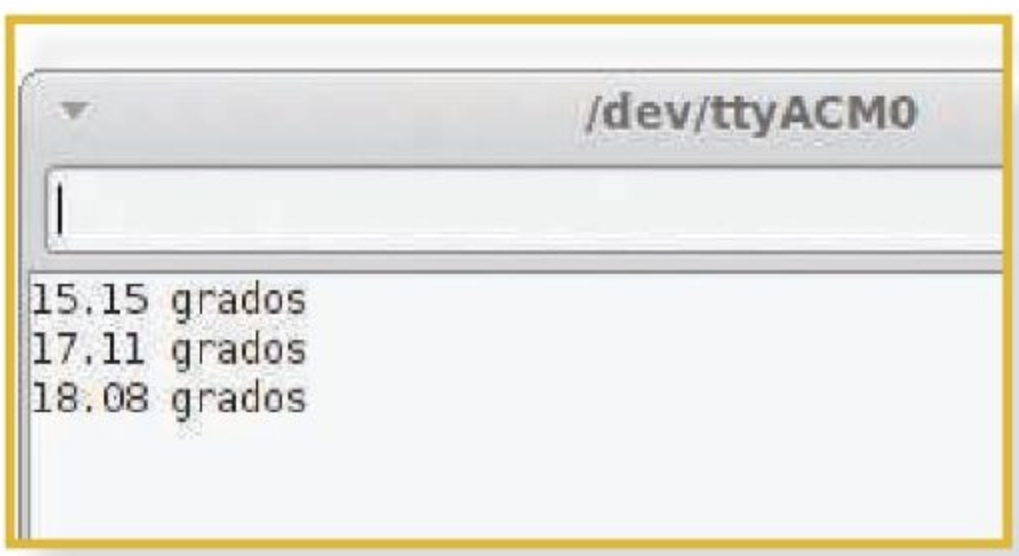
```
{
    leer-temp =
analogRead(pinentrada);
    leer-temp = (leer-temp / 1023 * 5
/ 0.01);
Serial.print(leer-temp);
Serial.print(“ grados \n”);
delay(2000);
}
```

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 sketches 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 se efectúan a través del pin digital; estas corresponden a las mediciones realizadas por el sensor de temperatura, con una espera de dos segundos entre cada medición. Si deseamos modificar esta espera, será necesario cambiar el siguiente valor:

```
delay(2000);
```



Con las conexiones realizadas y el sketch cargado, utilizamos el monitor serial para ver las mediciones de temperatura informadas 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 hacerlo, 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);
}
```

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:

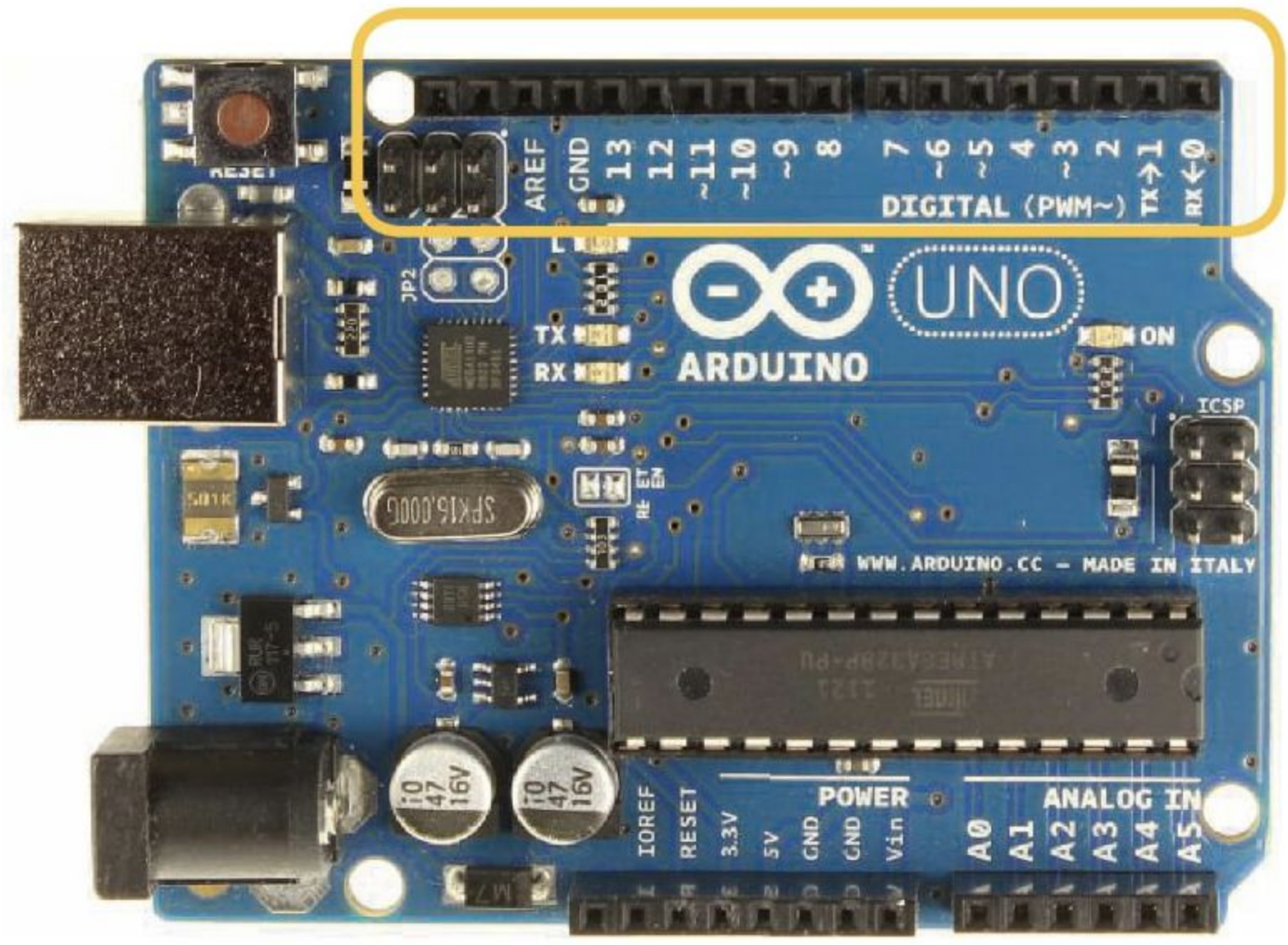
```
pinMode(pinentrada,INPUT);
```

Por otra parte, si necesitamos almacenar en una variable los valores captados en una entrada digital, debemos usar el código:

```
variable2 = digitalRead(pinentrada);
```

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.



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 en esta oportunidad 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 otras 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.

Los sensores son elementos cruciales en todos los proyectos que decidamos emprender con Arduino ya que permiten conocer las distintas magnitudes físicas o químicas que resultan de gran interés para nuestras aplicaciones.

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:

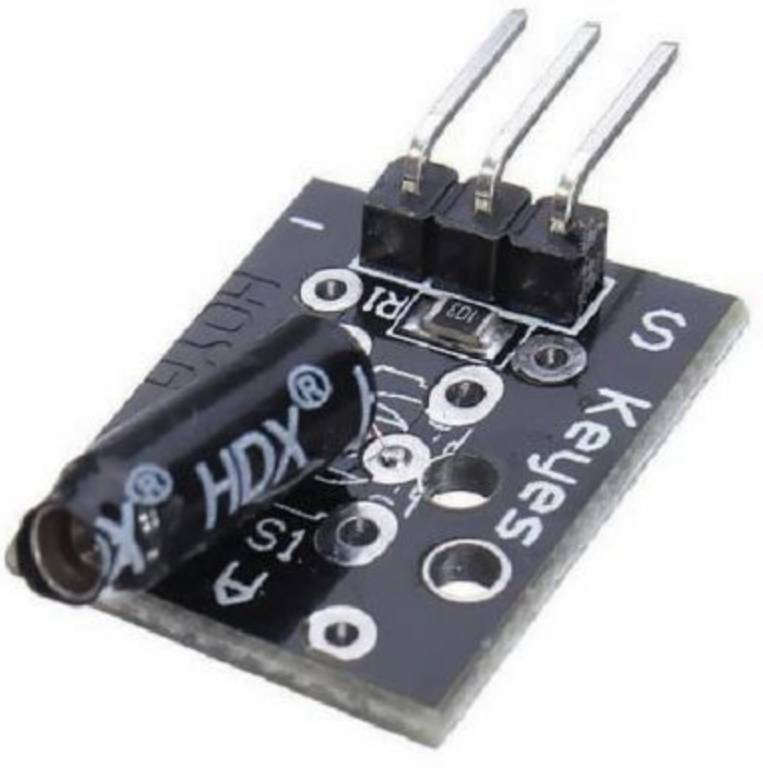
```
#include<OneWire.h>
OneWireds(10);
```

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 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);
  }
}
```



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.

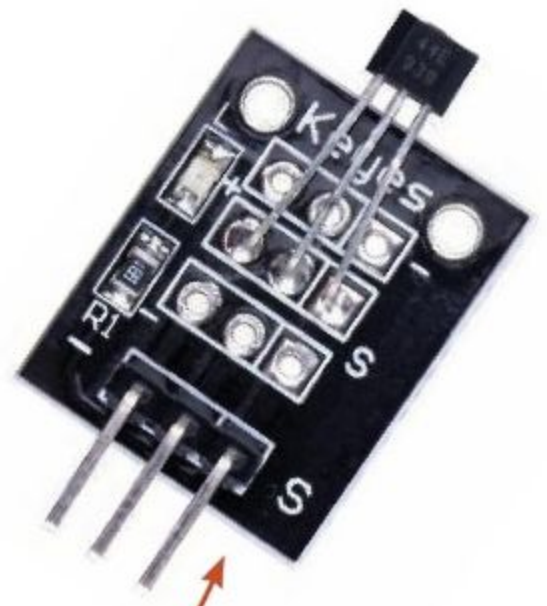
Sensor de campo magnético KY-003

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. 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:



El módulo KY-003 incorpora un sensor de campo magnético que funciona según los principios del efecto Hall.

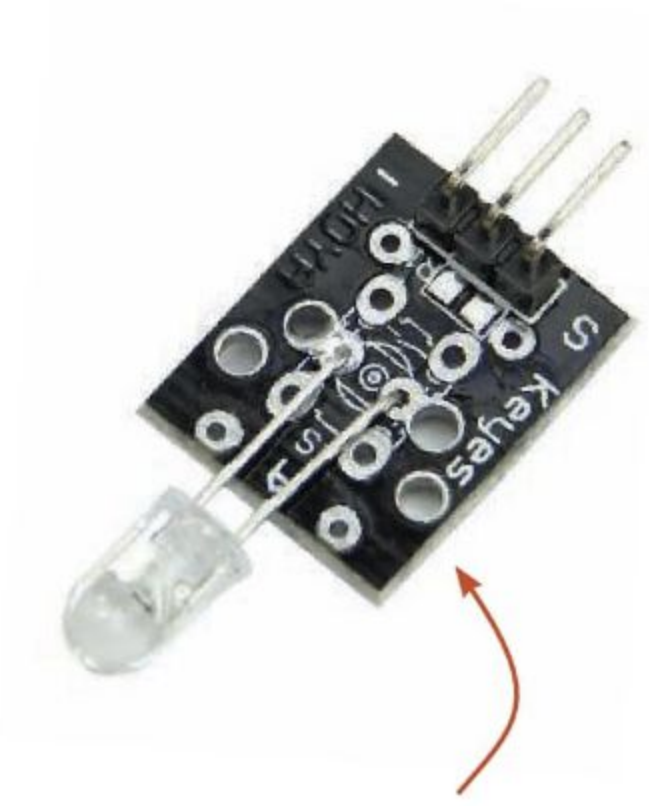
```
intLed = 13 ;
int SENSOR = 10 ;
int val ;

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, por ejemplo, al manipular televisores o reproductores de música.

El módulo **KY-005** incorpora un sensor emisor infrarrojo y posee las siguientes características:

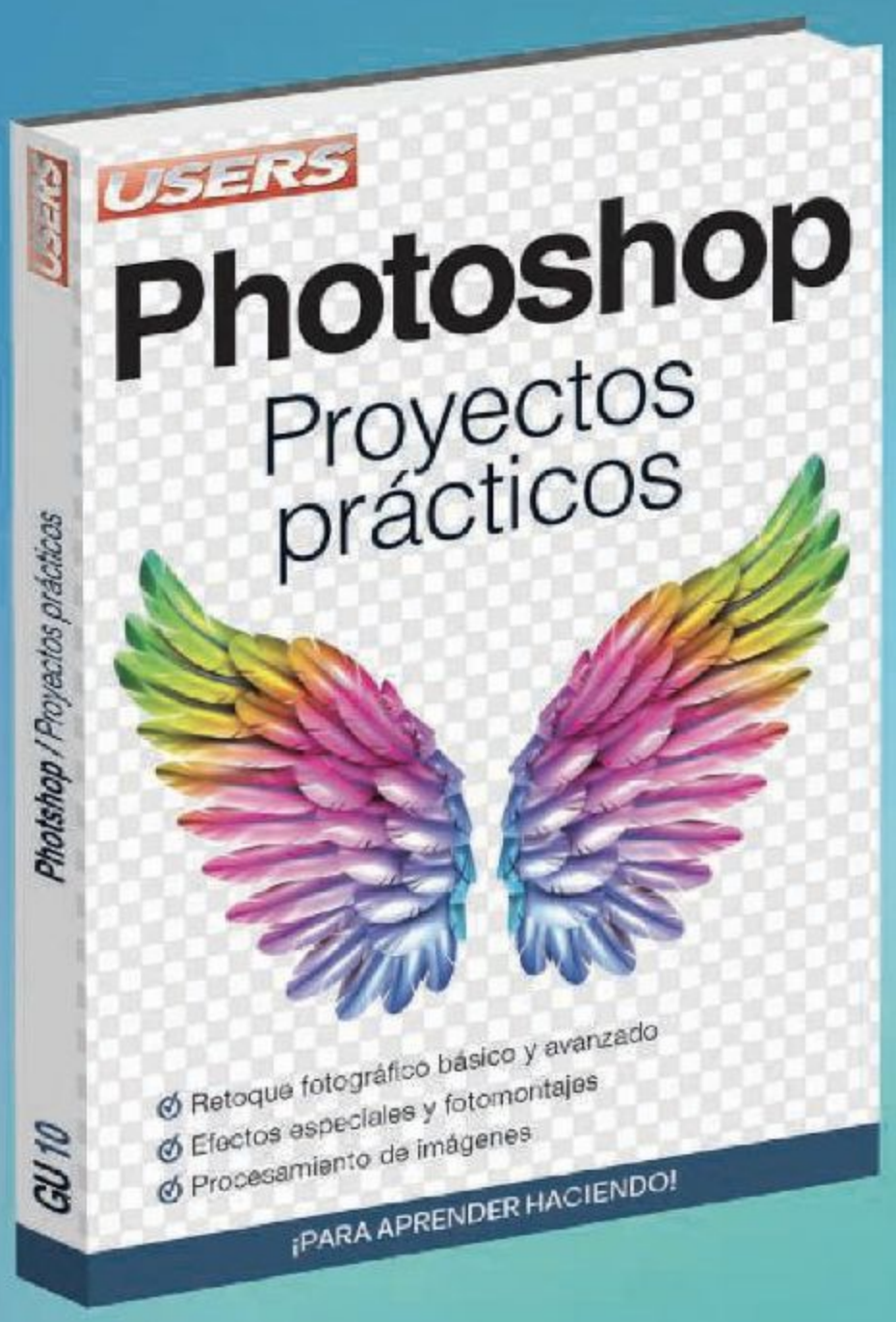
- Corriente: 30 a 60 mA
- Corriente de pulso: 0,3 a 1 A
- Voltaje inverso: 5 V
- Potencia de disipación: 90 mW
- Rango de temperatura: -25 a +80° C
- Temperatura de almacenamiento: -40 a +100° C

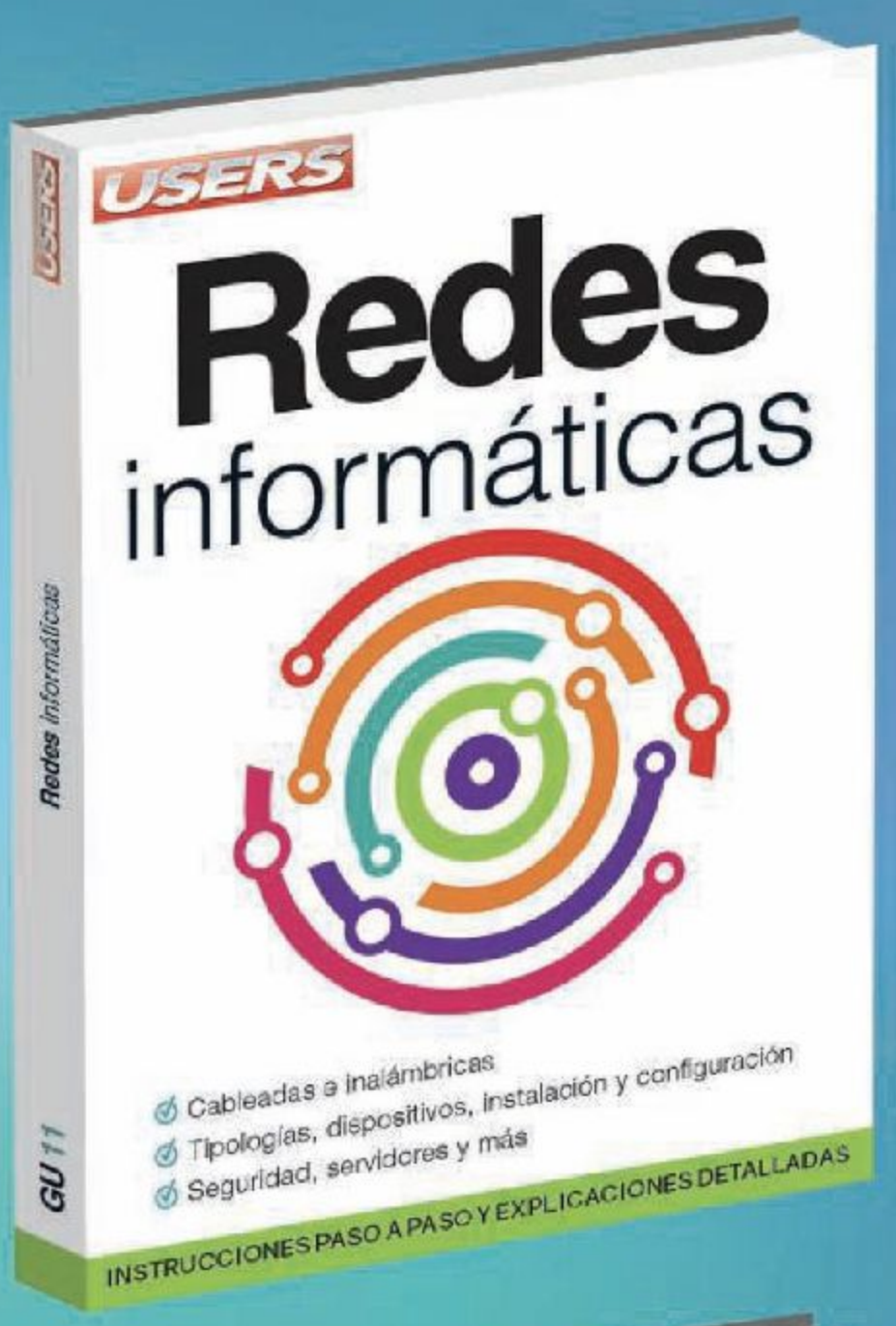


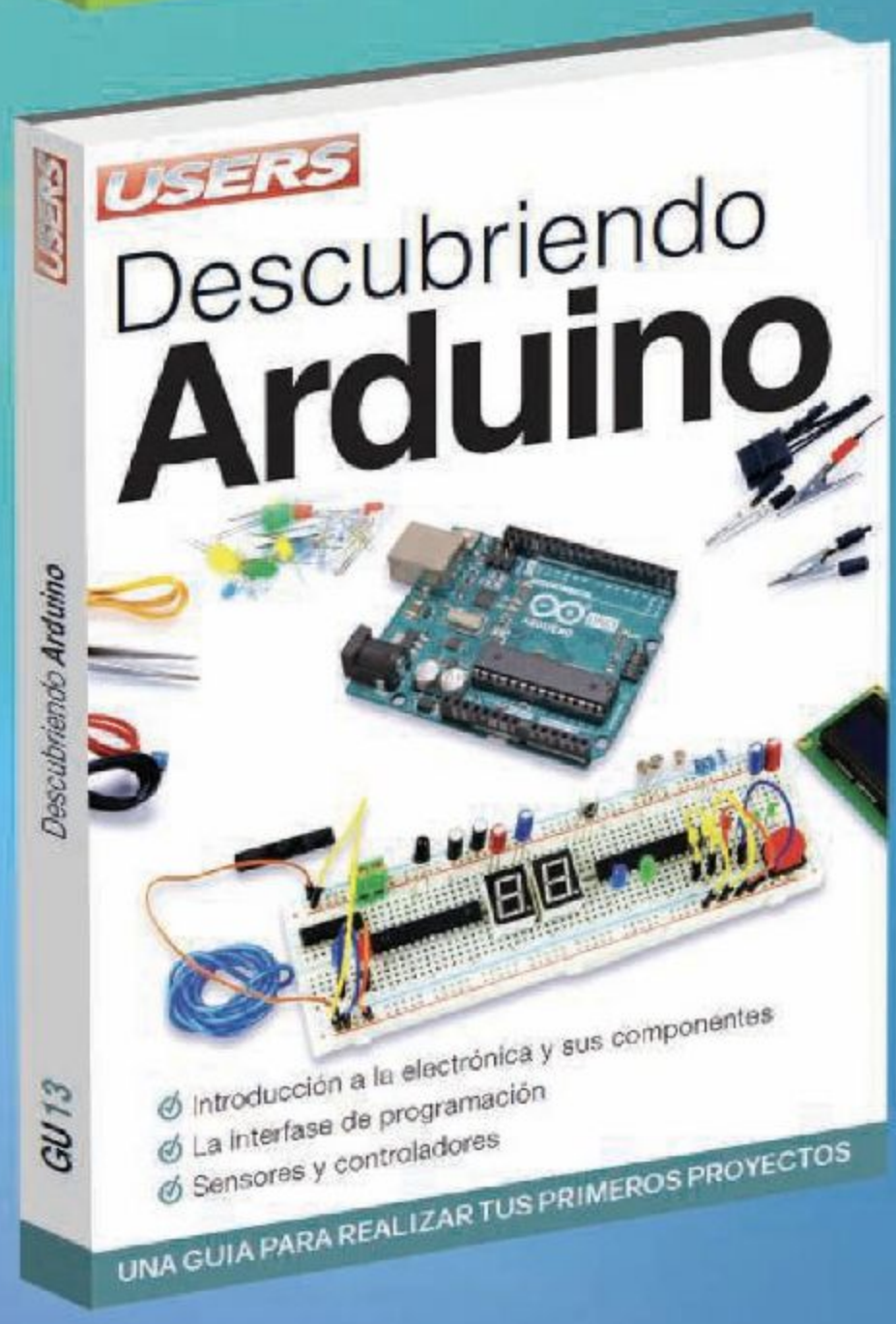
En esta imagen podemos ver el módulo KY-005, se trata de un sensor emisor de infrarrojo.

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 ();
}
}
```







SUSCRIBETE

usershop.redusers.com

+54-11-4110-8700

usershop@redusers.com

USERS

Descubriendo Arduino

Esta guía te abrirá el Mundo de la plataforma para proyectos electrónicos más utilizada en el mundo. Aprenderás elementos básicos de electrónica, los componentes necesarios, las distintas placas disponibles y la plataforma IDE para programar las placas. Veremos también algunos proyectos básicos para dar tus primeros pasos en esta fascinante plataforma.

Entre otros temas, aprenderás:

INTRODUCCIÓN
- Electricidad y electrónica
- Componentes y circuitos electrónicos
- Microcontroladores
- Placas de desarrollo

PRINCIPIOS BÁSICOS
- Comunidad
- Hardware y software
- Placas Arduino
- Placas no oficiales

COMPONENTES NECESARIOS
- Prueba de conexión
- Comparación con otras placas
- Protoboard y cables de puente
- Condensador
- Diodo / Diodo emisor de luz (LED)
- Puente H
- Potenciómetro
- Pantalla de cristal líquido
- Motor de corriente continua
- Optoacoplador / Pulsador
- Resistencias / Fotorresistencia
- Transistor / Zumbador piezoeléctrico
- Sensores / Servomotor

ARDUINO IDE
- Instalación del IDE
- Entorno de trabajo
- Configuración inicial
- Librerías

PROGRAMACIÓN
- Estructura de un sketch
- Funciones y parámetros
- Variables
- Datos y operadores
- Tipos de operadores
- Estructuras de control
- Bucles

LEDs
- Conexión básica de un LED
- ¿Cómo controlar el LED?
- Conectar el circuito
- Seis LEDs en secuencia
- Secuencia de 8 LEDs

DETECCIÓN DE LUZ
- ¿Cómo aprovechar una fotorresistencia?
- Tres versiones del código

SONIDO
- Buzzer
- Reproducir una escala musical y una melodía

SENSORES
- ¿Qué es un sensor?
- Entradas en Arduino
- Sensores para Arduino

¡REALIZA TUS PROPIOS PROYECTOS!

NIVEL DE USUARIO
Principiante

CATEGORÍA
Electrónica

ISBN: 978-987-4958-24-2

¡DE CERO A EXPERTO!