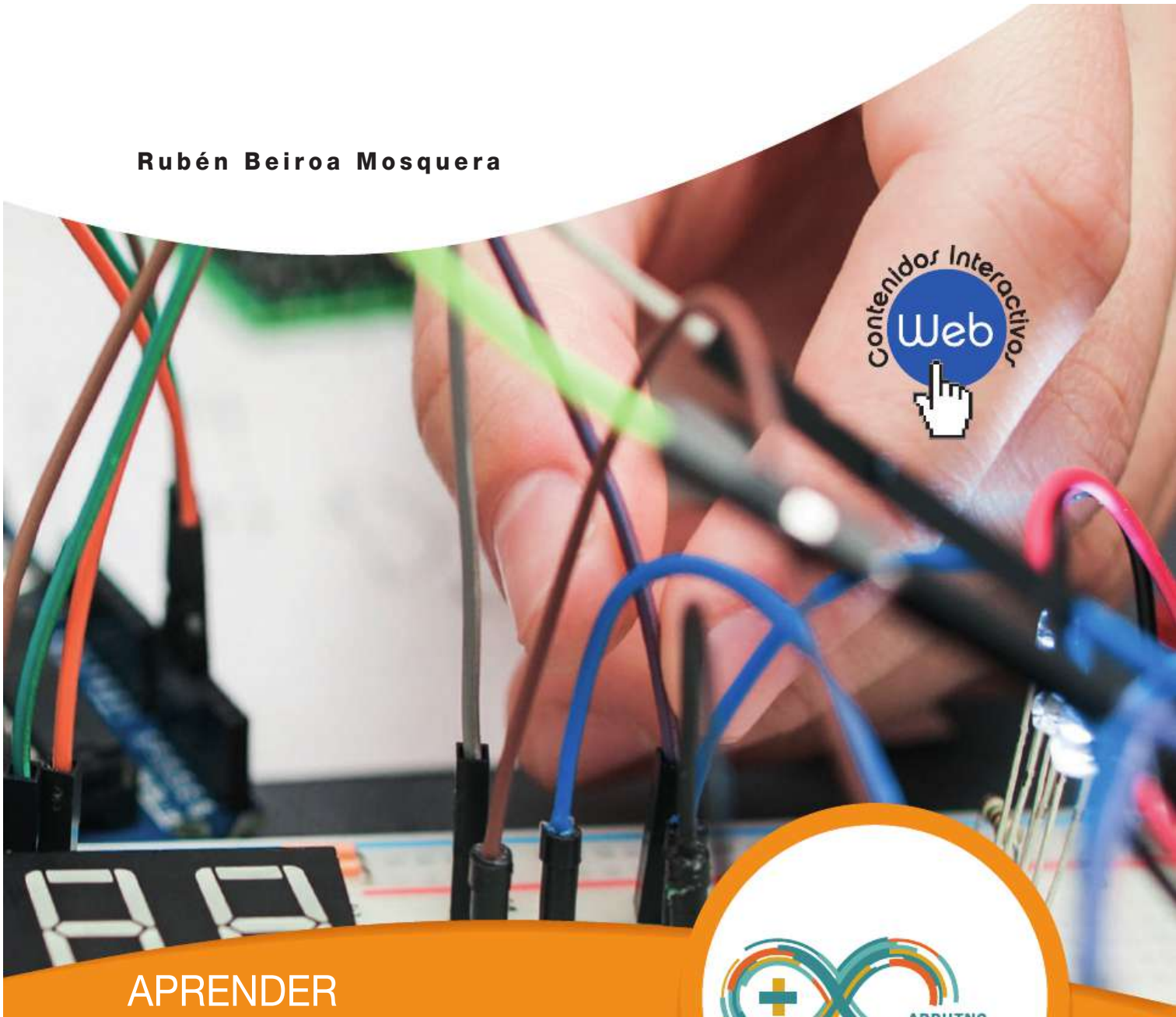
Rubén Beiroa Mosquera



APRENDER

Arduino, electrónica y programación

con 100 ejercicios prácticos

Alfaomega

Marcombo

Aprender

Arduino, electrónica y programación

con 100 ejercicios prácticos

Aprender

Arduino, electrónica y programación

con 100 ejercicios prácticos

Alfaomega

Marcombo

Diseño de la cubierta y maquetación: ArteMio
Correctora: Laura Seoane Sánchez-Majano
Directora de producción: Ma. Rosa Castillo Hidalgo
Revisión técnica: Pablo Martínez

Datos catalográficos

Beiroa, Rubén
Aprender Arduino, electrónica y programación con 100 ejercicios prácticos
Primera Edición
Alfaomega Grupo Editor, S.A. de C.V., México

ISBN: 978-607-538-379-8

Formato: 17 x 23 cm Páginas: 216

Aprender Arduino, electrónica y programación con 100 ejercicios prácticos
Rubén Beiroa Mosquera
ISBN: 978-84-267-2648-3, edición original publicada por MARCOMBO, S.A., Barcelona, España

Primera edición: Alfaomega Grupo Editor, México, enero 2019

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

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

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

ISBN: 978-607-538-379-8

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

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

Impreso en México. Printed in Mexico.

Empresas del grupo:

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

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

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

Argentina: Alfaomega Grupo Editor Argentino, S.A. – Av. Córdoba 1215, piso 10, CP: 1055, Buenos Aires, Argentina, – Tel./Fax: (54-11) 4811-0887 y 4811 7183 – E-mail: ventas@alfaomegaeditor.com.ar

Presentación

APRENDER ARDUINO, ELECTRÓNICA Y PROGRAMACIÓN CON 100 EJERCICIOS PRÁCTICOS

Los 100 capítulos que contiene este libro realizan un recorrido por las leyes básicas de la electrónica, los principios de la programación y las características de Arduino. Se recomienda leerlos por orden así como disponer del material básico para realizar los ejercicios prácticos: Arduino UNO, resistencias, protoboard, ledes, pulsadores y potenciómetros.

Una vez finalizado este libro, el lector habrá adquirido las técnicas y conocimientos para afrontar proyectos que en un principio se presentaban imposibles. Tendrá la capacidad de analizar soluciones existentes, realizar sus propias tesis y pruebas, y reutilizar código.

A QUIÉN VA DIRIGIDO

A todo aquel que busque iniciarse en la electrónica, la programación o adentrarse en el mundo de Arduino.

También está dirigido a aquellos que ya hayan realizado sus primeros pasos con Arduino, pues en este libro encontrarán las respuestas a las dudas que les surjan y aquellas que no hayan podido resolver hasta ahora.

LA FORMA DE APRENDER

Nuestra experiencia en el ámbito de la enseñanza nos ha llevado a diseñar este tipo de manual, en el que cada una de las funciones se ejercita mediante la realización de un ejercicio práctico. Dicho ejercicio se halla explicado paso a paso y pulsación a pulsación, a fin de no dejar ninguna duda en su proceso de ejecución. Además, lo hemos ilustrado con imágenes descriptivas de los pasos más importantes o de los resultados que deberían obtenerse y con recuadros IMPORTANTE que ofrecen información complementaria sobre los temas tratados en los ejercicios.

PLATAFORMA DE CONTENIDOS INTERACTIVOS

Para tener acceso al material de la plataforma de contenidos interactivos del libro, siga los siguientes pasos:

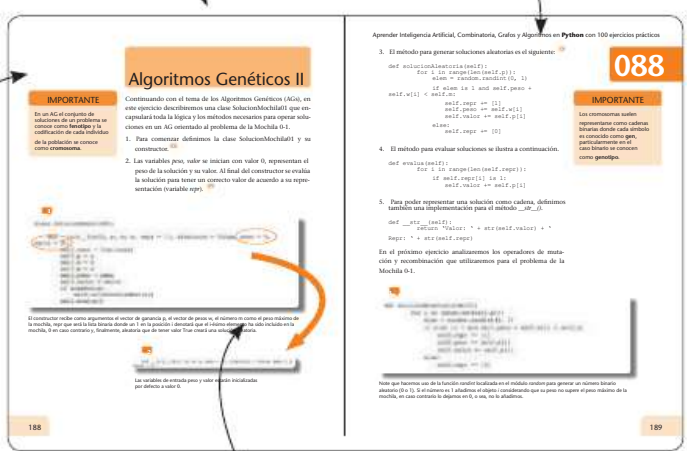
1. Ir a la página: **http://libroweb.alfaomega.com.mx**
2. Ir a la sección Catálogo y seleccionar la imagen de la portada del libro, al dar doble clic sobre ella, tendrá acceso al material descargable, complemento imprescindible de este libro, el cual podrá descomprimir con la clave **APARDUINO1**.

Cómo leer los libros "**Aprender...**"

El título de cada ejercicio expresa, sin lugar a dudas, en qué consiste este. De esta forma, si le interesa, puede acceder directamente a la acción que desea aprender o refrescar.

Puede seguir el ejercicio de forma gráfica y paso a paso. Los números colocados en las fotos le remiten a entradas en el cuerpo de texto.

El número a la derecha de la página le indica claramente en qué ejercicio se encuentra en todo momento.



Los recuadros Importante incluyen acciones que deben hacerse para asegurarse de que realiza el ejercicio correctamente. También contienen información interesante a aprender, porque le facilitará su trabajo con el programa.

Los ejercicios se han escrito sistemáticamente, paso a paso, para que nunca se pierda durante su realización.

A mis padres, por su apoyo, y a la
Editorial Marcombo, por su confianza.

Índice

Índice

¿Qué es Arduino?

IMPORTANTE

Puesto que Arduino es una plataforma son necesarios conocimientos sobre diferentes áreas o elementos que intervienen en el uso de Aduino:

Electrónica

Programación

Software

Hardware

Para que el aprendizaje sea más rápido y entretenido, en este libro se desarrolla mediante casos prácticos que permitan ir adquiriendo conocimientos sobre cada una de las cuatro áreas anteriormente mencionadas de forma paralela .

Arduino se ha convertido en un referente del hardware libre que surgió como una herramienta para estudiantes pero que, poco a poco, ha conseguido romper barreras. Cada vez son más las personas que ven en esta plataforma la herramienta idónea para aprender todo aquello que quisieron conocer y no pudieron.

Debido a su filosofía, a una gran comunidad de desarrolladores y desarrolladoras y a toda la tecnología y plataformas de las que disponemos, Arduino se ha convertido en una herramienta básica en el movimiento *maker* (crear objetos artesanales, pero utilizando la tecnología), la docencia (en las áreas de ciencia, ingeniería, tecnología y matemáticas), el impulso del IoT (Internet of Things o «Internet de las cosas») y el prototipado.

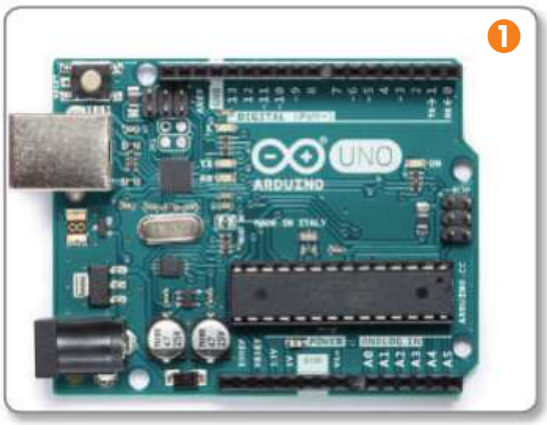
Todos aquellos que hayan estudiado electrónica o programación saben que muchos de los conceptos básicos son muy abstractos. Con Arduino esto cambia: en el momento en el que aprendamos, por ejemplo, una nueva instrucción, podemos realizar un programa de pocas líneas, conectar un simple led y, si el led se enciende (si ese es nuestro objetivo), significa que, no solo hemos aprendido algo nuevo, sino que hemos afianzado ese conocimiento.

¿Y si no funciona ese programa? Lo modificaremos las veces que sea necesario hasta que encontremos el error. Este proceso de aprendizaje es relativamente rápido. Sin embargo, Arduino no es solo una herramienta importante en el aprendizaje, sino también en el desarrollo de prototipos, en el que disponemos de un hardware totalmente operativo.

Empecemos por ver qué es Arduino: Arduino es una plataforma libre, educativa y de desarrollo. Generalmente, cuando se habla de un Arduino –«Estoy programando un Arduino»–, nos estamos refiriendo al elemento físico de la plataforma Arduino, su hardware ❶ (también conocido como tarjeta, placa o PCB). Arduino es algo más que un hardware, por lo que su definición más exacta es la de plataforma formada por: un hardware, un software ❷ (o entorno de programación) y un lenguaje de programación ❸.

La función de la plataforma Arduino es facilitar el uso de un microcontrolador (MCU) ❹ . A partir de ahora utilizaremos las siglas MCU para referirnos a un microcontrolador.

1



2

3

```
void setup() {
  // put your setup code here, to run once:
}
void loop() {
  // put your main code here, to run repeatedly:
}
```

4

MCU

IMPORTANTE

¿Control de procesos?

Qué complejo parece esto... Vamos a buscar en el diccionario de la RAE qué significa «control» y «proceso».

- Control: «Regulación manual o automática sobre un sistema».
- Proceso: «Conjunto de fases sucesivas de un fenómeno natural o de una operación artificial».

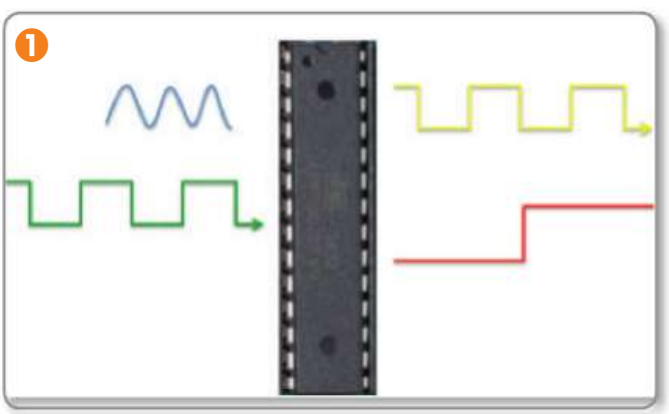
Entonces el control de procesos sería la regulación manual o automática sobre un conjunto de fases sucesivas de un fenómeno natural o de una operación artificial.

Un MCU es un circuito integrado que puede ser reprogramado y que está diseñado para el control de procesos mediante la lectura y generación de señales 1.

Para entender mejor qué es Arduino y un MCU tratemos de implementar una solución para el control del nivel de líquido de un tanque. No obstante, vamos a considerar que no disponemos de una plataforma como Arduino o similar ni conocimientos previos sobre programación, electrónica, ni MCU.

Como no disponemos de Arduino, necesitamos un MCU. Primeramente, tenemos que escoger el más adecuado, considerando las características técnicas del proceso que vamos a controlar. En el mercado disponemos de varios fabricantes de MCU, todos ellos con un amplio catálogo. Tendremos que decantarnos por un fabricante y escoger un MCU de entre todos los que tiene; para analizar los MCU, necesitamos trabajar con sus manuales.

El manual de un MCU no está desarrollado para que cualquier persona, independientemente de su formación, pueda interpretarlo. Nos enfrentaremos a manuales de hasta 450 páginas, con un lenguaje muy técnico, por lo que no solo es necesario un conocimiento previo sobre MCU, sino sobre programación, muchos «mecanismos internos», como Timers, interrupciones, etc. Posiblemente, si nunca hemos programado, no entenderemos para qué sirven.



Además, debemos analizar sus limitaciones en cuanto a potencia eléctrica, y si necesita más componentes para poder operar (como osciladores, condensadores...).

También nos limitará el formato (tamaño) de cada MCU; nos encontramos con muchos que no vamos a poder «manejar» por su reducido tamaño. Aun con todas estas dificultades, vamos a suponer que nuestra elección es la correcta.

A continuación, el siguiente problema sería cómo conectar el MCU al PC para poder programarlo: tendremos que comprar un módulo para poder programarlo, lo que nos va a obligar a realizar diferentes conexiones (implicará protoboards, cables, etc.). A todo esto quizás le debamos sumar más componentes que necesite el MCU para poder operar. Sin embargo, podemos con todo, solucionamos todos estos pasos y ya estamos listos para programar. Necesitamos un entorno de programación: lo más seguro es que el fabricante del MCU disponga de uno propio, pero quizás ofrece diferentes versiones (de prueba, para estudiantes, para profesionales) y podrán ser de pago o no.

Descargamos el entorno y ahora sí que empezamos a programar. No obstante, puede que ese entorno no sea todo lo «amigable» que esperemos. Puede ser un entorno muy completo con una gran cantidad de herramientas, menús... Lo cual es bueno, pero quizás, al principio, nos llegue a confundir, aunque al final, después de revisar documentación, lo entendamos y podamos seguir.

Si no tenemos conocimientos de programación, no nos va a quedar otro remedio que aprender, tendremos que buscar información (ejemplos, documentación, manuales, etc.).

Como acabamos de ver, existe una gran cantidad de obstáculos para simplemente empezar a programar un MCU y, con todo esto, una vez que los superemos, tenemos que ser capaces de llegar a una solución, desarrollar el programa adecuado e implementar físicamente este sistema (conectarlo a un entorno).

IMPORTANTE

Sin una formación adecuada, lo más seguro es que no nos veamos capaces de afrontar este reto, pero con Arduino esto cambia. Arduino nos elimina toda una serie de obstáculos para que nuestra principal preocupación sea programar un MCU y conectar el Arduino a los elementos que intervengan en un proceso.

A continuación veremos cómo se nos platea la misma tarea con un Arduino y veremos qué papel desempeña cada uno de los elementos de la plataforma Arduino.

Hardware Arduino

IMPORTANTE

Es importante recalcar que Arduino no fabrica los MCU de su hardware; estos son desarrollados y fabricados por Atmel.

Arduino incorpora los MCU en el proceso de fabrcación de las placas. En consecuencia, lo que hicieron fue estudiar ese manual de 450 páginas de cada MCU y dar una solución a ese problema, ofreciéndonos un sistema que incorpora todo la electrónica necesaria para programar e implementar un MCU.

Planteémonos el mismo caso del capítulo anterior, pero ahora, con un Arduino. Primeramente, al igual que pasaba con el MCU, tendremos que elegir un Arduino. Disponemos de un gran catálogo de placas Arduino, todas las placas presentan diferencias entre ellas (debido a la incorporación de periféricos como: WIFI, Ethernet, conector tarjetas SD, etc.), aunque las diferencias básicas son debidas al MCU en concreto que incorpore cada placa.

Gracias a este hardware de Arduino disponemos de:

- Conexión para PC
- La electrónica necesaria para el correcto funcionamiento del MCU
- Puntos de conexión para las patillas del MCU
- Alimentación externa (pila, batería, cargador)

Al comprar un Arduino no tenemos que preocuparnos por cómo conectarlo a nuestro PC, solo habrá que comprarlo y, con un cable USB (del tipo correspondiente según el modelo), conectarlo y empezar a programar (la mayoría de los Arduinos aportan esta solución) sin preocuparnos de si necesitamos componentes electrónicos adicionales, como ocurría cuando trabajábamos directamente con el MCU.

Además, si necesitamos conectar sensores resulta más sencillo realizar la conexión a través de los pines del Arduino que a través de las patillas de MCU.

También añadieron un conector Jack (con la electrónica correspondiente) para poder alimentar nuestro Arduino a través de pilas o baterías (no todos lo incluyen). Como disponemos de un gran catálogo de Arduino, revisaremos las placas más conocidas. Primero analizaremos el Arduino UNO y compararemos las restantes placas con respecto a este Arduino mediante un análisis básico.

Partimos entonces del Arduino UNO, ¿por qué el Arduino UNO? Porque es el único que permite reemplazar su MCU, gracias a que se encuentra insertado en un zócalo, el cual sí que está soldado a la placa y no el MCU [1].

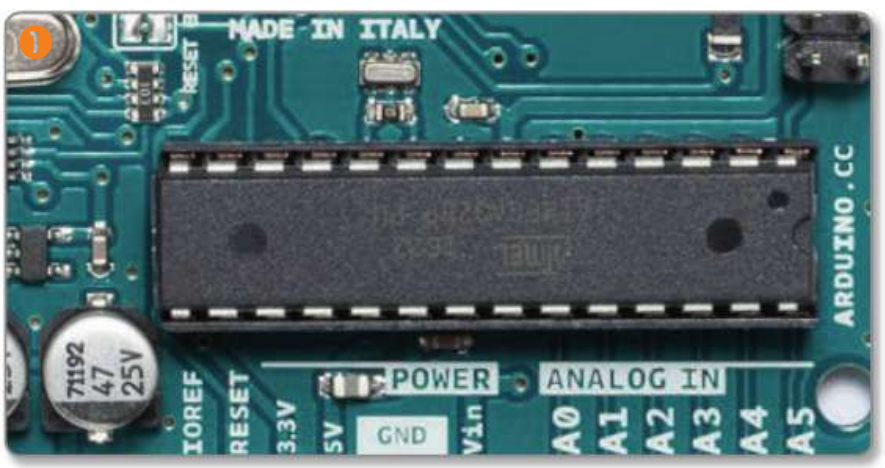
El resto de Arduinos tienen directamente soldado su MCU a la placa; el no estar soldado a la placa presenta unas grandes ventajas, como la posibilidad de programar un MCU, quitarlo de la placa e insertarlo en otro sistema electrónico.

Asimismo, lo hace ideal para principiantes (y no tan principiantes). En cualquier momento podemos cometer un error y dañar nuestro Arduino y podrían darse varias posibilidades: dañar simplemente el MCU, la placa, o ambos. En los dos últimos casos, no tendríamos más remedio que reemplazar nuestro Arduino. No obstante, en el primer caso, tenemos la posibilidad de reemplazar el MCU, podemos adquirir el MCU por separado y reemplazarlo. Aunque este paso no sería tan sencillo.

IMPORTANTE

Como comentábamos antes, Arduino no fabrica los MCU, sino que los adquiere y los implementa en el proceso de fabricación de las placas; es necesario que le carguen un programa (bootloader) para que pueda trabajar ese MCU en la plataforma Arduino.

Al comprar por nuestra cuenta el MCU, no vendrá con ese programa cargado y será trabajo nuestro realizar ese proceso.



Análisis de un Arduino

IMPORTANTE

Gracias a la web oficial de Arduino he podido solventar innumerables dudas, sobre todo de programación.

En su página principal dispone de un buscador en su margen superior derecho, en el cual podemos incluir instrucciones las cuales no entendamos bien su funcionamiento, nos direccionará a otra ventana que de forma clara reducida nos explicará su funcionamiento incluso a través de algún caso práctico.

Sea cual sea el uso que le queramos dar a un Arduino una parte importante (y la primera) es analizar el uso que le vamos a dar, con el cual determinar los requerimientos que debe cumplir el Arduino.

Con este paso previo realizado, seleccionaremos el Arduino adecuado, toda la información la tenemos disponible en la web oficial de Arduino (https://www.arduino.cc/) , será importante saber manejarnos en su web puesto que puede que la necesitemos para consultar información como:

1. Características técnicas de los productos Arduino
2. Descargar el software de Arduino
3. Resolver dudas sobre programación
4. Incluso compra de Arduino oficiales

Si nos dirigimos al apartado de products>arduino , visualizaremos una tabla con todos los productos de Arduino y, si seleccionamos el que nos interesa (en nuestro caso UNO), accederemos a su información técnica que se nos presentará en tres apartados:

1. Overview: descripción breve de las características básicas de producto seleccionado.
2. Tech Specs: tabla con las principales características del producto.
3. Documentation: amplía las características vistas en la tabla anterior y profundiza en algunas de ellas.

En el siguiente capítulo analizaremos la tabla ❶ con las principales características del Arduino UNO, puesto que para empezar a trabajar con él será más que suficiente.

1

OVERVIEW | TECH SPECS | DOCUMENTATION

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

IMPORTANTE

A la vista salta la gran diferencia de estudiar las características de un Arduino, para lo cual debemos analizar una tabla frente a un MCU con un manual extenso.

Es de agradecer esta simpleza, en los manuales se profundiza sobre la arquitectura interna de un MCU, que es importante pero para un principiante irrelevante y lioso.

Arduino UNO

IMPORTANTE

Al ser el voltaje de operación de 5V, no podemos interactuar de forma directa con muchos de los elementos de nuestro día a día como puede ser una bombilla. Al principio puede suponer algo confuso que el voltaje de trabajo sea de 5V y que podamos alimentar nuestro Arduino a través del conector Jack con una tensión de 7 a 12V (con un límite de 6 a 20V). El hardware del Arduino dispone de una electrónica que reduce la tensión de entrada a 5V para poder alimentar su MCU. Uno de los parámetros más importantes es la limitación de intensidad (20mA), el no cumplir esta especificación junto con la limitación de tensión son las principales causas de daño.

Debemos tener en cuenta que la limitación de 20mA no significa que por un pin no puedan salir más de esos Amperios, sino que no debe de salir más. Un Arduino no produce potencia eléctrica, lo que hace es "distribuir" esa potencia de la fuente de alimentación, por lo tanto si le demandamos 100mA por uno de sus pines, el Arduino lo que hará será "pillar" esa

A continuación se revisarán las principales características del Arduino UNO (recogidas en la tabla de Tech Specs).

1. **Microcontroller**-Atmega328P: nos indica el MCU que incorpora y, si hacemos click en esta referencia, podemos acceder a su manual.
2. **Operating Voltage**-5V: tensión máxima con la que puede trabajar el MCU del Arduino, por lo tanto es la tensión máxima que disponemos para interactuar con el entorno (encender luces, comprobar pulsadores).
3. I**nput Voltage (recommended)**-7-12V: como ya se mencionó, en la mayoría de las placas de Arduino han incorporado un conector Jack para alimentar un Arduino a través de una pila, batería o cargador. Esta solución no era imprescindible pero si resulta cómoda, para implementar soluciones con Arduino. Tal y como nos indican la tensión recomendada sería entre 7 y 12 V.
4. **Input Voltage (limit)**- 6-20V: estos límites nos definen la tensión máxima y mínima a la que podemos alimentar un Arduino, debemos alejarnos de estos valores para evitar malos funcionamientos o calentamiento de la placa.
5. **Digital I/O Pins** -14 (of which 6 provide PWM output): el Arduino UNO dispone de 14 pines digitales que se pueden comportar como entradas o salidas, por ser digitales los pines pueden estar "encendidos" o "apagados". Esto, junto con el hecho de que puedan ser entradas o salidas, nos permitirá encender o apagar un led (en caso de estar configurado como salida) o saber si un pulsador se ha pulsado (en caso de estar configurado como entrada). Además nos indican que 6 de estos 14 pines pueden generar señales PWM, lo que por ejemplo nos permite regular un motor o una luz. Profundizaremos en esto con casos prácticos.
6. **PWM Digital I/O Pins**-6: hace referencia a los pines que generan señales PWM del apartado anterior.
7. **Analog Input Pins**-6: estos pines nos permiten tomar lecturas de señales que varian entre 0 y 5V. Como puede ser el caso de sensores que devuelven un valor de tensión en función de la cantidad de magnitud física medida.
8. **DC Current per I/O Pin**-20mA: limitación de intensidad que puede entrar o salir por cada uno de los pines del Arduino.
9. **DC Current for 3.3V Pin**-50mA: como veremos, todos

los pines de un Arduino aparecen referenciados por una etiqueta, que nos indicará su función. En la parte de potencia del hardware del Arduino UNO tenemos un pin de 3,3V que ofrece esa tensión de forma constante y a diferencia del resto de pines permite "sacar" hasta 50mA.

10. **Flash Memory**-32 KB (ATmega328P) of which 0.5 KB used by bootloader: posiblemente en cuanto a las características que ofrece esta tabla las memorias son las más difícil de evaluar o comparar con otros Arduinos. La memoria FLASH es en la que se almacena el programa que carguemos en nuestro Arduino, que en el caso del Arduino UNO tiene un límite de grabaciones de 10.000. Vemos que nos indican que la memoria tiene una capacidad de 32KB pero que 0.5KB de los 32 ya vienen ocupados por algo que llaman bootloader. Esto es un gestor de arranque que permite que los MCU que incorporan los Arduinos funcionen en la plataforma Arduino, no es más que un programa que instalan en la fabricación del Arduino y que si nosotros compramos el MCU por separado no lo tendrá grabado y deberemos ser nosotros el que se lo carguemos.
11. **SRAM**-2 KB (ATmega328P): esta memoria es la que utiliza el MCU para trabajar con los datos temporales, es decir, los que necesita en cada momento para realizar las operaciones que hubiésemos programado. Estos datos se pierden al apagarse el Arduino. Esta memoria no tienen límite de escrituras(tampoco es que la utilicemos directamente lo podemos considerar como un mecanismo interno para que pueda funcionar el Arduino).
12. **EEPROM**-1 KB (ATmega328P): esta memoria permite almacenar datos y aunque se apague el Arduino podemos recuperarlos una vez se encienda, tiene un límite de 100.000 escrituras. Esta memoria si puede ser programada pero no lo haremos en este libro.
13. **Clock Speed**-16 MHz: define "la rapidez" con la que opera un Arduino y también interviene en temporizaciones que realizamos con Arduino.
14. **LED_BUILTIN**-13: este hardware dispone de un led en la placa asociado al estado del pin 13.
15. **Length**-68.6 mm: el largo del Arduino UNO.
16. **Width**-53.4 mm: el ancho del Arduino UNO.
17. **Weight**-25g: peso del Arduino UNO.

IMPORTANTE

intensidad y la sacará por el ese pin, el problema es que al pasar esa intensidad a través del MCU del Arduino este se dañará.

Las limitaciones de potencia eléctrica junto con el número de pines digitales, señales pwm y entradas analógicas son las características más básicas que nos pueden hacer elegir un Arduino frente a otro.

Si necesitamos leer 10 sensores analógicos, regular 10 leds, monitorizar 20 pulsadores el Arduino UNO no sería el ideal puesto que no dispone del hardware necesario. Para el desarrollo de los casos prácticos de este libro se ha utilizado el Arduino UNO y así se reflejará en los programas y esquemas, pero si se dispone de otro modelo de Arduino se pueden realizar todos los casos del mismo modo.

Si términos como intensidad o tensión resultan nuevos y por lo tanto confusos, antes de crear los primeros circuitos abordaremos este tema.

Comparativa Arduinos

A continuación realizaremos una comparativa del resto de Arduinos con respecto al Arduino UNO, quedándonos con las características principales.

- **Genuino 101**: Voltaje de trabajo 3,3V pero permite 5V por los pines, 4 pines PWM, dispone de acelerómetro, giróscopo y bluetooth.
- **2560**: 54 pines I/O digitales, 15 pines PWM, 16 entradas analógicas.
- **Micro**: 20 pines I/O digitales, 7 pines PWM, 12 entradas analógicas.
- **MKR1000**: voltaje de trabajo 3,3V ,8 pines I/O digitales, 12 pines PWM, 7 entradas analógicas, 1 salida analógica, conexión por WIFI y 7mA por pin.
- **PRO**: voltaje de trabajo 3,3V o 5V y 40mA de limitación por pin.
- **PRO MINI**: voltaje de trabajo 3,3V o 5V 40mA de limitación por pin, no incluye programador.
- **Genuino ZERO**: voltaje de trabajo 3,3V ,20 pines I/O digitales, 10 pines PWM,1 salida analógica y 7mA de limitación por pin.
- **DUE**: voltaje de trabajo 3,3V ,54 pines I/O digitales, 12 pines PWM, 12 entradas analógicas, 2 salida analógica, 130mA de limitación por todos los pines, salvo 800mA por 3,3V y 5V(cada uno).
- **ETHERNET**: 14 pines I/O digitales, 4 pines PWM, 6 entradas analógicas, 40mA de limitación por pin ,conexión a internet a través de cable de red y posibilidad de conectar una tarjeta microSD.
- **LEONARDO** 10: 20 pines I/O digitales, 7 pines PWM, 12 entradas analógicas y 40mA de limitación por pin.
- **MEGA ADK**: 54 pines I/O digitales, 15 pines PWM, 16 entradas analógicas, 40mA de limitación por pin un conector USB para comunicación con dispositivos Android.
- **MINI**:8 entradas analógicas y hasta 40 mA por pin.
- **NANO**: 22 pines I/O digitales, 12 pines PWM, 8 entradas analógicas y hasta 40mA de limitación por pin.
- **YÚN**: 20 pines I/O digitales, 7 pines PWM, 12 entradas analógicas, hasta 40mA de limitación por pin y conexión a intenert por WIFI o cable.
- Si abrimos la web de Arduino ❶ y accedemos al apartado de products ❷ encontraremos una tabla ❸ en donde podemos revisar más en detenimiento cada una de las características de los dispositivos mencionados en los puntos anteriores.


Es seguro https://www.arduino.cc
ARDUINO
CASA COMPRAR SOFTWARE PRODUCTOS EDUCACIÓN RECURSOS COMUNIDAD AYUDA
REGISTRARSE
1
¿QUE ES ARDUINO?
COMPRE UN ARDUINO
LEARN ARDUINO
DONAR
BLOG
LA EXPANSIÓN DE LA LÁMPARA DE LA ESTRELLA DE LA MUERTE CON MKR1000
ARDUINO. DAY 2018
JOIN THE OFFICIAL EVENT LIVE FROM MILAN
2
SOFTWARE
PRODUCTOS
EDUCACIÓN
ARDUINO
EN EL CORAZÓN
NIVEL DE ENTRADA
UNO
LEONARDO
101
ESPLORA
MICRO
NANO
MINI
ADAPTADOR MKR
KIT DE INICIO
PANTALLA LCD
3
CARACTERÍSTICAS MEJORADAS
MEGA
CERO
DEBIDO
MEGA ADK
M0
M0 PRO
MKR ZERO
USB HOST SHIELD
PROTO SHIELD
MKR PROTO SHIELD
4 RELÉS SHIELD
MEGA PROTO SHIELD
MKR RELAY PROTO SHIELD
ISP
USB2SERIAL MICRO
CONVERTIDOR USB2SERIAL
INTERNET DE LAS COSAS
YÚN
ETHERNET
TIAN
INDUSTRIAL 101
LEONARDO ETH
MKR FOX 1200

Hardware libre

IMPORTANTE

El hecho de que el hardware sea libre no significa que sea gratuito, lo cual resulta lógico puesto que su fabricación conlleva unos gastos.

En la definición de Arduino nos referíamos a él como una plataforma libre, por ende, el hardware de Arduino por formar parte de la plataforma de Arduino debe ser libre. Volviendo a la web de Arduino y a los detalles técnicos de este hardware, si nos dirigimos al apartado de documentation 1, encontraremos una serie de archivos disponibles para su descarga (son archivos con información sobre el diseño del hardware).

El hardware libre o "open source" (fuente abierta) ofrece una serie de libertades en cuanto a su diseño:

- Estudiarlo.
- Modificarlo.
- Reutilizarlo.

Comparte ciertas similitudes con el software libre, que tiene más antecedentes que el hardware libre.

Para que todo esto sea posible los usuarios deben tener acceso a los archivos de diseño, bajo una licencia que permita cada una de las libertades anteriormente mencionadas.

La licencia del hardware de Arduino es Creative Commons Attribution Share-Alike. Esta licencia, entre otros derechos, otorga la libertad de comercialización, siempre y cuando el que lo quiera hacer lo ponga en conocimiento de Arduino y publique sus diseños bajo la misma licencia.

Por lo tanto, nos podemos encontrar en el mercado con placas

oficiales y no oficiales; esto permite que podamos comprar placas de Arduino por unos pocos euros (de hecho ese era uno de los objetivos de los fundadores de Arduino).

En el apartado anterior pudimos ver cómo existen grandes diferencias entre ciertos Arduinos: las más básicas son las de potencia eléctrica y número de entradas y salidas.

Pero existen otras como la implementación de periféricos (Ethernet, WIFI, SD...). El hardware Arduino es modular, lo que quiere decir que si el modelo del que partimos no dispone de ciertos periféricos podemos añadir módulos para ampliar sus funcionalidades.

Estos módulos son más conocidos como Shields (escudos). Disponemos de todo tipo de Shields: GSM, WIFI, ETHERNET, SD, potencia, RFID, pantallas táctiles. etc. A todo esto podemos añadir una gran cantidad de sensores. Por lo tanto, Arduino nos permite interactuar con la tecnología de otra forma, ya no como usuarios sino como desarrolladores, y por lo tanto entender mejor el mundo tecnológico que nos rodea.

No cabe duda de que todos estos recursos, unidos al bajo coste, convierten a Arduino en la herramienta ideal para el aprendizaje teniendo cada vez mayor implantación en las aulas [2]. El hardware oficial de Arduino está certificado por la Comisión Federal de Comunicaciones estadounidense y por la europea, cumpliendo con las normativas referentes a emisiones electromagnéticas.

IMPORTANTE

El hecho de que Arduino cumpla las normativas electromagnéticas nos asegura que las señales que se generan en los dispositivos electrónicos no generen interferencias entre ellas, entre esas señales podemos encontrar:

WIFI

Bluetooth

Telefonía

etc.

[2] STEM

Software Arduino

1. Una vez que disponemos de nuestro Arduino lo conectamos a nuestro PC y empezamos a desarrollar nuestro programa. Para ello necesitamos un entorno de programación. Arduino dispone de un entorno de programación integrado conocido como IDE de Arduino. A partir de ahora nos referiremos a este entorno de programación como IDE de Arduino[1].

2. El IDE de Arduino es un entorno multiplataforma: lo podemos instalar en cualquier sistema informático (Windows, Mac Os o Linux).

3. Este entorno nos permite escribir un programa, verificarlo (comprobar errores) y cargarlo a nuestro Arduino. Dispone de una herramienta como es el Monitor Serie que permite recibir o enviar información a través del puerto USB de nuestro PC. Si, por ejemplo, programamos nuestro Arduino para que, cuando reciba un mensaje por el puerto USB que ponga «encender leds», debe encender los leds que tenga conectados a sus salidas, a través del Monitor Serie, podemos enviar el mensaje «encender leds». Entonces, si programamos adecuadamente el Arduino, podemos controlarlo a través del IDE.

4. El Monitor Serie también permite mostrar información que reciba por el puerto USB del PC. Si, por ejemplo, programamos nuestro Arduino para que tome las lecturas de unos sensores a los cuales esté conectado, y que envíe esos valores por el puerto USB, podemos visualizarlos en el IDE de Arduino. Por lo tanto, también podemos decir que con el IDE podemos monitorizar un Arduino. Y por supuesto, con la programación adecuada podemos controlar y monitorizar nuestro Arduino simultáneamente.

5. Disponemos de un creador de gráficas que nos permite representar datos que recibamos por el puerto USB de nuestro PC. También disponemos de ejemplos y librerías para que nos sea más sencillo aprender a programar. Las librerías que tiene instaladas son las oficiales.

¿Un Arduino solo se puede programar con el IDE de Arduino? ¿El IDE de Arduino solo puede programar Arduinos? No, podemos programar un Arduino con otro entorno de programación siempre y cuando sean compatibles. Si tenemos práctica con un entorno de programación determinado podemos comprobar si es posible programar con ese software nuestro Arduino, seguramente necesitemos instalar alguna extensión. Y con el IDE de Arduino pasa lo mismo, con las extensiones correctas podemos programar otros módulos.

En conclusión, Arduino pone a nuestra disposición un entorno sencillo y gratuito que nos permitirá de una forma rápida empezar a crear nuevos proyectos.

008

IMPORTANTE

¿Qué es una librería?

El término "librería" hace referencia a un programa que tenemos o podemos tener instalado en nuestro IDE, que podemos usar en cualquier momento.

Que incorpora funciones que nos facilitan la programación y que simplemente haciendo referencia al inicio del programa del uso de una determinada librería, adquirimos los permisos para usar sus funciones.

Llegado el momento veremos como utilizar las funciones de una librería

Lenguaje de programación

IMPORTANTE

El lenguaje máquina nos obliga a conocer la estructura y funcionamiento interno de un MCU [1], lo cual suele ser complejo , pero nos permite optimizar nuestro programa.

Por su importancia, a lo largo del libro haremos algunos casos con este tipo de lenguaje, para disponer de una báse sólida en cuanto a programación de MCU.

Con nuestro Arduino conectado a nuestro ordenador y el IDE instalado, solo nos falta empezar a programar. En cuanto al lenguaje de programación (idioma artificial diseñado para expresar instrucciones que pueden ser llevadas a cabo por una máquina), podemos diferenciar varios niveles. El lenguaje a más bajo nivel sería el lenguaje máquina: es el lenguaje capaz de almacenar e interpretar una máquina, en nuestro caso, un MCU.

En los inicios de los MCU, estos se programaban en lenguaje máquina (a día de hoy también se hace). El lenguaje máquina consiste en instrucciones a nivel binario, lo que nos obliga a encadenar una cierta cantidad de ellas para poder realizar tareas sencillas. Además, cada máquina puede tener su propio lenguaje.

Con este lenguaje de programación surgen ciertos inconvenientes: primeramente, un mismo programador debe aprender un lenguaje diferente para cada máquina. Por lo general, no son lenguajes intuitivos (lo que los hace difíciles de recordar) y programas sencillos pueden llegar a alcanzar un gran tamaño.

Un programa desarrollado en lenguaje máquina se convierte en difícil de interpretar y modificar. Viendo todo esto, se empezó a evolucionar en la programación hasta llegar a lo que conocemos a día de hoy como lenguaje a alto nivel o lenguaje estructurado.

Existen varios lenguajes a alto nivel: C, Python, C++, Java... El lenguaje a alto nivel está formado por instrucciones más complejas que el lenguaje máquina, lo que significa que una instrucción en este lenguaje puede equivaler a varias en lenguaje máquina.

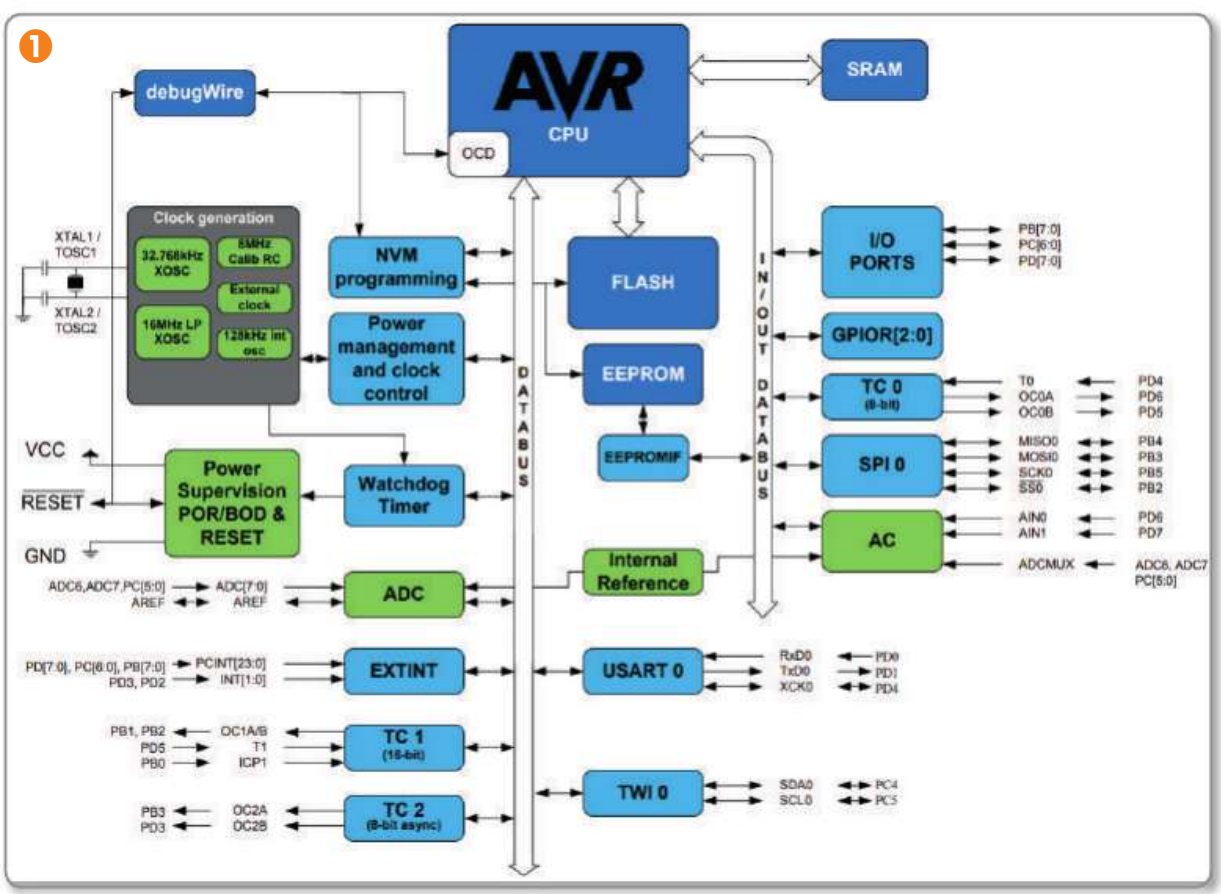
Con esto conseguimos programas más cortos, más fáciles de interpretar y modificar. El lenguaje a alto nivel que emplea el IDE de Arduino se basa en el lenguaje C, aunque es cierto que la mayoría de los lenguajes comparten estructuras o funciones similares, como pueden ser los bloques de control de flujo (if, esle, while). Aparte del lenguaje compartido con C, dispone de instrucciones propias para un Arduino.

Independientemente de si programamos en lenguaje estructurado o máquina, la carga de un programa al MCU del Arduino conlleva una serie de pasos que se conoce como compilación. La compilación se puede considerar como una traducción del programa que hemos desarrollado a un lenguaje que pueda almacenar e interpretar el MCU.

El proceso de compilación es complejo, puesto intervienen varios pasos:

- Precompilador
- Compilador
- Linker
- Carga de programa

Por la complejidad de cada uno de estos procesos no entraremos a estudiarlos, pero debemos recalcar que hoy en día podemos programar una máquina en lenguaje estructurado porque los entornos de programación han evolucionado, introduciendo una serie de procesos para la traducción y carga de un programa.



Historia y filosofía de Arduino

Arduino nace en el año 2005 en el Instituto Interactivo Ivrea (Italia). En este centro se dedicaban al estudio de la forma, en cómo interactúa el ser humano con la tecnología.

1. La tecnología de la que disponían no era precisamente versátil, no agilizaba el aprendizaje como lo hace Arduino. A todo esto hay que añadirle el aspecto económico.
2. Decidieron empezar con el proyecto de Arduino parte del profesorado (no solo del centro) y parte del estudiantado. Al poco de empezar tenían la sospecha de que el centro iba a cerrar y, con el temor de que el proyecto de Arduino quedase en el olvido, decidieron compartir su trabajo en internet (lo que ha sido un paso importantísimo para que Arduino sea lo que es hoy en día).
3. El proyecto empezó a crecer con la colaboración de una gran cantidad de personas, y los fundadores de Arduino reconocen que es el trabajo de mucha gente. Esta es la gran ventaja de Arduino: alrededor de este proyecto se ha generado una gran comunidad muy activa que siempre está compartiendo ideas y aportando soluciones.
4. A través de la web de Arduino se crean foros sobre cualquier temática[1], no solo para aportar ideas sino para solucionar dudas que puedan tener otros usuarios. Sin embargo, no solo a través de la web de Arduino se comparte este proyecto, también existe una gran cantidad de webs, blogs, vídeos tutoriales dedicados a Arduino.
5. Los objetivos de Arduino consistían en desarrollar un sistema económico que no costase más que un libro y que hiciera que personas sin conocimientos de programación, pero con ansias de aprender y desarrollar, pudiesen tener la herramienta que necesitaban, y podemos asegurar que lo han conseguido.
6. Lo que podamos hacer con Arduino dependerá en gran medida de nuestra imaginación y capacidad. No obstante, apoyándonos en su gran comunidad, en la tecnología de la que disponemos y otras plataformas, veremos que, cada vez, tendremos menos límites.
7. Al controlar un proceso con un Arduino no solo estamos trabajando con este hardware sino que necesitamos conectarnos a sensores, módulos... Arduino presenta una gran ventaja para todas aquellas personas con inquietudes por desarrollar, aportándoles todas las facilidades que hemos visto en los capítulos anteriores. ¿Qué pasaría si no existiesen sensores o módulos fáciles de usar? Que todas las soluciones que aporta Arduino se pueden ver en cierto modo limitadas para las personas que están empezando.
8. Para una persona inexperta en programación o electrónica ¿de qué le sirve que Arduino sea relativamente sencillo de usar si, a la hora de implementarlo con otros componentes, estos son difíciles de programar o de conectar? Perdemos parte de la agilidad de Arduino.
9. Tenemos a un simple click[2] toda la información que necesitamos, desde cómo se conecta un sistema a cómo se programa.


COMMUNITY
HELP
FORUM
1
BLOG
PROJECT HUB
ARDUINO USER GROUPS
PLAYGROUND


ARDUINO
PROJECT HUB
2
ADD PROJECT
SEARCH PROJECTS
SIGN IN
ARDUINO / DISTRELEC: AUTOMATION & ROBOTICS CONTEST
MAKING, MANUFACTURING, DRIVING THE 4.0 REVOLUTION
IN PARTNERSHIP WITH DISTRELEC
ARDUINO
DISTRELEC

Intensidad y tensión

IMPORTANTE

Existen dos tipos de corrientes eléctricas, la corriente continua (DC) y la alterna (AC).

Nos centraremos en la DC que es con la que opera nuestro Arduino. La principale diferencia entre ambas es que la DC no varía con el tiempo y la AC sí.

La corriente que tenemos en nuestras casas es AC, puesto que su transporte conlleva menos pérdidas que la DC.

El primer caso que abordaremos será el simple encendido de un led a través de nuestro Arduino. Para ello necesitaremos unos conocimientos básicos sobre electrónica y algunos componentes.

En este y en próximos capítulos revisaremos los conceptos básicos sobre electrónica, circuitos y componentes electrónicos. En capítulos anteriores mencionamos los términos de intensidad y tensión, es hora de profundizar en ellos.

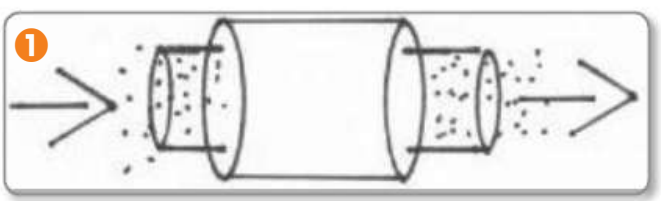
La intensidad o corriente eléctrica es el "combustible" que consumen nuestros dispositivos electrónicos para poder realizar la tarea para la cual fueron diseñados. Este combustible ❶ no es más que el movimiento de partículas (cargadas eléctricamente) a través de un material conductor (por ejemplo, el cobre de un cable).

1. La intensidad se mide en Amperios (A).
2. A la hora de circular por un material conductor tiene un sentido, que cuando definamos la tensión veremos que dependerá de esta.

Podemos imaginarnos la tensión (o diferencia de tensión) como una fuerza ,o más bien como una diferencia de fuerzas. Por ejemplo, una pila tiene dos polos el positivo (+ o Vcc) al cual si conectamos algo se suele referenciar esa unión con un cable rojo y el polo negativo (- o GND) que al contrario que el anterior si se conecta algo a este polo se hace con un cable negro ❷.

1. La tensión se mide en Voltios (V).
2. Si conectamos un componente directamente a nuestra pila, de por ejemplo 9V, podemos asegurar que la fuerza a la que

❶



❷

está sometido ese componente es de 9V [3].

3. La tensión, al igual que la intensidad, tiene un sentido, el cual se indica desde la tensión mayor hacia menor (la intensidad siempre tiene el mismo sentido que la tensión).

Las señales que tiene que gobernar el Arduino bien sea para encender un led o para tomar la lectura de un sensor, siempre vendrán determinadas por un valor de tensión e intensidad. Antes de realizar cualquier conexión debemos tener en cuenta las limitaciones de tensión e intensidad del Arduino (5V y 20mA) y también del componente o componentes que se puedan encontrar conectados a nuestro Arduino, ya que, al igual que este, se pueden dañar por un exceso de tensión o intensidad.

El primer paso que debemos realizar en nuestros proyectos es estudiar las intensidades y tensiones que pueden circular por nuestros circuitos. Lo próximo que trataremos en los siguientes temas será:

- Cuantificación de tensión e intensidad (ley de ohm).
- Análisis de circuitos.
- Primeros componentes de interés : resistencias y leds.
- Multímetro, dispositivo que nos permitirá medir en casos reales el valor de tensión e intensidad de nuestro circuitos.

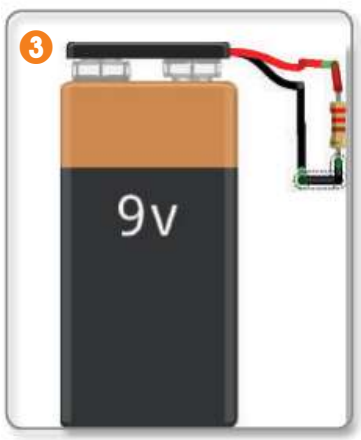
Después de revisar estos puntos ya dispondremos de todos los conocimientos necesarios para afrontar el primer caso de estudio (encendido de un led con Arduino).

IMPORTANTE

Al igual que la intensidad, la tensión puede ser continua o alterna (el Arduino trabaja en continua).

Si un componente o componentes no se encuentran sometidos a una diferencia de tensión, no circulará intensidad a través de ellos.

Debemos de tener la precaución de no conectar directamente el polo positivo y el negativo de un pila para evitar un cortocircuito, [4] lo que dañaría la pila (o fuente de alimentación) y posiblemente el circuito electrónico.



Resistencia

IMPORTANTE

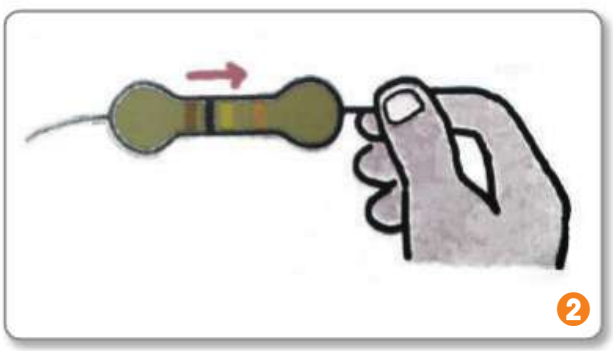
Como ya mencionamos, las resistencias no tienen polaridad y por lo tanto nos da igual cómo se conecten en un circuito, pero el resto de componentes que estudiemos sí tienen polaridad, por lo que será importante dónde se conecta cada patilla.

Las resistencias son los elementos más comunes que nos podemos encontrar en un circuito. Las resistencias permiten que la intensidad circule a través de ellas, y su principal función es limitar la cantidad de intensidad que circula por un circuito electrónico.

Cuanto mayor sea la resistencia que conectemos, menor será la intensidad que circule por ella. Una resistencia se puede considerar como un fuerza que se opone al paso de la corriente eléctrica y cuanto mayor sea el valor de esa fuerza (es decir de la resistencia) menos corriente circulará.

Sabiendo ya lo que es una resistencia, podemos adelantar que el uso que le daremos será para la protección de componentes o incluso del propio Arduino (si recordamos existen ciertas limitaciones de intensidad en los pines del Arduino, que en algunos casos nos aseguraremos de que se cumplen mediante la conexión de resistencias).

1. La unidad de medida de las resistencias son los ohmios.
2. Los ohmios se representan por la letra griega Ω.
3. Las resistencias constan de dos patillas y no tienen polaridad, lo que quiere decir que cuando conectemos una resistencia siempre necesitaremos dos puntos de conexión (uno por cada patilla), pero no importará en qué punto de conexión conectemos cada patilla, el funcionamiento será el mismo.
4. Para saber el valor de una resistencia tenemos dos opciones: o bien con un multímetro ❶ o mediante su banda de colores ❷ .



5. Con el multímetro simplemente giraremos el selector hasta el indicativo Ω y conectaremos los terminales a los extremos de las patillas (indistintamente puesto no tiene polaridad).
6. En caso de cuantificar la resistencia por su banda de colores, de todos los colores de la banda hay uno que nos indica cómo se debe leer la banda, por lo general de color dorado.
7. El color dorado nos indica que debemos empezar a codificar los valores empezando por el otro extremo, recorriendo todos los colores hasta volver a la banda dorada (y esta última no se cuantifica).
8. Iremos codificando los colores conforme a su valor correspondiente en la tabla 3 .
9. El último valor nos indicará la potencia de 10 por la que hay que multiplicar el código obtenido. Así obtenemos el valor de la resistencia.
10. La resistencia dispone de un símbolo genérico para su representación en un esquemático 4 .

Por último, si vamos a una tienda y pedimos una resistencia nos preguntarán dos datos: valor y potencia. El valor de la resistencia se refiere al tamaño de esta, es decir, a los ohmios. En cuanto a la potencia se refiere a cuánta soporta, que viene de la multiplicación de la intensidad que atraviesa la resistencia por la tensión a la que se encuentra sometida.

En el próximo capítulo calcularemos la intensidad que pasa por una resistencia y también la potencia disipada.

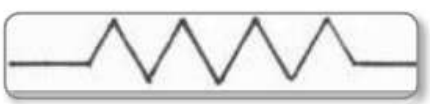
IMPORTANTE

Un multímetro es un dispositivo que nos permite cuantificar magnitudes físicas como resistencias, intensidades, voltajes...

Color	Valor
Negro	0
Marrón	1
Rojo	2
Naranja	3
Amarillo	4
Verde	5
Azul	6
Morado	7
Gris	8
Blanco	9

3

4



Ley de Ohm

IMPORTANTE

Una vez conocidos los valores de tensión e intensidad que circularán por una resistencia, debemos escoger el valor de potencia adecuado para que no se queme.

La potencia viene definida por la fórmula P=I*V, en nuestro caso la potencia sería de: P=0,009*9=0,081W (vatios). La resistencia que compremos debe soportar una potencia superior a 0,081W.

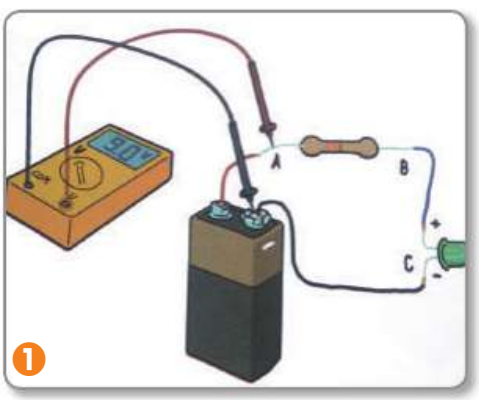
La Ley de Ohm es la ley básica de la electrónica, que define que la tensión a la que se somete una resistencia es directamente proporcional al valor de la resistencia multiplicado por la intensidad que circula a través de ella.

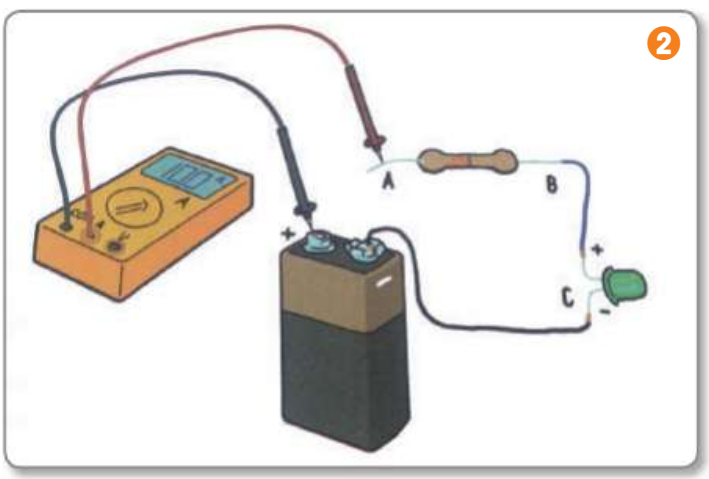
$$V=R*I$$

1. Si conectamos la resistencia a la pila (cada patilla en un polo diferente), la tensión a la que se encuentra sometida la resistencia es igual a la diferencia de tensión que aporta la pila.
2. Esto lo podemos comprobar con el multímetro: giramos el selector hasta el símbolo V y nos debemos asegurar que la tensión que mide es continua (DC). Si no fuese así pulsaríamos el botón de select y ya debería de quedar configurado para medir tensiones en DC ❶.
3. De la fórmula de la Ley de Ohm conocemos dos valores: la resistencia (en este caso será de 1000 ohmios) y la tensión (9V), por lo que podemos despejar la intensidad:

$$I=V/R=9/1000=0,009A=9mA$$

4. Esta intensidad podemos comprobarla con el multímetro; como la intensidad es del orden de miliamperios, giramos el selector hasta el símbolo mA (nos tenemos que asegurar que se encuentre en DC) ❷.



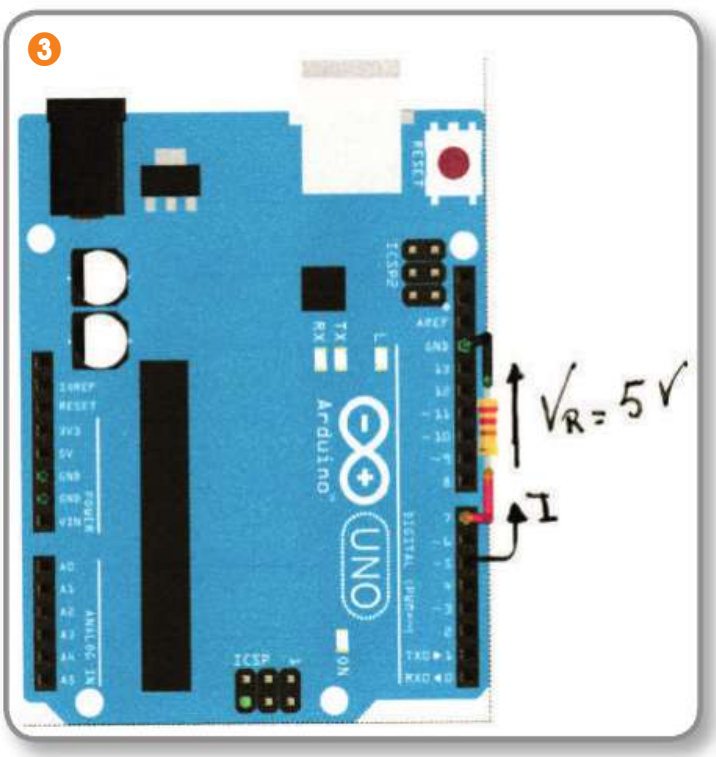


5. Es necesario cambiar el conexionado de los cables del multímetro, puesto que ahora vamos a medir intensidad y para ello en mi caso dispongo de dos puntos de conexión dependiendo del nivel de intensidad debemos hacer la conexión correspondiente. En este caso es del orden de mA por lo tanto lo conectaremos en el que tiene la referencia de mA.
6. Debemos tener cuidado a la hora de tomar mediciones de intensidad, para no puentear ningún elemento y tomar la medida correcta.

Con la ley de Ohm podemos calcular el valor de resistencia mínima para asegurarnos de que en ningún momento se supera la demanda de 20mA por cada pin (sabemos por la tabla de características que la tensión que puede aportar cada pin es de 5V).

$$R=V/I=5/0{,}020=250\Omega$$

Con un valor de resistencia mínimo de 250Ω es seguro que conectemos lo que conectemos a continuación de ella (leds, pulsadores, ..) no se dañará el Arduino por un exceso de consumo de intensidad, pero esto no quiere decir que si por ejemplo también conectamos un led ,este no se pueda dañar .También debemos tener en cuenta las limitaciones de los componentes que se vayan a conectar [3] .



Diodo led

IMPORTANTE

Disponen de dos patillas y, a diferencia de las resistencias, sí tienen polarización, esto nos indica que deberemos prestar atención a cómo conectamos un diodo en el circuito.

Un diodo es un componente electrónico que, por sus características de funcionamiento, en según qué condiciones permite o no la circulación de corriente.

Los leds son un tipo de diodo que, al circular corriente a través de él ❶ emiten luz; cuanta más corriente, más luz emiten. Pero como todo componente, tienen unas características que debemos evaluar para no dañarlo.

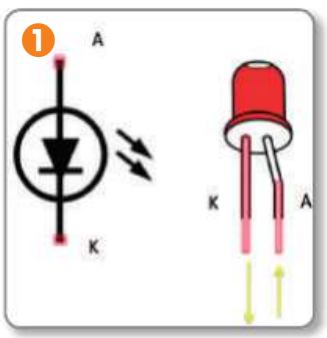
Un diodo, ya sea un diodo o diodo led, tiene dos modos de funcionamiento:

- Polarización inversa ❷
- Polarización directa ❸

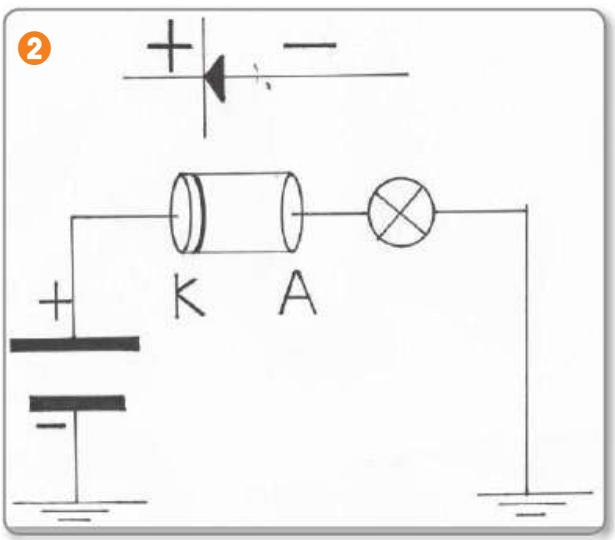
Si se encuentra polarizado inversamente no permite la circulación de corriente, sin embargo con una polarización directa sí circula corriente. Podemos considerar que un diodo polarizado en inversa se comporta como un interruptor abierto, y en directa en un interruptor cerrado.

¿Cuándo se encuentra polarizado en directa y en inversa? Si revisamos un diodo led nos encontraremos con una patilla más grande que otra. A la grande se le denomina ánodo y a la pequeña cátodo. Para que un diodo se encuentre polarizado en directa la intensidad que circula a través de él debe entrar por el ánodo y salir por el cátodo.

❶



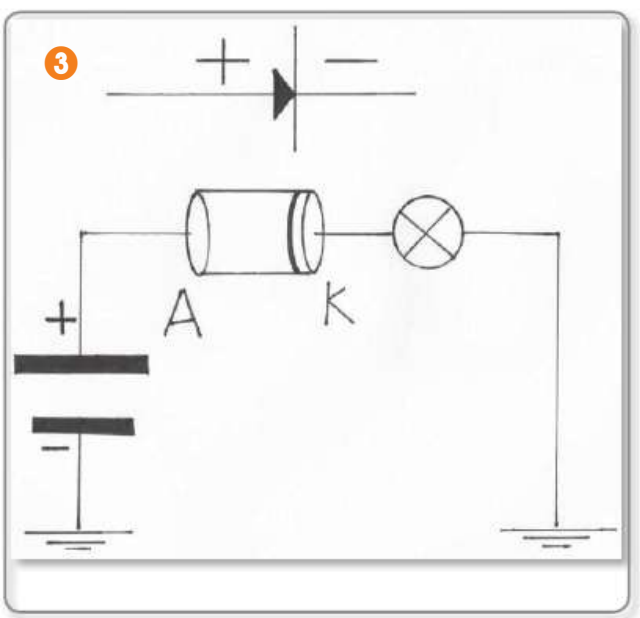
❷

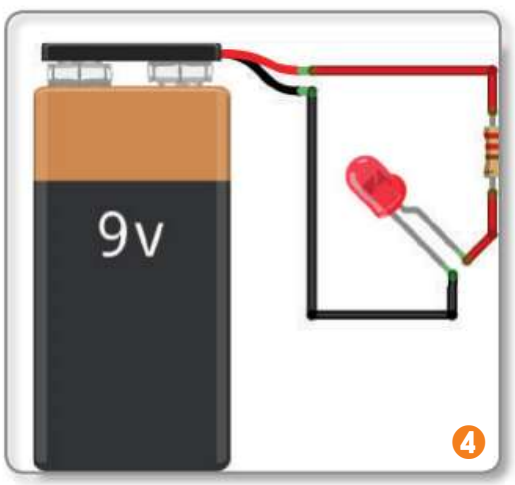


1. Para que un led se encienda debe estar polarizado en directa y además debemos analizar las características de tensión e intensidad. No podemos someter un led a cualquier diferencia de tensión ni a cualquier circulación de intensidad puesto que lo podemos quemar.
2. Por lo general, los leds más comunes demandan una tensión de funcionamiento de alrededor de 1,8V (incluso de 2V). Pero también es una tensión de limitación, no podemos superar ese valor de tensión, de ser así se quemaría.
3. Al principio explicábamos que, cuanta más intensidad pase por un diodo led, más luz emitirá; pero al igual que ocurre con la tensión, esta intensidad debe encontrarse entre un rango de valores. La intensidad de los diodos más comunes suele oscilar entre 5 y 20mA. Con todo esto, lo primero que advertimos es que no podemos conectar un led directamente a una pila o al Arduino, puesto que por el valor de la tensión (9V y 5V respectivamente) se quemaría.
4. Lo que haremos será conectar una resistencia para limitar el consumo de intensidad y para que en la resistencia caiga la tensión necesaria y así no dañar al led 4 .

IMPORTANTE

En el próximo capítulo calcularemos la resistencia que debemos conectar, pero antes revisaremos las leyes de Kirchhoff, que nos permitirán analizar circuitos. Los diodos son componentes ampliamente utilizados en electrónica, por eso según avancemos en los capítulos los estudiaremos más a fondo.





Leyes de Kirchhoff

IMPORTANTE

Las leyes de Kirchhoff nos permitirán conocer cómo se comportan la corriente y tensión a lo largo de un circuito, así como las dos reglas básicas del análisis de circuitos.

Un nodo es todo aquel punto del circuito en el cual se unen dos o más cables.

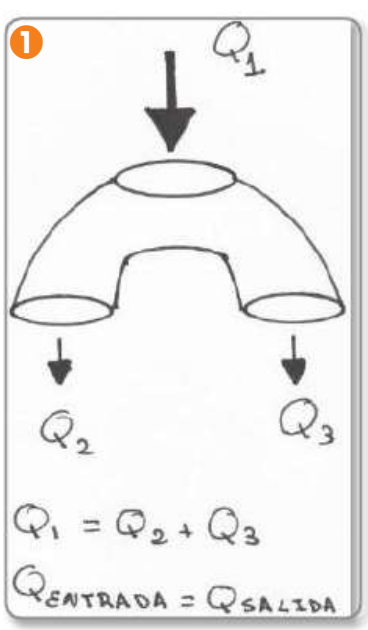
Por cada nuevo nodo se generarán nuevas intensidades.

Para entender cómo se comporta la intensidad en un circuito se suelen hacer símiles con el agua.

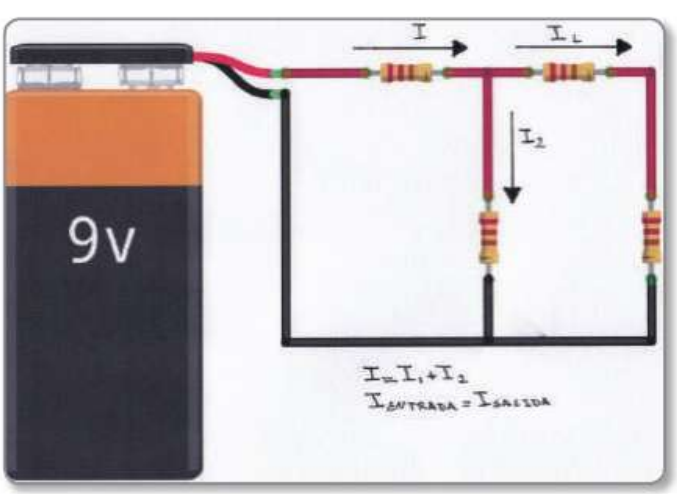
1. Supongamos que la intensidad circula por un circuito al igual que el agua por las tuberías.
2. Siendo así, podemos considerar que la toma principal del agua (que es de donde se obtiene todo el caudal que circulará por el circuitos de tuberías) es equivalente a una pila (puesto que también es de donde surge toda la intensidad que circula).
3. Las tuberías son similares a los cables de nuestro circuito, puesto que en ambos casos sirven como medio de transporte de agua o intensidad.
4. Y la demanda de caudal por cada ramal (ya bien sea para un grifo, un lavabo, etc.) es como una resistencia que demanda una cantidad de intensidad.
5. Así, podemos asegurar que el caudal total que salga por la toma principal es igual a la suma de caudales que salga por todos los dispositivos conectados al circuito de agua 1 .
6. Del mismo modo, podemos asegurar que la intensidad que salga por la pila es igual a la suma de intensidades de cada uno de los nodos del circuito.
7. En conclusión, la intensidad de entrada (aportada por la pila) es igual a la de salida (a la que se consume).

La ley de intensidad de Kirchhoff establece que el sumatorio de

1



1



intensidades sobre un nodo (intensidades de entrada + salida) es igual a cero.Para estudiar las tensiones en cada elemento de un circuito debemos dividirlo en lo que se conoce como mallas ❷ .

1. Una malla no es más que una referencia a una especie de camino cerrado, formado por los mismos cables y componentes.
2. En un circuito puede haber varios caminos e incluso entre ellos se pueden compartir componentes.
3. Debemos marcar el sentido de cada tensión dentro de la malla, puesto que esto nos indicará si en el sumatorio de tensiones de una malla es necesario sumar alguna de las tensiones con signo negativo.

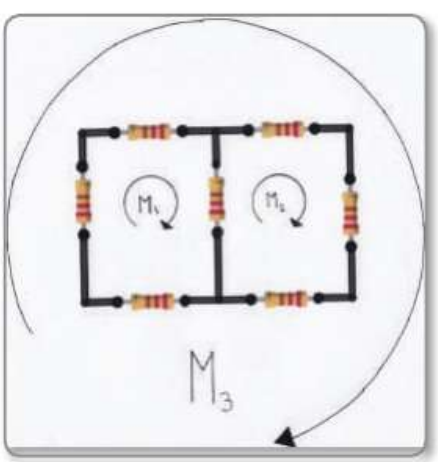
La ley de tensiones de Kirchhoff afirma que el sumatorio de tensiones de una malla es siempre igual a cero.

Con esta última ley estamos en disposición de entender el análisis del circuito, para calcular la resistencia que quedaba pendiente de calcular del capítulo anterior.

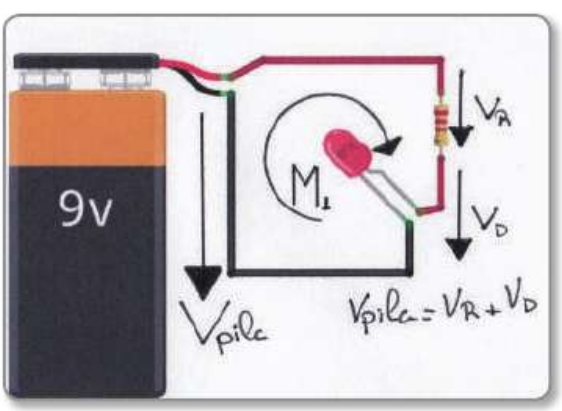
Si analizamos la malla resultante, de conectar la resistencia con la batería y el diodo ❸, deducimos que:

- La tensión de la pila es igual a la caída de la tensión de la resistencia más la del diodo:
  Vpila=Vres+Vdiodo
- Como conocemos los valores de la tensión de la pila y la que soporta el diodo podemos obtener la tensión de la resistencia: Vres=Vpila-Vdiodo=9-1,8=7,2
- Conociendo el valor de la tensión de la resistencia y queriendo limitar el flujo de intensidad a 10mA calculamos el valor de la resistencia: R=Vres/I=7,2/0,010=720 Ω

❷



❸



Conectar un led a Arduino

IMPORTANTE

Por convenio, cuando conectemos algún componente con el pin de 5V o de 3,3V identificaremos esta unión por un cable rojo, y en caso de ser a GND por un cable negro.

Para realizar las conexiones nos ayudaremos de una protoboard, que es algo así como un panel de conexiones ❶ .

- Distinguimos varias partes en función del modo de conexión interno de los puntos: una parte superior y una inferior (recuadros azules) en las cuales todos los puntos de una misma fila están conectados entre sí; dos partes intermedias (recuadros verdes) en las que los puntos de una misma columna están conectados entre sí.
- En la parte media nos encontramos una ranura que separa cada columna en dos partes, las cuales son independientes entre sí ❷ .

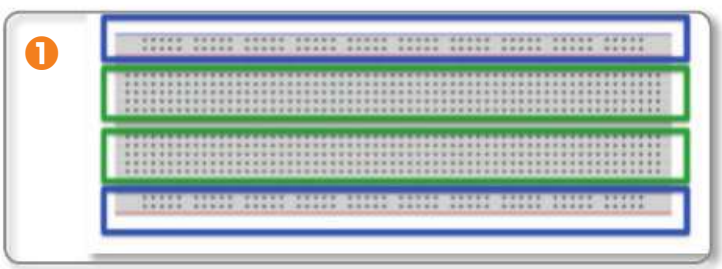
Si nuestro objetivo es que el led se encuentre continuamente encendido podemos conectarlo al pin de 5V, que ofrece una tensión constante ❸ .

1. A diferencia del caso anterior, en el que el led se conectaba a una tensión de 9V y realizábamos los cálculos necesarios para obtener la resistencia adecuada, en este caso (independientemente que sea al pin de 5V o como veremos a un pin digital) se conectará a una tensión de 5V.
2. Por lo tanto debemos rehacer los cálculos:

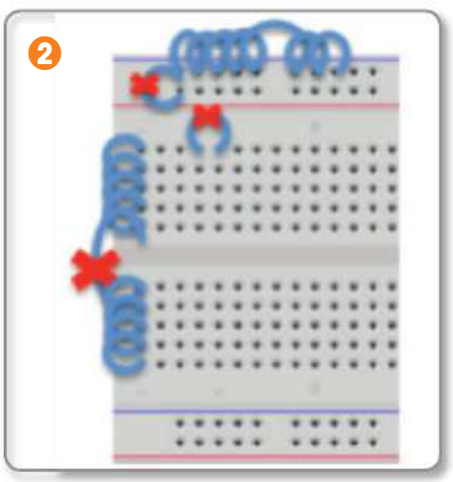
   $$Vcc=Vres+Vdiodo$$
   $$Vres=Vcc-Vdiodo=5-1,8=3,2V$$

3. Sabemos que la tensión debe soportar una tensión de 3V independientemente de su valor, debemos calcular su valor para no dañar ni el Arduino ni el led, por lo tanto vamos a establecer que la intensidad será de 10mA.

❶



❷



4. Vres=I*R
   R=Vres/I=3,2/0,01=320Ω
5. Por estándar emplearemos una resistencia de 330Ω, por lo tanto si rehacemos los cálculos la intensidad será de:
   I=Vres/R=3,2/330=0,0096A=9,6mA

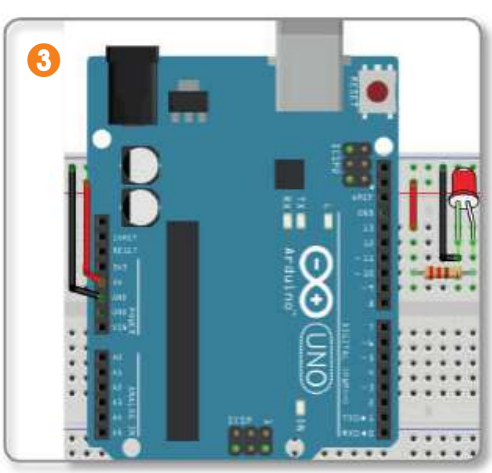
Hasta el momento hemos empleado únicamente los pines de GND y 5V del Arduino. Para empezar a programar lo que haremos será controlar el encendido del led a través de los pines digitales.

1. Debemos cambiar el conexionado, en vez de conectarse al pin de 5V lo conectaremos al pin 13 4 .
2. Disponemos de 14 pines digitales, enumerados del 0 al 13
3. Estos pines se pueden comportar como salidas (controlar procesos) o entradas (tomar lecturas); este comportamiento deberemos configurarlo por programación.
4. Es importante tener siempre presente sus caracteríticas de tensión e intensidad: 5V y 20mA.
5. Los casos se basarán en un led conectado al pin 13, pero son perfectamente válidos para cualquiera de los demás pines digitales (simplemente es necesario indicarlo por programación).
6. Evitar en estos primeros casos conectar algún componente en los pines 0 y 1, profundizaremos en esto pero adelantar que intervienen en la comunicación entre el Arduino y nuestro ordenador.

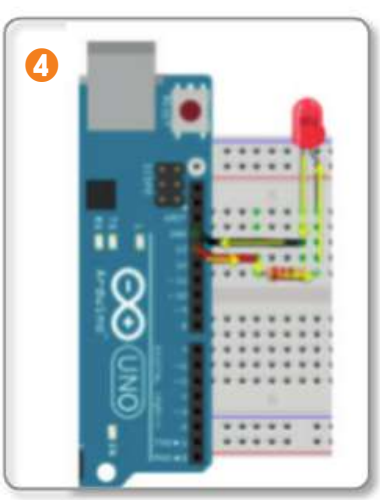
IMPORTANTE

A menos que se especifique lo contrario, siempre trabajaremos con el último circuito que quede montado, en este caso con el que hace referencia a la imagen 4. Este será el que debemos de tener como referencia para los próximos capítulos hasta que se cree uno nuevo.

3



4



Conexión Arduino IDE

IMPORTANTE

El puerto USB ofrece una tensión de alimentación de 5V y un máximo de 500mA.

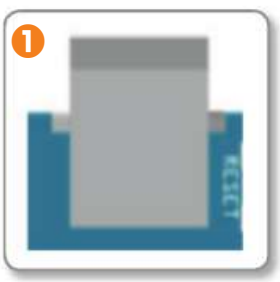
Esto quiere decir que, si elegimos este periférico para alimentar nuestros proyectos, no puede haber un consumo superior a 500mA; de ser así, la alimentación fallará y no funcionará.

1. Con el led conectado al Arduino podemos conectar el Arduino al ordenador.
2. Si la conexión se realiza al pin 13 y nunca hemos programado el Arduino, nos encontraremos con que el led queda en un modo de parpadeo.
3. Como ya sabemos, es necesario que le instalen en el MCU un gestor de arranque (bootloader); una de las formas rápidas que tiene para comprobar su funcionamiento es la carga del programa que haga que el led asociado al pin 13 parpadee.

En este primer caso interviene otra parte del hardware: el conector USB ❶ .

1. Nos ofrece un punto de conexión entre el Arduino y el IDE, para la transferencia de programas y datos.
2. Permite que el Arduino quede alimentado.

El siguiente pasó será instalar el IDE de Arduino, para ello seguiremos los siguientes pasos.



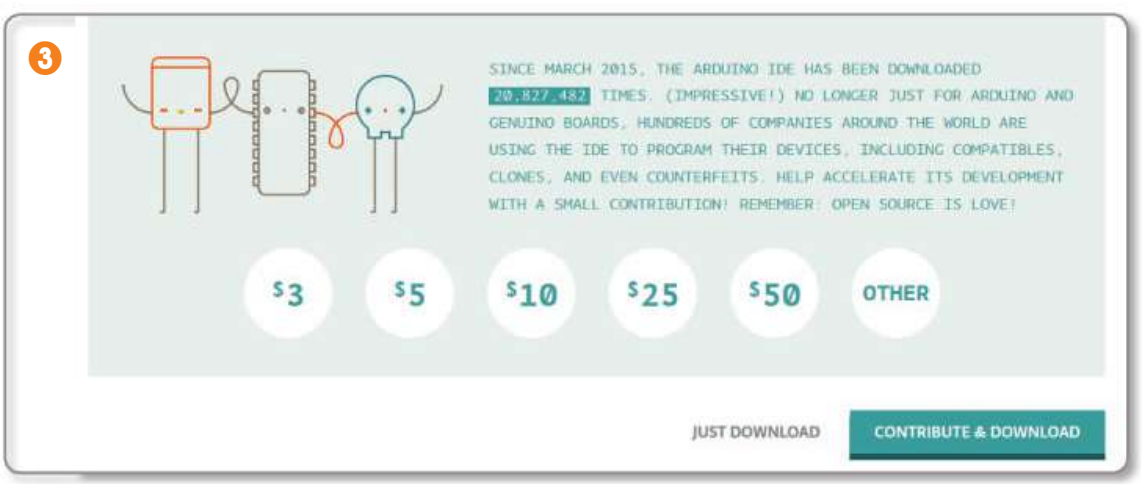
1. Accedemos al apartado de software de la web de Arduino https://www.arduino.cc/en/Main/Software.
2. Seleccionamos el instalador para el sistema operativo del ordenador ❷ .
3. Continuamos con la descarga ❸ .
4. Si abrimos el ejecutable simplemente seguiremos los pasos que nos indique para instalar el IDE .

5. Una vez instalado el IDE, si lo abrimos podemos distinguir una serie de apartados de este entorno 4 .

IMPORTANTE

En este libro trabajaremos con la versión 1.8.5 del IDE de Arduino.

En próximos capítulos nos centraremos en las partes básicas de este entorno así como las reglas que debemos seguir.



Conociendo el IDE y la programación

IMPORTANTE

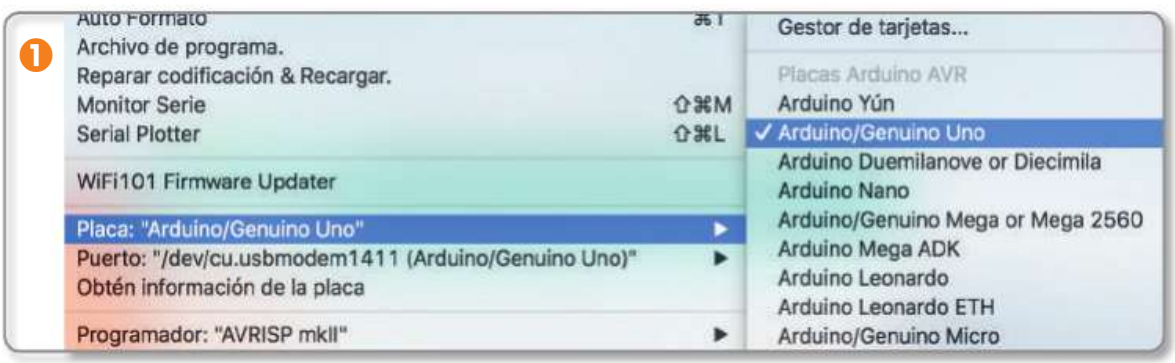
Es importante realizar los dos primeros pasos, el primero para que el IDE realice las operaciones necesarias para la correcta carga del programa según el Arduino con el que trabajemos; y el segundo para verificar que el programa se cargará al Arduino.

Debemos comprobar que el IDE se encuentra configurado de acuerdo al Arduino que se va a programar, y que lo detecta por el puerto USB correspondiente .

1. Nos dirigimos a las herramientas, al apartado de placa y verificamos que la placa configurada es la del Arduino UNO ❶ .
2. Volvemos a las herramientas, al apartado de puerto, y verificamos que detecta que nuestro Arduino se encuentra conectado a uno de los puertos. Dependiendo del sistema operativo (en mi caso MAC OS X), el identificativo del puerto cambia /dev/cu.usbmodem; por ejemplo, en Windows será mediante COM ❷ .

Con los pasos anteriores realizados, podemos adentrarnos en lo que es la programación. Al crear un nuevo programa este viene con unas líneas de código ya escritas. Estas líneas definen las dos estructuras básicas de un programa de Arduino: void setup y void loop ❸ .

- Ambas estructuras permiten englobar líneas de código entre las llaves de apertura y cierre asociadas.
- Cada vez que se enciende o resetea nuestro Arduino se ejecutan las líneas de código pertenecientes a la estructura del void setup.



- El void setup se emplea por lo general para realizar configuraciones e inicializaciones.
- En él se pueden escribir tantas líneas de código como necesitemos, el único límite es que el programa final por su peso no exceda la capacidad de la memoria Flash del MCU, que es donde se va almacenar.
- Las líneas de código tanto del void loop como del void setup se ejecutarán de forma secuencial, es decir, empieza la ejecución en la primera línea y continúa la ejecución de las restantes líneas por orden hasta llegar a la última.
- Una vez finalizada la ejecución de la última línea del void setup el programa "salta" a la primera línea del void loop.
- El programa entra en un bucle infinito por el cual se encuentra continuamente en la ejecución de las líneas de código que hubiese en el void loop.
- Al ejecutarse sus líneas del mismo modo que el void setup (secuencialmente). Una vez ejecutada la última línea el programa vuelve a la primera para volver a realizar este proceso una y otra vez.

A medida que se adquieran conceptos de programación veremos que podemos cambiar la ejecución secuencial de las líneas de código mediante estructuras que permitan la toma de decisiones al igual que nos encontraremos que podemos fragmentar el programa para optimizar recursos y hacerlo más comprensible.

En resumidas cuentas, todas las configuraciones e inicializaciones las haremos en el void setup y el funcionamiento que le queramos dar a nuestro proyecto lo implementaremos en el void loop.

estructura §

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Reglas y comentarios en programación

IMPORTANTE

A lo largo del libro se harán referencia a buenas prácticas a la hora de empezar un proyecto o para el desarrollo de este.

El comentar nuestros programas se considerará como una buena práctica que debemos acostumbrarnos a realizar.

Antes de empezar a programar debemos tener en cuenta ciertas normas:

1. El IDE de Arduino distingue mayúsculas de minúsculas, por lo tanto debemos fijarnos a la hora de escribir una instrucción si se escribe en mayúsculas o minúsculas.
2. No son necesarias tabulaciones, no es obligatorio separar las instrucciones del margen izquierdo del IDE, pero sí recomendable, sobre todo cuando veamos instrucciones de control de flujo, para estructurar el programa adecuadamente.
3. Si hacemos click con el botón derecho en cualquier parte del programa se nos abrirá un desplegable y tenemos la opción de autoformato, que nos permite reordenar las lineas de código para que se entienda mejor su estructura 1 .
4. Todas las instrucciones acaban en ";", de no hacerlo y al tratar de cargar el programa a nuestro Arduino, el IDE no nos lo va a permitir y nos mostrará un error. El punto y coma permite que el IDE diferencie cuándo empieza y acaba una instrucción; si nos lo saltamos, juntaría una o más instrucciones, las evaluaría como si fuesen una única ins-

trucción y nos daría un error diciéndonos que no reconoce esa instrucción.

5. Es obligatorio que un programa contenga los dos campos básicos de void setup y void loop. Aunque estén vacíos deben ir definidos.

Dentro del setup y loop, encontramos dos líneas de código que empiezan por //. Estas líneas son comentarios.

1. Es importante comentar nuestros programas, un comentario sirve para hacer aclaraciones o explicaciones a lo largo del programa. Esto nos va a permitir que, por ejemplo, si hacemos un programa y, dentro de un año, lo queremos modificar, nos resulte más sencillo interpretarlo con comentarios que sin ellos y, por lo tanto, poder modificarlo.
2. Podemos hacer tantos comentarios como queramos, no va a influir en el tamaño del programa que finalmente se cargue al Arduino. El IDE de Arduino evalúa qué líneas del programa son comentarios y cuáles no, las que sean comentarios no las va a tener en cuenta a la hora de enviarlas al Arduino.
3. Existen dos tipos de comentario ❷ : de una sola línea o multilínea. Podemos hacer comentarios en cualquier parte del programa.
4. Los comentarios de una sola línea vienen precedidos por //, lo cual indica que desde ese punto hasta el final de la línea es todo un comentario.
5. Los comentarios multilínea empieza por /* ; en medio podemos escribir tantas líneas como queramos, que quedarán fijadas como comentarios de forma automática. Para indicar fin de comentario multilínea se hace con */.

IMPORTANTE

Los comentarios también son útiles para encontrar errores en nuestro programa, puesto que si el Arduino no funciona como deseamos podemos ir comentando líneas de código para que no se carguen y buscar el error, en vez de borrar o modificar líneas.

❷
```
//Con las dos barras indicamos que todo lo que se escribe a partir de las barras y en esta línea es un comentario
/*
 * Si empezamos por una barra y un asterisco y pulsamos enter
 * podemos hacer tantos saltos de línea como necesitemos puesto
 * que automáticamente se define otro asterisco y barra de cierre
 */
```

pinMode digitalWrite

IMPORTANTE

Sabiendo el valor que se escribe en un registro a través de las palabras reservadas: OUTPUT e INPUT podemos sustituirlas por su equivalente en binario.

pinMode(13,1): el pin 13 quedaría configurado com salida.

pinMode(13,0): el pin 13 quedaría configurado com entrada.

Lo mismo ocurre con las palabras reservadas HIGH y LOW.

digitalWrite(13,1): pin 13 activado.

digitalWrite(13,0): pin 13 desactivado

Como sabemos, los pines digitales se pueden comportar como entradas o salidas, por lo que necesitamos una instrucción para configurar su comportamiento.

- pinMode(parametro1,parametro 2);

Vemos que, entre paréntesis, esta instrucción necesita dos parámetros: el primero hace referencia al pin que queremos configurar, y el segundo, al modo de configuración (entrada/salida).

- Pin 13 como salida: pinMode(13,OUTPUT);
- Pin 13 como entrada: pinMode(13,INPUT);

¿Es necesario configurar todos los pines? No, solo los que utilicemos. Antes de seguir, vamos a explicar un poco el sistema binario, lo que nos va a permitir entender mejor lo que estamos haciendo.

1. El sistema binario es un sistema de numeración en el que solo existen dos números: 0 y 1. Todos los sistemas informáticos trabajan con el sistema binario 1 .
2. La unidad más básica del sistema binario es el bit. Podemos imaginar un bit como una celda en la cual solo puede haber un 0 o un 1 pero siempre tiene que haber un valor, no existen bits vacíos.
3. ¿Qué estamos haciendo realmente con la instrucción pinMode(,)? El MCU del Arduino internamente trabaja con ceros y unos, el MCU dispone de unos registros para realizar con-


1

figuraciones. Lo que nosotros estamos haciendo con la instrucción pinMode es escribir un valor determinado en esos registros.

4. Viendo esta primera introducción al sistema binario, de la instrucción pinMode, el primer parámetro hace referencia al registro que estamos configurando.
5. Y el segundo parámetro indica el valor que escribimos. Este registro es de un único bit, por lo tanto escribiremos un 0 o un 1.
6. En caso de OUPUT escribimos un 1 y en caso de INPUT escribimos un cero ❷ .
7. Como no puede haber registros vacios, ¿qué pasaría si no escribimos nada en un registro? Al resetearse o encenderse el Arduino los registros se resetean por lo que su valor por defecto es 0.

Para poder activar o desactivar una salida disponemos de la siguiente instrucción:

- digitalWrite(parámetro1,parámetro2);
- Al igual que pinMode, dispone de dos parámetros, el primero para indicar sobre qué pin estamos realizando la operación y el segundo para señalar si queremos que se active o no.
- Activar salida 13; digitalWrite(13,HIGH);
- Desactivar salida 13 ;digitalWrite(13,LOW);

Al igual que ocurría con pinMode, con digitalWrite estamos escribiendo en el registro correspondiente un 0 o un 1. Por defecto tiene un cero, por lo tanto, si no lo cambiamos, no se encenderá el led.

IMPORTANTE

A lo largo de los próximos capítulos haremos continuas referencias al sistema binario puesto que es la base para entender la electrónica digital.

❷

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

}
```

Pin	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Configuración	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Cargar primer programa

IMPORTANTE

Si realizamos un programa con varios errores, la consola del IDE nos muestra el primero que encuentre. Si, por ejemplo, tenemos cuatro errores, el IDE nos mostrará el primero que encuentre; debemos solucionarlo y volver a compilar para que nos diga cuál es el siguiente error. Sin errores, podemos subir el programa al Arduino.

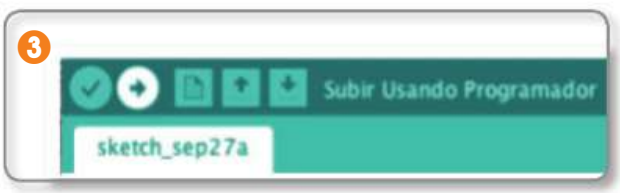
La consola nos irá mostrando mensajes sobre cada proceso.

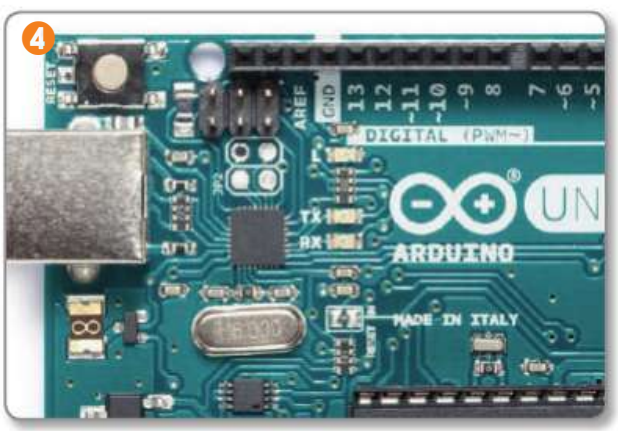
1. Lo primero que tenemos que hacer es comprobar que no hayamos cometido ningún error. Para ello, nos vamos a la barra de botones y pulsamos el botón de verificar (botón en blanco) ❶ .
2. Cada vez que abramos un nuevo proyecto en Arduino y tratemos de cargar o verificar su contenido, si nunca hemos guardado este proyecto, nos indicará si queremos guardarlo ❷ . Es recomendable hacerlo, por una parte para poder reutilizarlo cuando lo necesitemos, y por otra, para que no nos aparezca continuamente la ventana de diálogo para almacenarlo. Por defecto, le asocia un nombre en función de la fecha de creación y del número de proyectos que llevemos creados ese día, todo esto precedido del término sketch («boceto», en inglés).
3. Al instalar el IDE de Arduino se genera una carpeta con el nombre de Arduino en el directorio de Mis Documentos. Por defecto, los programas se guardan en esta carpeta (podemos cambiar la carpeta de destino). Al guardar el programa, se crea otra carpeta dentro de la anterior con el nombre del programa y, dentro, el archivo de programa, con extensión .ino; a través de la consola podemos conocer el proceso de verificación (en este caso, compilando programa).

4. Para poder cargar el programa hacemos click en el botón correspondiente ❸ .

El proceso de carga de un programa es complejo, y en él intervienen varios subprocesos en el IDE e incluso en el propio hardware.

1. El programa final que se carga al MCU está formado por un conjunto de códigos binarios que es capaz de interpretar y almacenar.
2. Con el programa final obtenido, el IDE lo envía por el puerto USB para reprogramar el Arduino.
3. Para que se pueda cargar es necesario que se reinicie el Arduino, puesto que así entrará en funcionamiento el gestor de arranque que esperará durante un segundo si existe la trasferencia de un nuevo programa. De ser así el arduino queda a la escucha de todos los datos transferidos para almacenarlos en la memoria FLASH del MCU.
4. Finalizado este proceso el MCU comienza con la ejecución del nuevo programa empezando por las líneas de código del void setup para luego ejecutar en bucle las líneas del void loop.
5. Pero para que todo esto sea posible, el MCU debe "entender" lo que le envíe el IDE y como no tiene la capacidad de trabajar directamente con el protocolo USB, el hardware del Arduino UNO incorpora un segundo MCU que se encarga de hacer la esa traducción ❹ .





Temporizaciones

IMPORTANTE

El inconveniente de las instrucciones delay y delayMicrosenconds es el bloqueo de la ejecución de otras instrucciones; en una inmensidad de casos esto más que una solución nos resultará un inconveniente, por eso disponemos de lo que se conoce como interrupciones por tiempo. Las interrupciones por tiempo no se estudiarán en este libro, dejando este tema para el siguiente volumen.

Si analizamos el hardware del Arduino UNO, encontraremos que incorpora un oscilador ❶ . Sin entrar en más detalles nos quedaremos con que un oscilador es un dispositivo capaz de generar señales periódicas.

Estas señales permiten que el Arduino disponga de una señal de reloj, algo que le indica cada cuánto debe ejecutar instrucciones, y también se utiliza para las temporizaciones que nos permitirán realizar operaciones cada cierto tiempo, regular procesos mediante señales PWM o incluso obtener el tiempo trasncurrido entre eventos que surjan, por ejemplo el tiempo transcurrido entre varias pulsaciones.

El "mecanismo" del MCU que maneja este tipo de señales se concoce como Timer. El MCU del Arduino UNO dispone de 3 Timers, y es de gran interés su funcionamiento, pero de momento veamos las funciones más sencillas y en el capítulo dedicado a las señales PWM profundizaremos en los Timers.

El leguaje de programación de Arduino dispone de instrucciones para la gestión de tiempo. En este caso veremos cómo hacer que un led parpadee cada segundo. Para ello emplearemos instrucciones que detengan la ejecución de código durante un tiempo determinado.

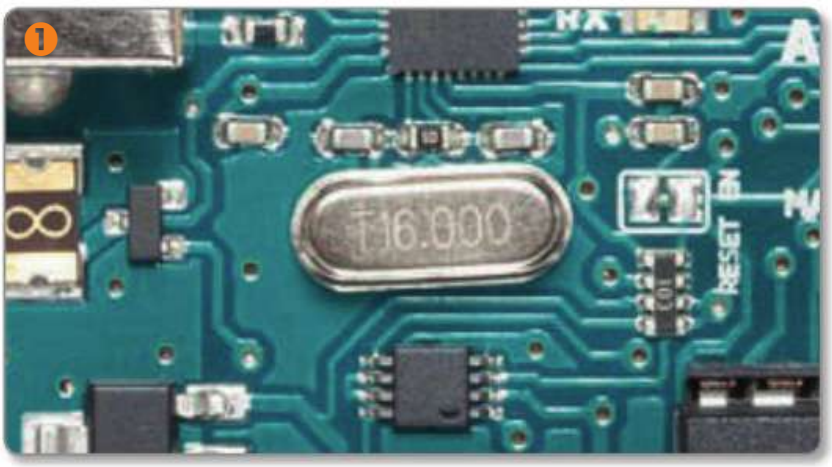
- delay(parámetro);

1. Esta instrucción detiene la ejecución del programa durante un tiempo pasado por parámetro. La unidad de tiempo con la que trabaja es en milisegundos.
2. Detener durante 1 segundo: delay(1000); ❷ .
3. Límites del parámetro: 0 a 4294967295 (sin decimales).

- delayMicroseconds(parámetro);

1. Instrucción similar a la anterior, pero cambia la unidad de tiempo: esta instrucción trabaja en microsegundos y los límites del valor del parámetro son de 3 a 16383.
2. Esta instrucción nos permite detener la ejecución del programa en tiempos más pequeños que el delay, mientras que el delay permite llegar a temporizaciones más grandes.

1



2

```
void setup()
{
  //Configuramos el pin13 como salida
  pinMode(13,OUTPUT);
}
void loop() {
  //Encendemos el led
  digitalWrite(13,HIGH);
  //Lo mantenemos encendido durante 1 segundo
  delay(1000);
  //Apagamos el led
  digitalWrite(13,LOW);
  //Lo mantenemos apagado durante 1 segundo
  delay(1000);
  /*Al finalizar el segundo se vuelve a repetir
  la operación*/
}
```

Entradas digitales

IMPORTANTE

Si tenemos conectada una batería de, por ejemplo, 12V, no debemos conectar por ejemplo una pila de 9V al pin de Vin puesto que esto pondría en paralelo ambas fuentes de alimentación, y debido a que son de diferente potencial se dañarían pudiendo incluso dañar el Arduino.

Trabajar con un pin como una entrada significa que este debe leer el valor de una señal. Antes de aprender a hacerlo con un Arduino, vamos a hacerlo con un multímetro.

Para medir el voltaje de una señal con un multímetro tenemos que conectar los dos polos de la batería a este. En el Arduino, sucede lo mismo; al fin y al cabo, estamos realizando la misma operación.

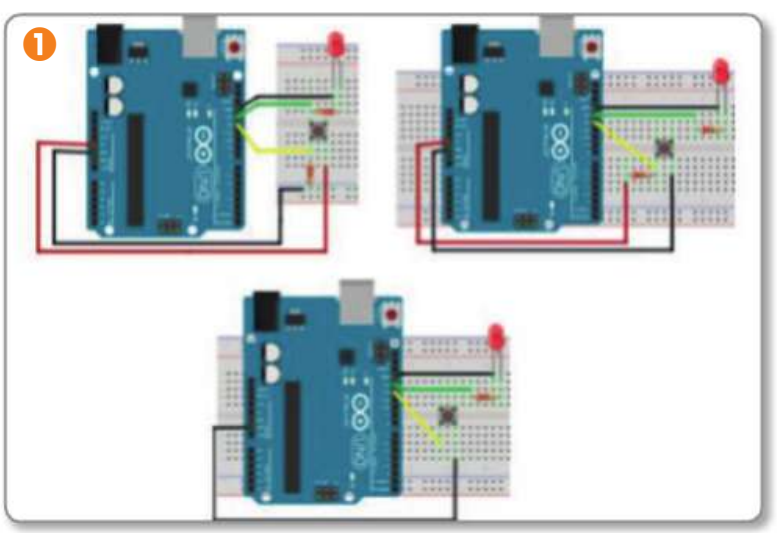
En el multímetro teníamos dos cables: negro (-) y rojo (+). Sus equivalentes en el Arduino son GND y cualquier otro pin (digital o analógico, a través del cual se quiera tomar la lectura).

A la hora de conectar un interruptor o un pulsador, disponemos de tres tipos de conexiones. Pull-Up, Pull-Down e InputPull-up ❶.

Antes de entrar de lleno en el estudio de estas conexiones, revisemos una parte del hardware del Arduino UNO que definiremos como hardware de potencia ❷.

1. Disponemos de un pin de 5 V y otro de 3,3 V, a través de los cuales podemos demandar 20 y 50 mA, respectivamente.
2. Con dos pines de GND, es indiferente en cuál realicemos la conexión.
3. También observamos el pin de Vin, a través de él podemos alimentar nuestro Arduino, comparte las mismas propieda-

❶



des que el conector Jack.

4. Lo ideal, en caso de alimentar el Arduino por el Vin, sería entre 7 y 12 voltios; según las especificaciones y la demanda de intensidad, no debe superar 1 amperio.
5. No hay problema por tener conectado el Arduino por USB al ordenador y al mismo tiempo a una batería, el hardware se encarga de escoger en todo momento la "mejor" fuente de alimentación.
6. Independientemente del punto de alimentación que escojamos al MCU le llegarán 5V, que son según sus especificaciones con los que puede operar.

Como tenemos acceso al diseño del hardware de Arduino, podemos estudiarlo 3 .

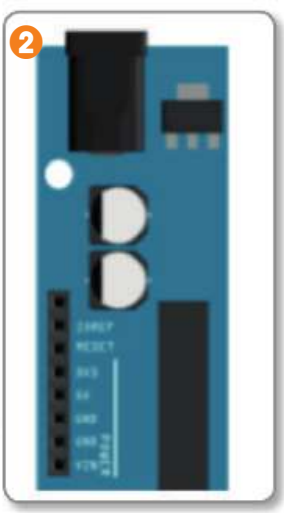
1. El primer elemento -X1- es equivalente del conector Jack en un esquema eléctrico.
2. Si seguimos por la línea 2 (PWRIN) nos encontramos con un diodo (D1), su función es la de proteger al resto del circuito ante una intensidad elevada.
3. A continuación la línea se divide hacia un condensador (PC1) y un regulador; este último es el que consigue reducir la tensión de entrada a 5V.
4. Ese primer condensador se utiliza para que la tensión no caiga bruscamente cuando desconectemos la pila.
5. A la salida del regulador (líneas 2 y 4) tenemos el segundo condensador (PC2), cuya función es similar a la del primero, pero para el resto del circuito.

IMPORTANTE

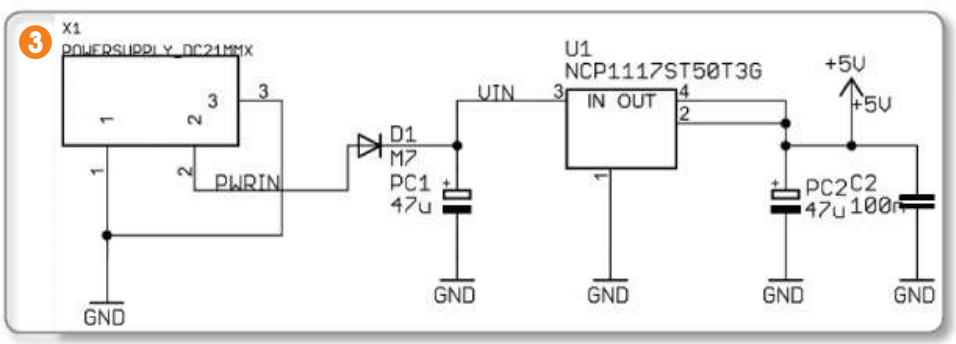
El motivo por el cual no debemos alimentar a menos de 6V por el conector Jack es precisamente por el diodo de protección, puesto que este hace que se reduzca hasta 1,2V la tensión de alimentación a la entrada del regulador.

Ello daría una tensión de 4,8V, que no sería suficiente para el correcto funcionamiento del MCU (es necesario que sean 5V).

2



3



Alimentación

IMPORTANTE

Es importante analizar las características de la fuente que escojamos.

Intensidad máxima instantánea.

En caso de ser una pila o batería se ha de tener en cuenta su capacidad, para calcular el tiempo que puede estar aportando energía hasta descargarse.

Si trabajamos con una batería es conveniente no descargarla del todo.

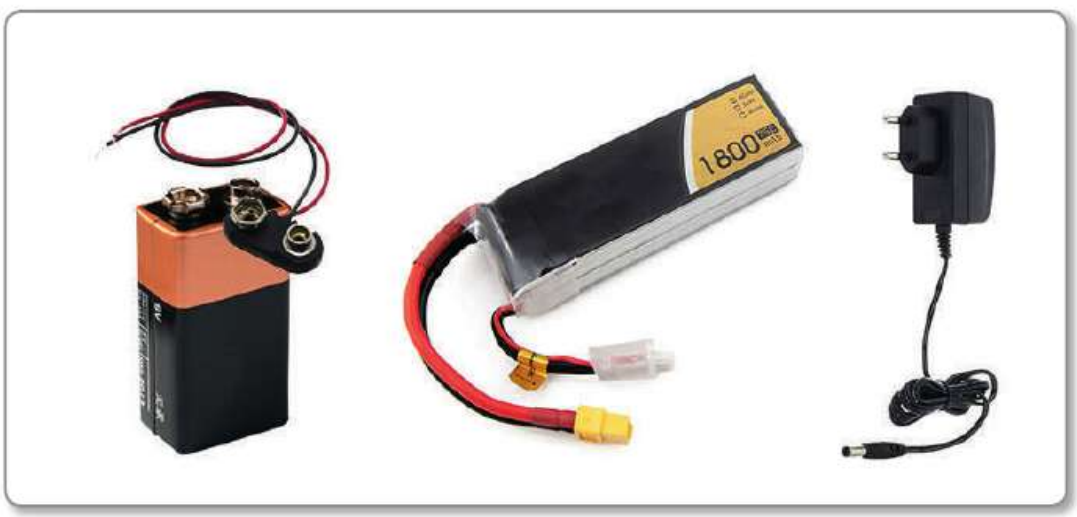
También debemos evaluar el coste en caso de escojer una batería; hay que tener en cuenta que necesitaremos de más dispositivos para cargarla.

Otro aspecto esencial para el correcto funcionamiento de nuestro Arduino es la elección de la alimentación adecuada. Como el Arduino no produce ni tensión ni intensidad necesita conectarse a un "generador" 1 que suministre la alimentación en cada momento.

1. El primer generador con el que trabajamos fue el ordenador, a través del periférico USB del hardware de Arduino; a través de él podemos requerir un máximo de 5V y 500mA.
2. El segundo periférico por el cual podemos alimentar un Arduino es el conector Jack, a través del cual podemos demandar un máximo de 1A y una tensión entre 7 y 12V como mejor rango de tensiones.
3. A este segundo periférico podemos conectar diferentes tipos de "generadores": pilas, baterías y otras fuentes de alimentación.

- Una pila por lo general no se puede cargar, aspecto importante a la hora de escoger este generador ,puesto que una vez que agote su carga será necesario reemplazarla.
- Por el contrario una batería sí permite ser recargada, y por lo general suelen ser de mayor capacidad que las pilas, aún así es importante conocer su vida útil (el número de cargas y descargas).
- En cuanto a las fuentes de alimentación, son dispositivos que conectamos a un enchufe, aportan un tensión constante

1



gran diferencia con respecto a las pilas y baterías puesto que según se vayan descargando su tensión también lo hace; otra diferencia a destacar reside en que una fuente de alimentación no necesita ser cargada, pero depende de la red eléctrica con lo que ante un posible fallo en el suministro nos quedamos sin fuente de alimentación.

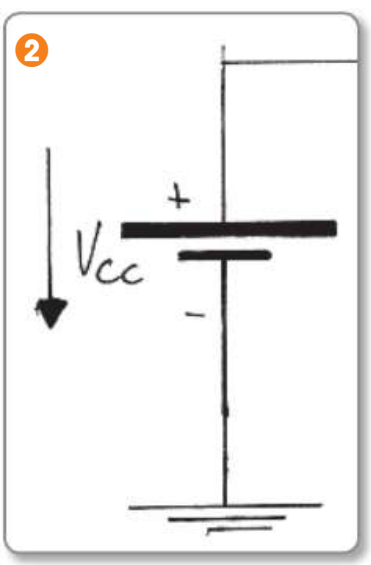
- Una fuente de alimentación dispone de un símbolo esquemático, el cual especifica los dos polos del generador [2] .
- Sea cual sea el generador, este deberá disponer de un conector macho Jack de 2,1mm.
- En caso de tener que modificar el conector que traiga originalmente:

1. Este dispone de dos patillas, una central y otra exterior.
2. Conectaremos el polo positivo a la patilla central.
3. Y el negativo con la patilla exterior.

En algún momento nos surgirá la idea de alimentar nuestros proyectos mediante un panel solar, esto sería posible aun sin tener amplios conocimientos sobre el tema, de hecho disponemos de una gran variedad en el mercado cuya instalación es sencilla, pero hay que tener en cuenta que de implementar un panel solar también necesitaremos una batería y el hardware necesario para cargar la batería a través del panel solar.

IMPORTANTE

La manipulación de generadores de tensión y corriente puede resultar peligrosa, por lo tanto la mejor opción es buscar una solución que no nos obligue a manipularlos y nos permita su implementación de forma sencilla.



Pulsador Pull Down y digitalRead

1. Con este tipo de conexión solo circulará intensidad si pulsamos el botón ❶ .
2. Con el Arduino haremos lo mismo, que es monitorizar el pulsado del botón ❷ .

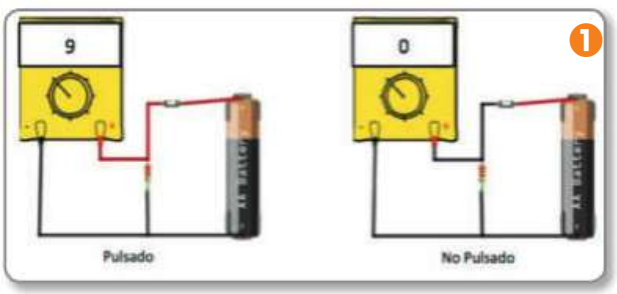
En este primer caso lo que se busca es que el led se encienda a través del Arduino en función de si se ha pulsado o no el botón.

- digitalRead(parámetro);

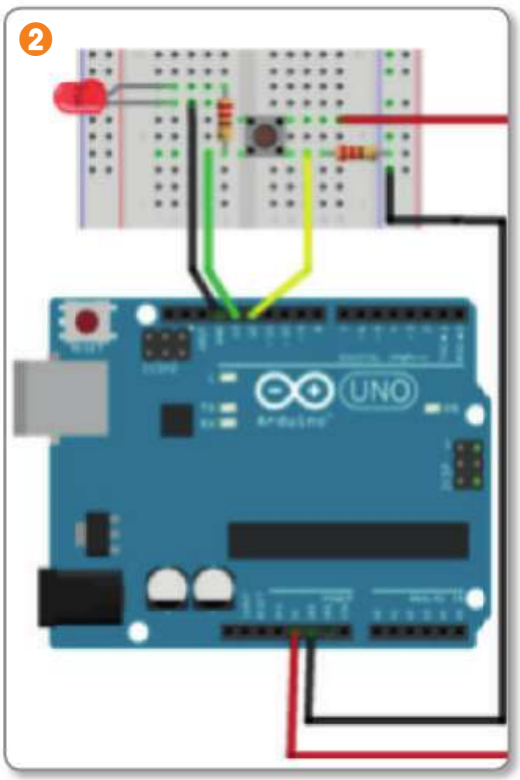
Esta instrucción utiliza un único parámetro, que es el pin del cual queremos leer la tensión. Nos devuelve un 1 si la tensión de entrada es de 5V y un 0 si la tensión es de 0V. Lo que hacemos es leer el valor del registro correspondiente.

Lo que hay que hacer es actualizar el estado de led en función del valor del pulsador. Para ello, empleamos la instrucción que ya habíamos visto de digitalWrite(13; hasta aquí, no tenemos ningún cambio ❸ .

Lo diferente aparece en el segundo parámetro de la instrucción , digitalRead(12));. Los parámetros de una instrucción no tienen por qué ser valores fijos; realmente son campos que necesitan recibir un valor y este valor puede tener varios orígenes: bien porque se introduce un valor fijo, porque escribimos en ese campo una variable que tome el valor de la variable o, como es este caso, utilizamos una instrucción que devuelve un valor ❹.

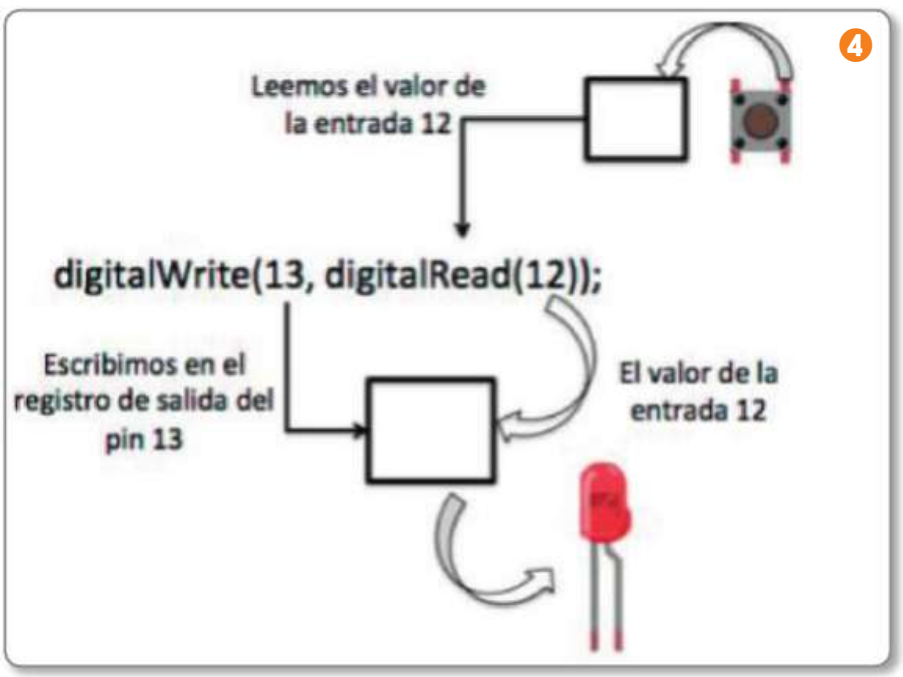


❶

❷


```
void setup()
{
  //Configuramos el pin13 como salida
  pinMode(13,OUTPUT);
  /*Inicialente apagamos el led para demostrar que
  es por la acción del pulsador*/
  digitalWrite(13,LOW);
  //Configuramos el pin 12 como entrada
  pinMode(12,INPUT);
}
void loop() {
  //Encendemos el led según el valor leído
  digitalWrite(13,digitalRead(12));
}
```

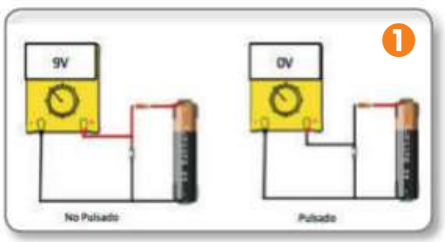
3



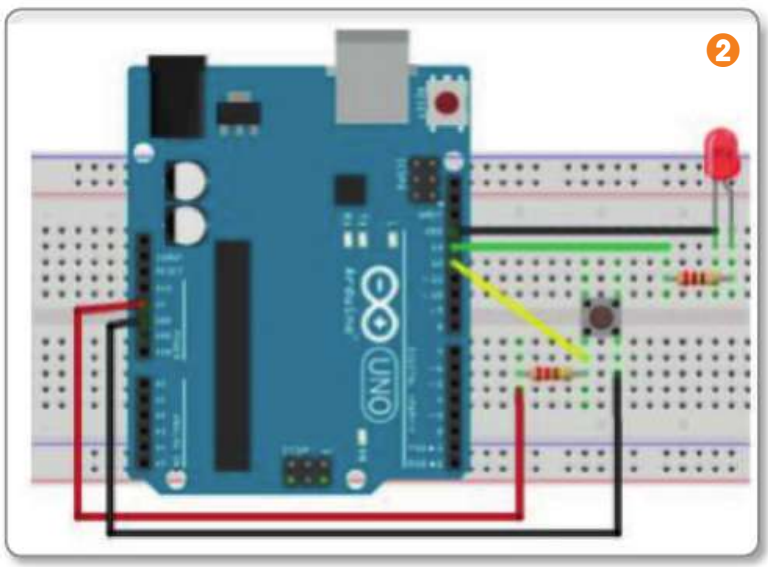
4

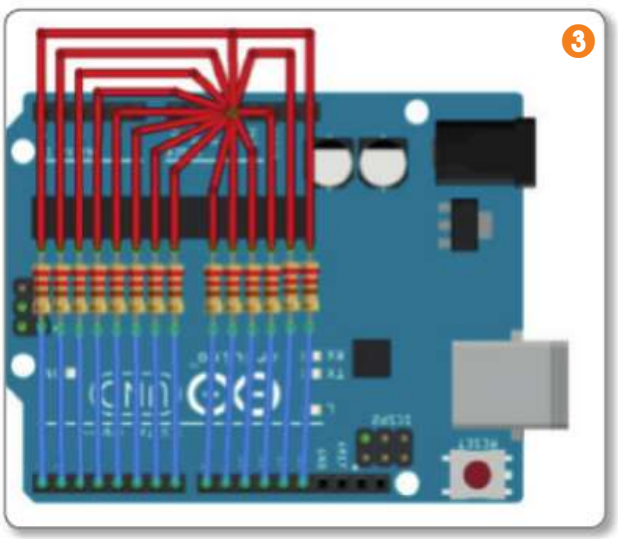
Pulsador Pull_Up e Input Pull_Up

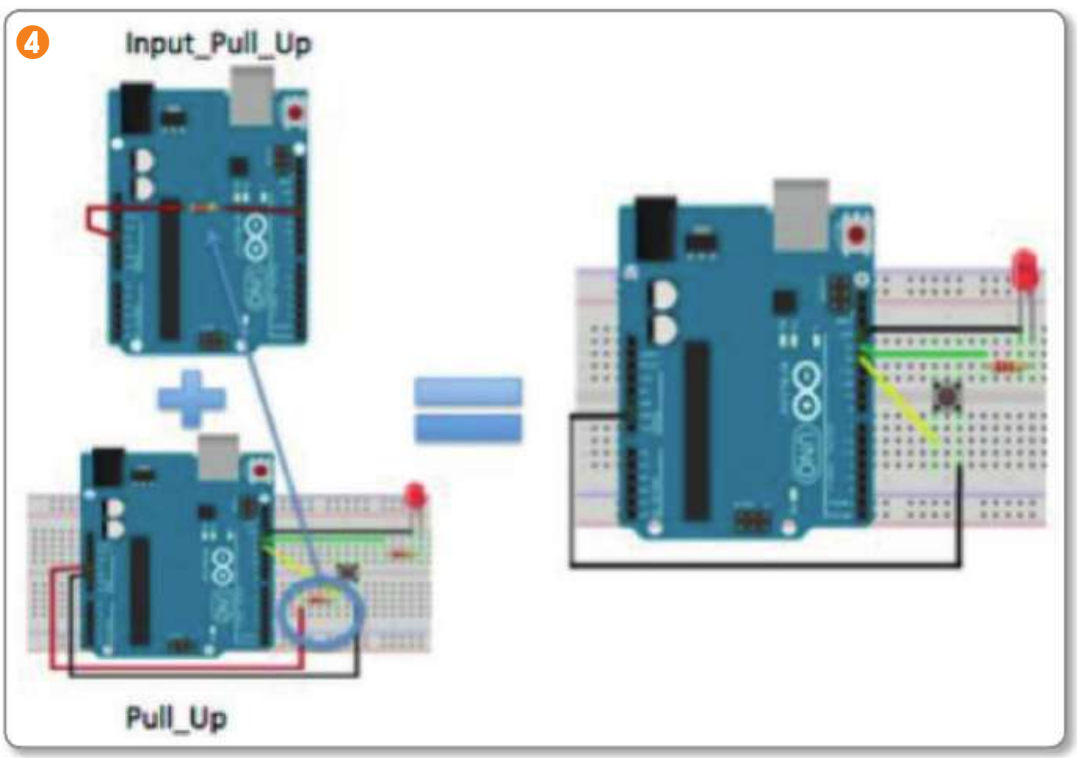
- El comportamiento del circuito Pull_UP funciona a la inversa que el Pull_Down: en el Pull_Down, cuando pulsamos la lectura es de 9V y, cuando no pulsamos es 0V, mientras que en el Pull_Up es al revés ❶ .
- Con este nuevo circuito, si cargamos el programa del capítulo anterior veremos cómo se invierte el comportamiento del sistema. Cuando se pulsa el botón el led se apaga, y cuando no se pulsa queda encendido ❷ .
- Por último, vamos a realizar una conexión Input Pull_Up. Internamente, los MCU´s incorporan una resistencia asociada a cada pin que por defecto se encuentra desconectada.
- Por programación podemos hacer que el MCU conecte una de esas resistencias por un lado al pin indicado y por el otro a 5V ❸.
- El valor de cualquiera de las resistencias es de 20KΩ, por lo que su consumo es muy reducido (0,25mA).
- Con la resistencia interna conectada nos ahorramos la resistencia conectada al pulsador ❹.
- Para realizar dicha configuración empleamos la instrucción, pinMode(,); pero esta vez en lugar de ser una INPUT o OUTPUT emplearemos la palabra reservada INPUT_PULLUP.
- Lo que nos permite esta solución es eliminar la resistencia externa que empleábamos en la conexión Pull_Up, ya que esa resistencia ahora se encuentra internamente conectada en el MCU.



❶


❷


3


4
Input_Pull_Up
Pull_Up

Invertir salida

Una vez revisadas las tres posibilidades que tenemos para conectar un botón será decisión nuestra escoger la más adecuada.

La conexión, empleando las resistencias internas del MCU, presenta la gran ventaja de eliminar componentes e incluso nos permitiría cablear directamente un pulsador . El inconveniente con el que nos encontramos (para los dos casos de Pull_Up) es su funcionamiento, que es inverso al de una conexión Pull Down.

Para esto tenemos dos formas de solucionarlo, lo que podemos hacer es que en vez de actualizar el led con el mismo estado que el del botón lo que haremos será invertirlo. Podemos elegir entre dos opciones para solventar el problema:

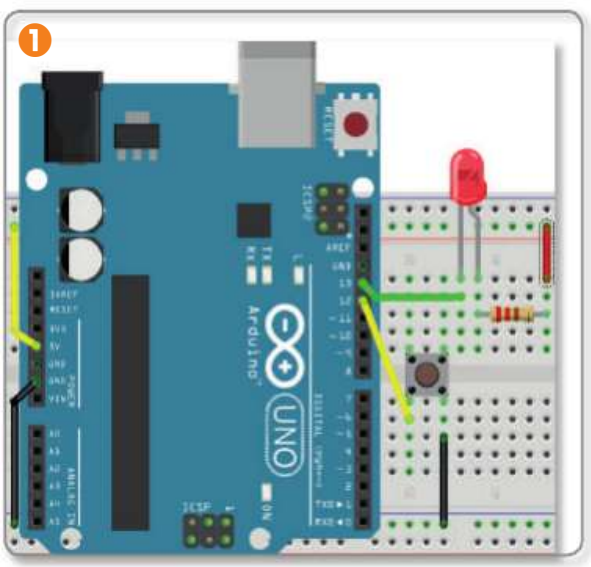
- Por hardware, invertiendo la conexión del led.
- O por programación, invirtiendo la lectura del pulsador.

Vamos a empezar por analizar la primera opción, para ello conservaremos el último conexionado del botón (INPUT_PULLUP) y el mismo programa, pero cambiaremos la conexión del led 1 .

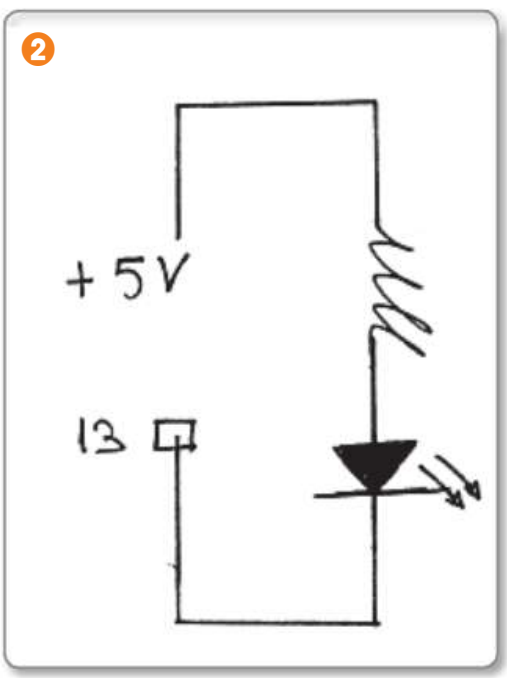
Cuando el botón no se pulsa, la diferencia de tensión sobre la conexión del led con la resistencia es cero, porque por un extremo de la conexión se conectan a 5V y por el otro extremo conectado al pin13 también hay 5V 2 .

En la opción de invertir la salida por programación, repetiremos la última conexión del capítulo anterior 3 .

1

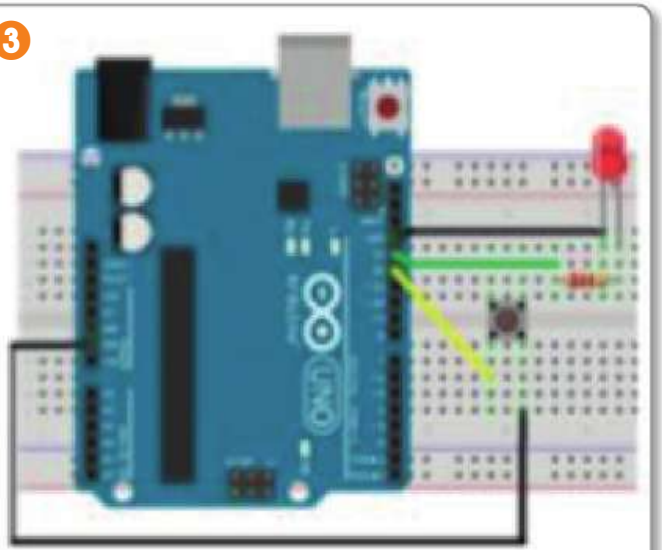


2



Y si recuperamos el programa, podemos invertir la lectura con el símbolo ! 4 . La exclamación invertida permite invertir un valor binario de un tamaño de máximo de un bit. Es decir, permite invertir un 1 o un 0 por su inverso (solo un uno o solo un cero).

3



4

```
void setup()
{
  //Configuramos el pin13 como salida
  pinMode(13,OUTPUT);
  /*Inicialente apagamos el led para demostrar que si se enciende
  es por la acción del pulsador*/
  digitalWrite(13,LOW);
  /*Configuramos el pin 12 como entrada
  conectada a la resistencia interna*/
  pinMode(12,INPUT_PULLUP);
}
void loop() {
  digitalWrite(13,!digitalRead(12));
}
```

Variables

IMPORTANTE

Según vayamos ampliando usos del Arduino revisaremos más tipos de variables, pero siempre es importante tener claro el tipo de dato con el que estamos operando, para evitar errores de almacenamiento u operación.

Una variable permite almacenar información con una referencia (nombre de una variable). El crear una variable se conoce como «declarar una variable». Existen diferentes tipos de variables, cada una de ellas almacena un tipo de dato.

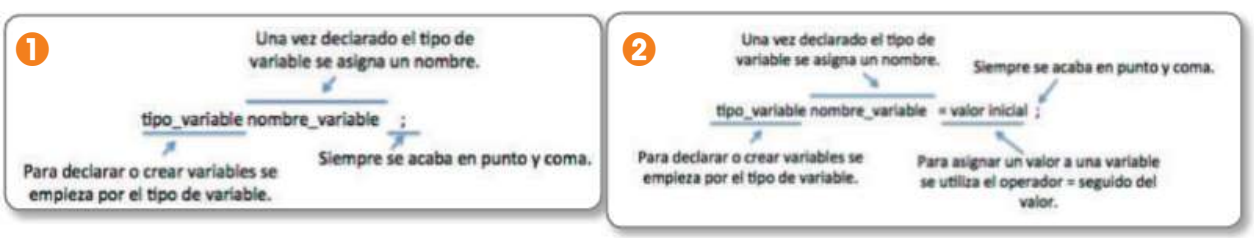
- Independientemente del tipo de variable, todas siguen la misma estructura a la hora de ser declaradas ❶ .
- Podemos asignar un valor inicial a una variable, de no ser así todos los bits que conforman dicha variable se inician a 0 ❷ .

El primer tipo de variable que veremos serán el tipo boolean. Este tipo de variable contiene un solo bit o, lo que es lo mismo, una celda en la cual solo puede haber un 0 o un 1. Para declarar una variable existen ciertas normas:

- No puede haber más de una variable con el mismo nombre.
- Los nombres de las variables no pueden empezar por un número.
- Las variables no llevan signos de acentuación.
- Tampoco llevan la letra ñ o símbolos especiales.
- Las variables pueden estar formadas por más de una palabra, pero no pueden estar separadas por espacios sino por barra baja (_). Ejemplo: Sensor_de_temperatura.

Las variables se pueden clasificar según el tipo de variable y según el área de uso. El concepto área de uso hace referencia a las zonas del programa en las que puede ser utilizada una variable. Atendiendo a este tipo de clasificación diferenciamos dos tipos: globales y locales. Una variable es global cuando se puede utilizar en cualquier parte del programa y es local cuando solo se puede utilizar en una parte concreta del programa.

Las variables globales son aquellas que se declaran fuera de cual-



quier estructura (de momento solo conocemos el void loop y void setup) ❸ . Por eso podemos utilizar esa variable dentro del void loop, a pesar de no estar declarada en esa parte del programa.

Hay que tener en cuenta que una variable se puede declarar en cualquier parte del programa. En consecuencia, si se declara dentro de una estructura se conoce como variable local ❹ .

❸

```
//Declaración de la variable
boolean pulsador = 0; // boolean pulsador;
void setup()
{
  //configurar el pin como salida
  pinMode(13, OUTPUT);
  //Nos aseguramos de que el led empieze apaga
  digitalWrite(13, LOW);
  //Pin 12 como entrada--
  pinMode(12, INPUT_PULLUP);
}
void loop()
{
  pulsador = !digitalRead(12);
  digitalWrite(13, pulsador);
}
```

❹

```
void setup()
{
  //configurar el pin como salida
  pinMode(13, OUTPUT);
  //Nos aseguramos de que el led empieze apagado
  digitalWrite(13, LOW);
  //Pin 12 como entrada--
  pinMode(12, INPUT_PULLUP);

}
void loop()
{
  //Declaración de la variable
  boolean pulsador = 0; // boolean pulsador;
  pulsador = !digitalRead(12);
  digitalWrite(13, pulsador);
}
```

IMPORTANTE

En caso de declarar una variable dentro de una estructura, esta solo podrá ser utilizada en esa estructura y en ningún caso en otra.

De tratar de hacerlo y cargar el programa, el IDE no nos lo permitirá y no mostrará un aviso ❺ .

❺

Comunicación Serie

IMPORTANTE

La herramienta del Monitor Serie no solo nos resultará útil para la transferencia de datos. Si la empleamos hábilmente, la podemos utilizar para encontrar errores, en muchas ocasiones nos encontraremos con que el Arduino no funciona y no tenemos claro de dónde puede venir el fallo; es en este momento donde nos puede resultar de gran utilidad el Monitor Serie.

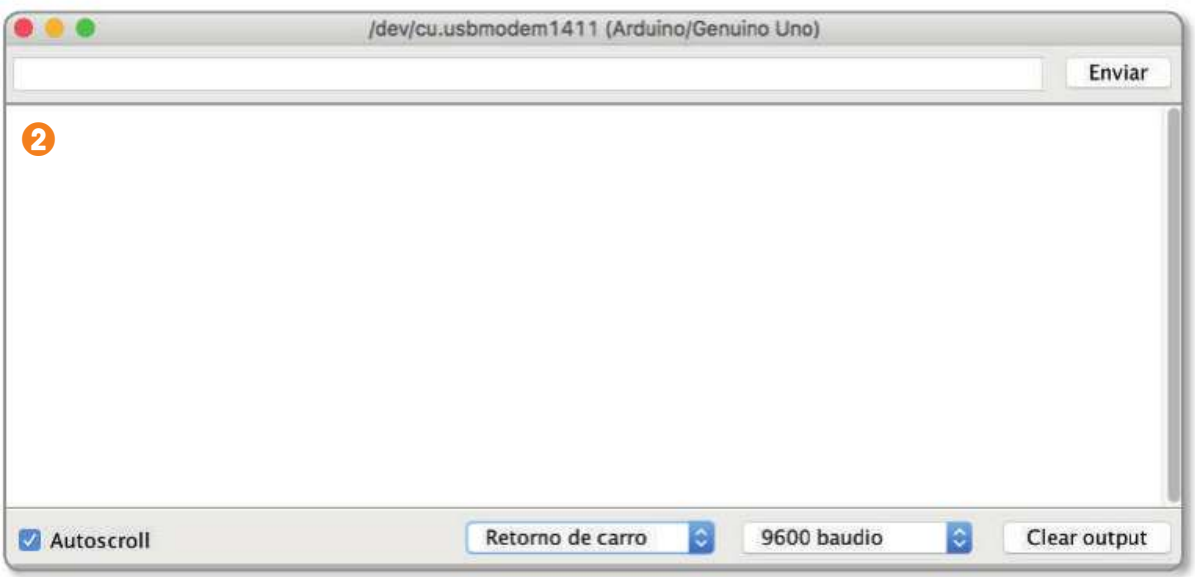
Podemos ir marcando ciertos mensajes a lo largo del programa que nos permitan identificar el origen del error.

Hemos mencionado el sistema binario y hemos hecho referencias a ceros y unos, pero como a pesar de ver que los programas puedan funcionar, este concepto se puede volver abstracto así que vamos a "darle forma".

La Comunicación Serie nos va a permitir enviar valores al IDE y, a través del interfaz del Monitor Serie, poder visualizarlos.

1. Al seleccionar esta herramienta se abrirá una ventana con un interfaz simple ❶ .
2. Este interfaz nos permitirá enviar órdenes o información al Arduino gracias al campo de texto que tenemos en la parte superior del Monitor ❷ .
3. En la disposición de fondo blanco (debajo del campo de texto) iremos visualizando todos los datos que nos envíe el Arduino.
4. En la parte inferior de este Monitor tenemos los controles para poder configurar su funcionamiento.
5. El Autoscroll, si lo tenemos seleccionado visualizaremos siempre el último dato recibido.
6. A la derecha disponemos de un botón que permite "borrar" todos los datos visibles.
7. Posiblemenete el apartado más importante es el que hace referencia a la velocidad de transferencia de datos, que se configura con el segundo control empezando por el margen derecho.
8. Si lo seleccionamos se abrirá un desplegable ❸ con todas las velocidades estándar para que seleccionemos en cada momento con cual necesitamos que trabaje.
9. La Comunicación Serie se realiza entre dos dispositivos, por lo tanto los dos deben enviar y recibir datos a la misma velocidad .
10. En conclusión, si el Monitor Serie se configura a una velocidad de 9600 bits/s, el Arduino deberá estar configurado a la misma velocidad.


1


/dev/cu.usbmodem1411 (Arduino/Genuino Uno)
Enviar
2
Autoscroll
Retorno de carro
9600 baudio
Clear output


9600 baudio
3
✓ 9600 baudio
19200 baudio
38400 baudio
57600 baudio
74880 baudio
115200 baudio
230400 baudio

Monitorización

IMPORTANTE

Debemos tener precaución a la hora de usar el Monitor Serie puesto que, cada vez que abrimos la comunicación desde el IDE (es decir, cada vez que abrimos la ventana del monitor), el Arduino se resetea.

Esto, como ya sabemos, implica que se vuelva a ejecutar el void setup.

En este capítulo no profundizaremos demasiado en el funcionamiento de la comunicación serie, buscaremos más lo práctico y útil. Lo que se pretende es disponer de una herramienta para estos primeros conceptos con el Arduino, que nos permita entender mejor lo que estamos haciendo y llegado el momento saber qué está pasando "dentro del Arduino".

Para trabajar con esta comunicación emplearemos diferencies instrucciones, todas ellas comenzarán con el "prefijo" Serial.

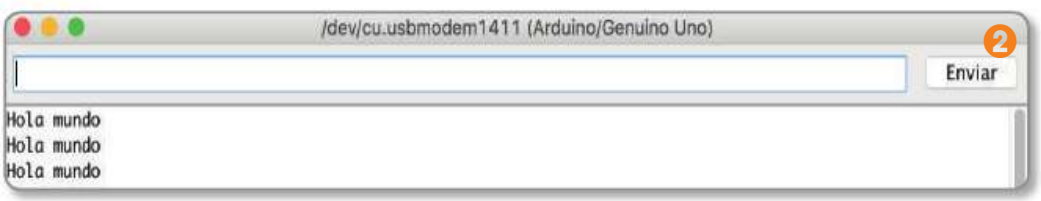
1. Debemos configurar e inicializar la Comunicación Serie, para ello utilizaremos la instrucción Serial.begin(); . Entre paréntesis debemos definir la velocidad de la transferencia de datos que deberá ser la misma que tenga el Monitor Serie.
2. En la programación de Arduino disponemos de varias instrucciones para enviar información, pero nos decantaremos por Serial.println();
3. Esta instrucción permite enviar información seguida de un salto de línea, que, si hacemos un símil con la escritura de unas líneas en un Word, equivale a escribir la información y pulsar la tecla Enter, para que en la siguiente escritura empecemos en la siguiente línea ❶ .
4. Con esta instrucción todos los datos que enviemos se irán visulizando en diferentes líneas ❷ .
5. También podemos enviar el valor devuelto por una instrucción, que en el caso de una entrada nos permite "visualizar" sus dos estados (0 ó 1) ❸ .

Con estos simples pasos podemos monitorizar el funcionamiento del Arduino. Esto funcionaría igual en cualquier entorno de programación o mismamente si desarrollamos un entorno SCADA o IHM.

De cara a la programación del Arduino no sería necesario modificar nada, puesto que el funcionamiento consiste en enviar información por su puerto a una determinada velocidad y lo que ya nos podemos imaginar es que tanto un entorno SCADA como IHM deben tener la misma velocidad de puerto y atender al puerto USB que se encuentre conectado el Arduino (aparte del resto de programación de captura y procesamiento de información).

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  Serial.println("Hola mundo");
  delay(1000);
  //Visualizaremos este mensaje cada segundo por el monitor

}
```

1



2

```
void setup() {
  Serial.begin(9600);
  pinMode(12,INPUT_PULLUP);
}

void loop() {
  Serial.println(digitalRead(12));
}
```

3

Transistores en CI

IMPORTANTE

Uno de los principales usos que podemos hacer con un transistor y un Arduino es aumentar de forma indirecta la potencia que podemos controlar por cada pin del Arduino (límite de 20mA y 5V), ya que podemos controlar una potencia superior conectada a las patillas del transistor mediante una señal de baja potencia generada por el Arduino.

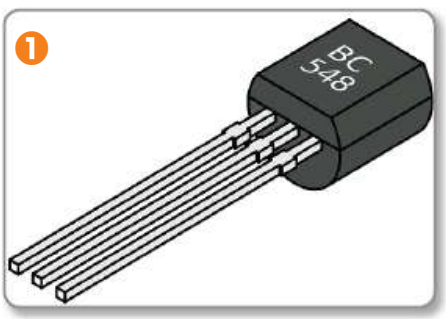
Lo que hemos hecho hasta el momento han sido una serie de operaciones básicas (trabajar con las entradas y salidas digitales). Esto es posible por la programación realizada sobre el MCU; en su momento definíamos un MCU como un circuito integrado (CI) que puede ser reprogramado.

Pero dentro de los CI no solo tenemos los MCU´s; es por eso que haremos un paréntesis en el aprendizaje de Arduino y revisaremos los CI más básicos. Este estudio lo hacemos por diversos motivos:

- Nos podemos encontrar en diferentes proyectos con CI.
- En según qué proyectos, posiblemente nos compense buscar una solución a través de otros CI que no sean un MCU.
- Nos permitirán entender mejor el funcionamiento de un MCU, puesto que podemos considerarlo como un componente elaborado a partir de la unión de CI más simples.
- Sirve como una introducción a la electrónica digital.

Pero entonces ¿por qué en el título del capítulo pone transistores? Eso es por que los transistores son el elemento base de los CI, y haremos un breve análisis sobre ellos 1 .

La principal función de un transistor es la de ampliación: permite pasar de una señal pequeña a una señal mayor en función de la polarización y ganancia del transistor. Polarización y ganancia son términos que pueden sonar difíciles, por lo que los estudiaremos mediante un caso práctico.



No profundizaremos demasiado en los transistores, los estudiaremos como si fueran interruptores (que es el uso más básico que tienen). Podemos considerar un transistor como un interruptor, como ya sabemos un interruptor permite abrir o cerrar un circuito. Los interruptores de nuestras casas nos permiten, mediante una mecanismo manual, cerrar o abrir el circuito para que se encienda una bombilla.

El transistor es igual, la diferencia es que en vez de controlarlo por una acción manual lo hacemos mediante una señal eléctrica.

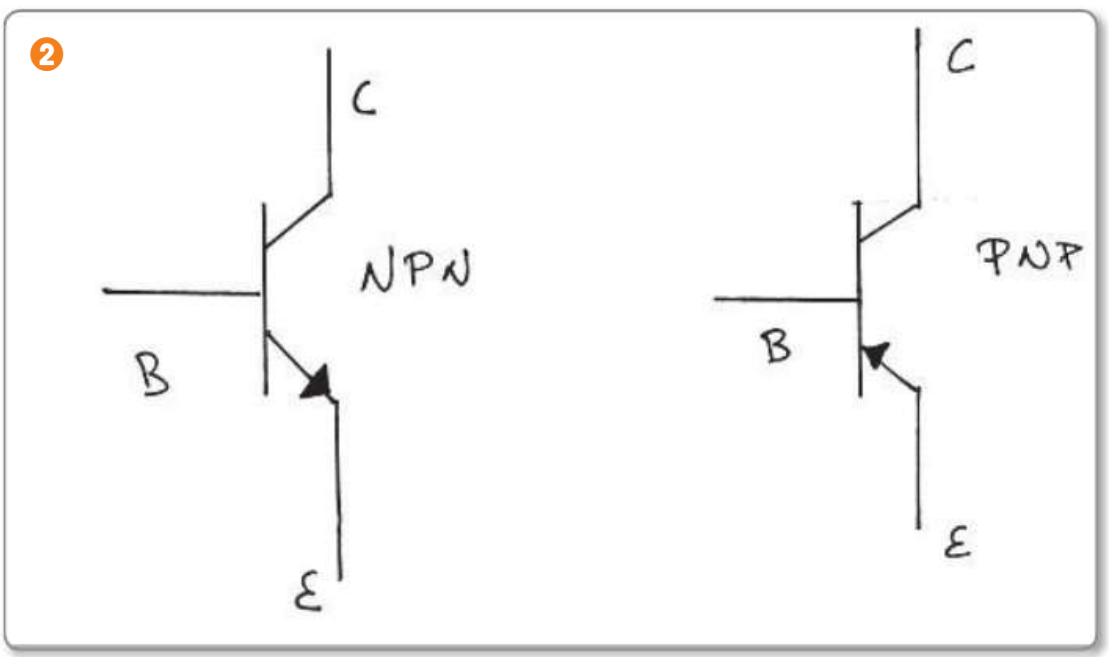
Antes de continuar, debemos considerar que disponemos de diferentes tipos de transistores. Los más empleados son los BJT y los MOSFET (o MOS), las diferencias entre ellos vienen de su fabricación, lo que implica que su implementación y características de uso son diferentes.

Los transistores más comunes en la fabricación de CI son los MOSFET, pero analizaremos los BJT.

Aparte, dentro de cada tipo de transistor nos encontramos con unas siglas PNP y NPN, esto simplemente nos indica el sentido de las tensiones e intensidades de operación del transistor.

Los transistores tienen 3 patillas, una de ellas es la que nos permite controlar mediante una señal si se abre o cierra el circuito conectado a las otras dos patillas. Según el transistor que utilicemos este tendrá una representación diferente, así como una nomenclatura diferente para cada patilla; en el caso de los BJT será el siguiente ❷ :

❷



Polarización y curvas características del BJT

Un transistor BJT puede operar en tres modos:

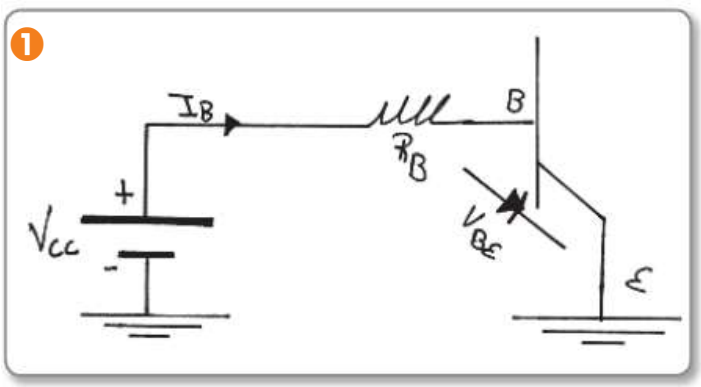
- Conducción, lo que se conoce como que trabaje en su zona activa. Este modo es empleado para la amplificación de señales.
- Saturación y corte, en estos dos modos podemos considerar que el transistor funciona como un interruptor, abriendo o cerrando un circuito.

Los modos de funcionamiento en los que nos centraremos serán los de saturación y corte, es el uso más "sencillo" que le podemos dar a un transistor y el primer uso que le podemos dar es el de aumentar la potencia de los pines del Arduino.

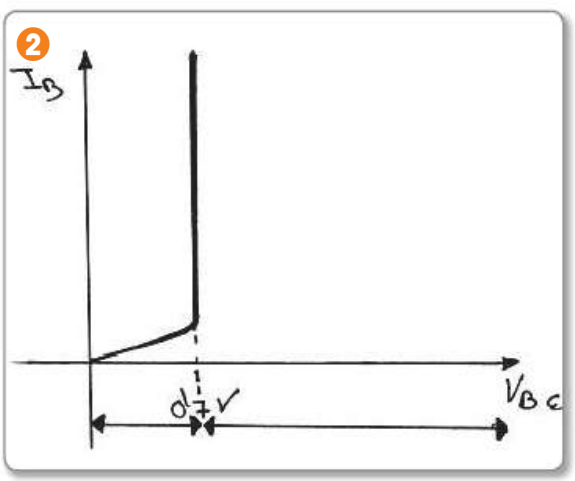
Puesto que con una señal de pocos mA podemos controlar una intensidad mayor.

1. De lo que se trata es de controlar un transistor desde el Arduino, y obviamente la única forma será mediante una señal de una determinada tensión e intensidad generada por uno de los pines de Arduino.
2. Esta señal polarizará el transistor para que entre en uno de sus tres modos.
3. Para ello necesitamos un circuito de polarización, que estará formado por señales fuentes de alimentación y resistencias 1 .
4. Para dimensionar el circuito de polarización debemos entender las curvas de funcionamiento de un transistor.

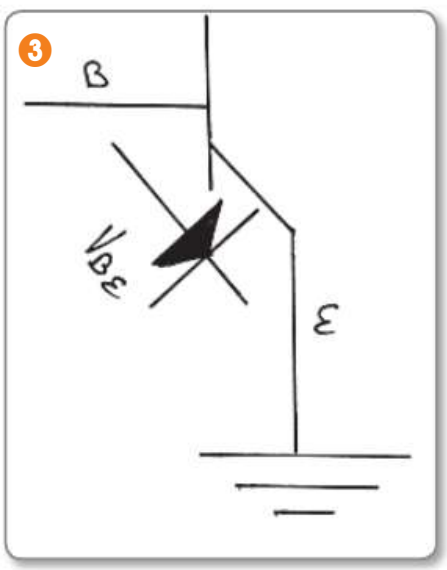
1

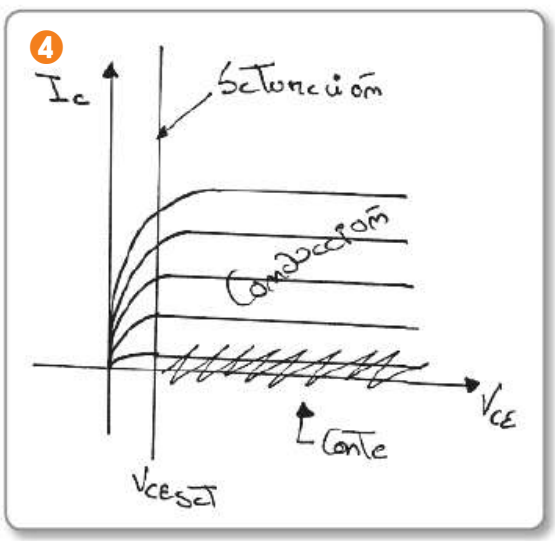


2



5. De la curva de entrada ❷ deducimos que so la tensión que "surge" entre base y emisor es como mínimo Vbeon (tendrá un valor diferente según el transistor) el transistor se encontrará en conducción o en saturación, y si es inferior se encontrará en corte (con lo que el transistor se comporta com un interruptor abierto).
6. La tensión Vbe ❸ viene de una especie de diodo que viene de la unión base-emisor, por lo tanto si no alcanzamos la tensión de trabajo del diodo este no permite la circulación de corriente.
7. Conseguiremos esa tensión de trabajo mediante el circuito de polarización.
8. Si polarizamos el transistor para que no se encuentre en corte debemos estudiar si está polarizado para trabajar en activa o saturación.
9. Para ello nos vamos a la curva de salida, de la cual obtenemos las condiciones para ambos casos ❹.
10. En ella podemos ver una segunda tensión Vce, esta tensión es similar a la tensión que estudiamos en la primera curva que surgía entre base-emisor, pero en este caso surge en las uniones colector-emisor.
11. Si la tensión es inferior a Vcesat(según el transistor) y el transistor se encuentra en saturación (interruptor cerrado) y si es superior en conducción (amplificación de señales).





Circuito BJT

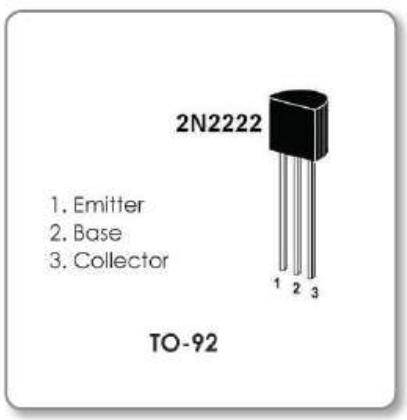
El uso más sencillo de un transistor será mediante circuitos integrados o, como veremos en próximos capitulos, mediante puertas lógicas.

En este capítulo veremos un caso sencillo de uso de un transistor, en concreto el transistor BJT.

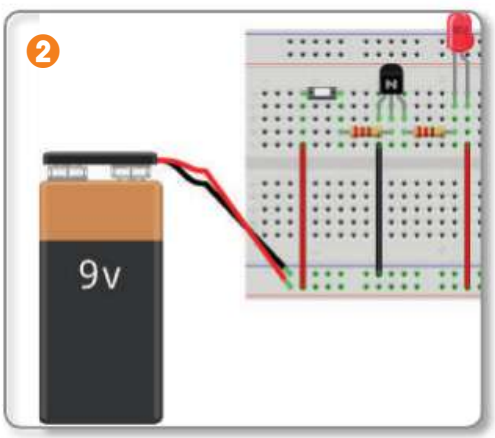
1. Es necesario identificar las patillas del transistor, para ello lo pondremos con la parte plana hacia nosotros y las patillas nos quedarán de izquierda a derecha de la siguiente forma Emisor-Base-Colector ❶ .
2. En la base conectamos una resistencia de 1kΩ y un botón y a continuación a la pila de 9V.
3. El emisor lo conectaremos directamente a tierra.
4. Al colector conectamos una resistencia de 470Ω un led y la pila de 9V.
5. El objetivo es que cuando se pulse el botón el transistor deje de estar en corte y permita la circulación de intensidad entre Colector-Emisor y así se encienda el led .Cuando no se pulse ,el transistor entrará en corte y no circulará corriente ❷ .
6. Para analizar su comportamiento es necesario dibujar el esquemático ❸ .
7. Tenemos dos posibilidades, que se pulse el botón y el transistor se polarice o bien en conducción o saturación o que no se pulse y el transistor se polarice en corte.
8. Debemos analizar el circuito resultante entre Base-Emisor ❹ .
9. En la unión Base-Emisor dibujaremos un diodo a modo de simbolizar la caída de tensión necesaria para que el transistor no entre en corte.
10. La tensión que necesitamos en este caso es de al menos 0,5V, si la unión Base-Emisor alcanza ese valor el transistor saldrá de la zona de corte.

En caso de no pulsarse el botón la diferencia de tensión en la unión será inferior a lo que se conoce como Vbeon, por lo que entraría en corte lo que concluiría con que no se produciría una “unión“ entre las patillas de Colector-Emisor y el diodo no se encendería.

❶



❷

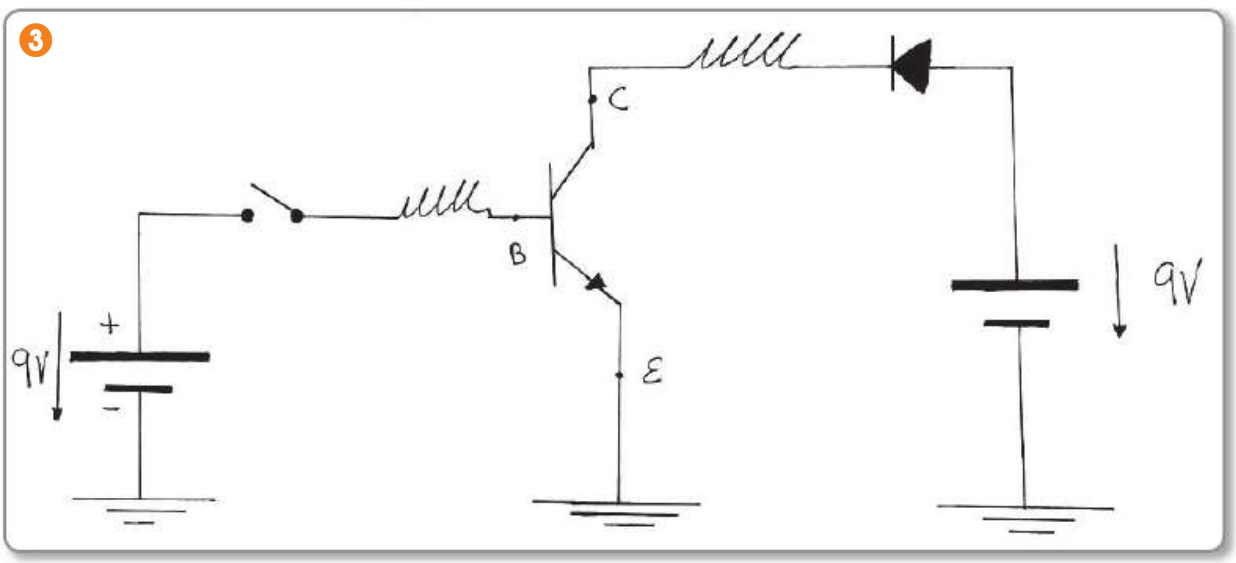


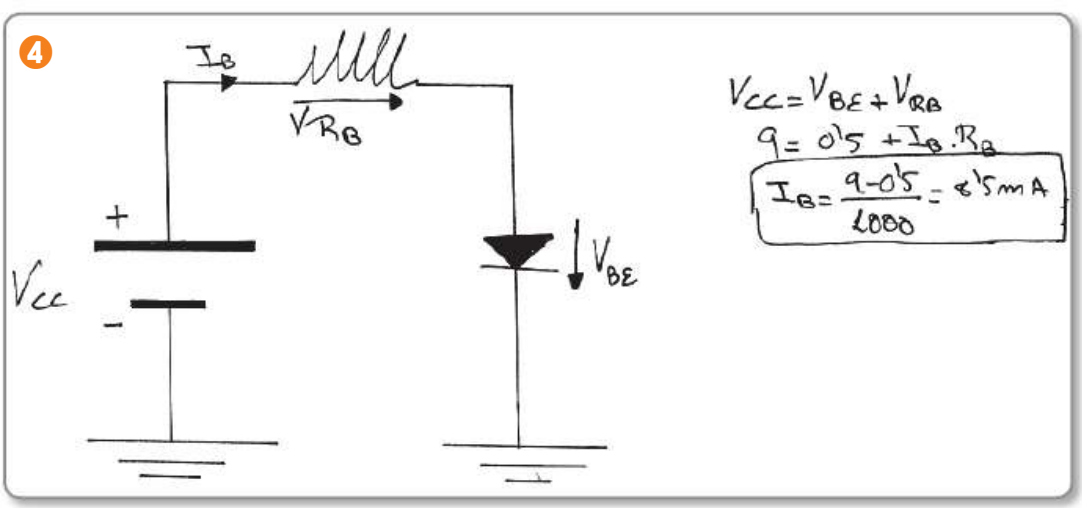
Pero si pulsamos el botón, la unión Base-Emisor dispondría de la diferencia de tensión necesaria para entrar en conducción o saturación , por lo tanto se “unirían” las patillas Colector-Emisor encendiéndose así el led.

En este caso nos quedaría por comprobar si entraría en saturación o en conducción, pero esto implicaría realizar aún más cálculos y pasos. Y como el objetivo es tener una base para en posteriores capítulos utilizar puertas lógicas con esto será necesario.

IMPORTANTE

Las puertas lógicas se fabrican con los transistores tipo MOSFET, pero a pesar de eso lo visto en este capítulo nos vale como idea para entender cómo funcionan las puertas lógicas.





Puertas lógicas

IMPORTANTE

En estos esquemas se ha utilizado un nuevo componente, que es alimentador de protoboards (ywrobot) 1 . Gracias a este hardware podemos alimentar de forma segura los circuitos en la protoboard, tiene salidas a 3,3 y 5V un límite de 700mA, y por el conector Jack tenemos un límite de 12V.

Los transistores resultaron en su momento una revolución (a día de hoy también, todos los dispositivos electrónicos están formados a partir de transistores), permitieron sustituir relés y válvulas que eran la base de los CI, con lo que se consiguió una mejora de sus prestaciones.

Si volvemos al capítulo 27 , en donde invertíamos el valor de una salida, vemos que es un caso muy sencillo y parece una aplicación muy sencilla para tener que emplear un Arduino. Este caso resulta idóneo para integrar la primer puerta lógica.

- Puerta NOT (o inversor).

Estas puertas tienen dos patillas (entrada y salida) y su función es sacar el valor inverso de la entrada por su salida. Es decir si entran 5V saca 0V y viceversa.

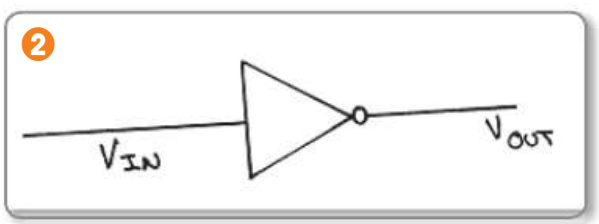
Una puerta NOT se representa con un triángulo junto con un círculo en su salida. Los círculos en electrónica digital significan inversión 2 .

Internamente una puerta NOT está formada por dos transistores MOSFET en serie, los cuales se polarizan a la inversa, (su funcionamiento es similar al BJT y también disponen de 3 patillas) 3 .

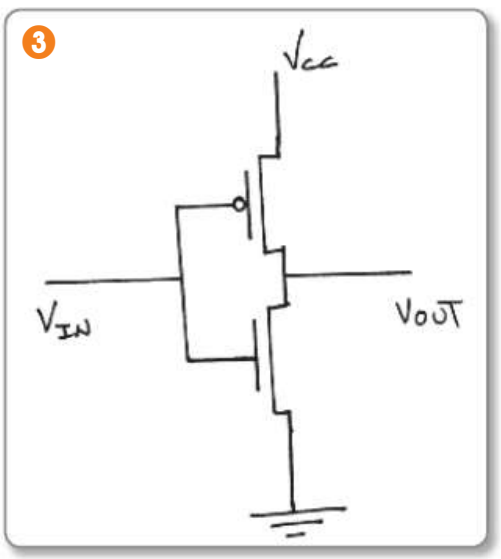
1. Si la Vin es de 0V, el transistor que se “activa“ es el superior conectando Vout a 5V.

1

2

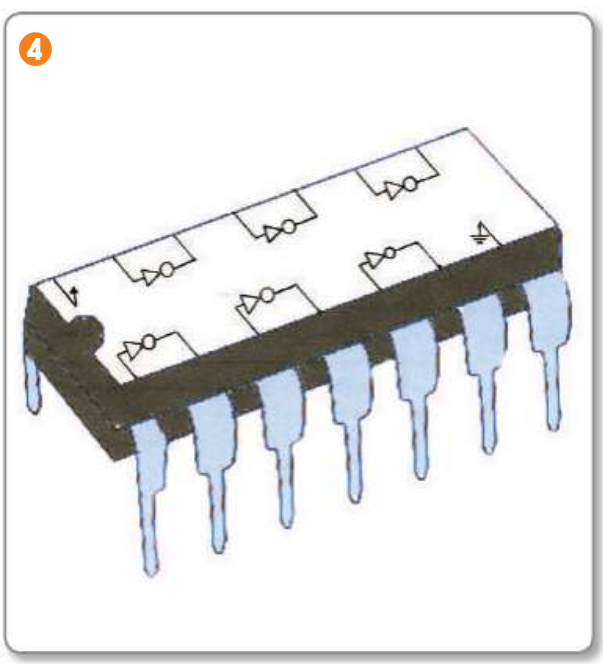


3

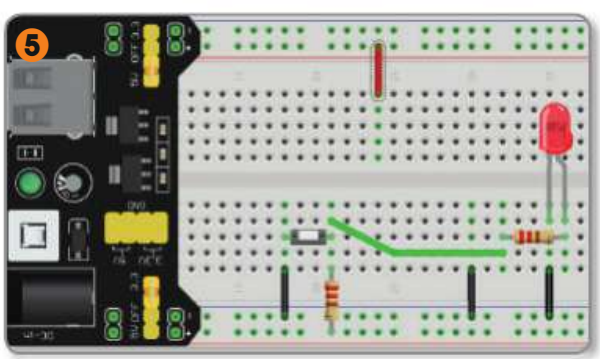


2. Y si en caso contrario Vin es igual a 5V se “activa“ el transistor inferior conectado la salida a 0V.
3. Para este caso emplearemos el CI 74HC04, que integra 6 inversores 4 .
4. Es importante tener en cuenta el patillaje, puesto que necesitamos alimentar el CI. Todos los CI tiene una pequeña muesca en su encapsulado que junto con el esquema del CI conocemos la función de cada patilla.
5. Para entender mejor este caso vamos a conectar la salida de un botón Pullup a un led, con lo que comprobamos que el led se enciende cuando no se pulsa el botón y se apaga cuando se pulsa 5 .
6. Gracias al inversor podemos invertir el funcionamiento y con el mismo conexionado del botón y led hacer que se encienda cuando se pulsa el botón y se apague cuando no se pulse 6 .

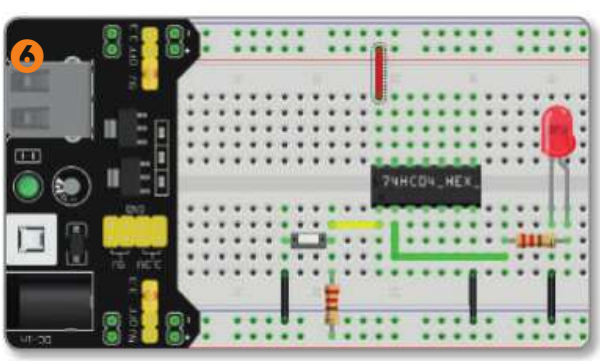
4



5



6



Operador y puerta lógica AND

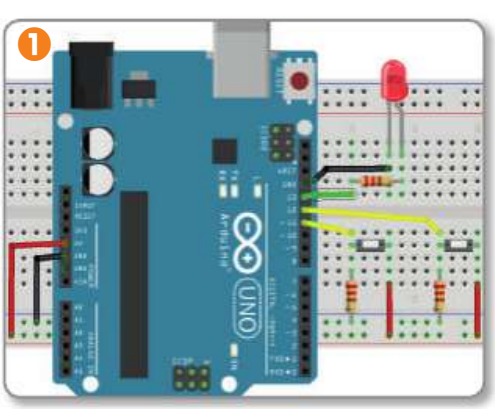
El operador && permite operar con bits, todo operador devuelve un resultado; como estamos trabajando con bits devolverá un 0 o un 1.

- El operador && para el sistema binario es algo así como un multiplicador, por lo tanto si multiplicamos dos bits y uno de ellos o los dos vale 0 el resultado será 0, y si los dos tienen el valor de 1 el resultado será 1.
- Podemos probar su funcionalidad en un caso práctico, conectar dos botones y un led al Arduino ❶ .
- El led se encenderá solo en el caso de que los dos botones se pulsen. Con que uno de los dos no se encuentre pulsado el led deberá apagarse ❷ .
- Podemos concatenar tantos && como necesitemos, por ejemplo si queremos que el led solo se encienda si se han pulsado 4 botones, pues no hay más que ir añadiendo al digitalWrite && y las lecturas de las entradas correspondientes.
- Por si nos resulta más fácil de recordar podemos sustituir el operador && por la palabra reservada and.

La programación no resulta complicada, pero disponemos de una puerta lógica que ya trae esta funcionalidad implementada: la puerta AND. Al igual que la puerta NOT existe un símbolo para esquematizar una puerta AND ❸ .

- Utilizaremos el circuito integrado 74HC08, que integra 4 puertas AND ❹ .
- Una puerta AND está formada por 4 transistores ❺ , dos de ellos se encuentran conectados en serie y se "activan" si las dos entradas se encuentran a 5V, si es así Vout se conecta a los 5V.
- En caso de que alguna o las dos entradas se encuentren a 0V se activa uno de los transistores inferiores conectados en paralelo con lo que Vout se conecta a 0V.
- Podemos repetir el caso anterior, en el que el led solo se encienda cuando los dos botones están pulsados ❻ .
- Y al igual que ocurría con el operador && podemos hacer que el led se encienda si por ejemplo hay 4 botones pulsados pero para ello debemos combinar varias puertas AND entre sí ❼ .

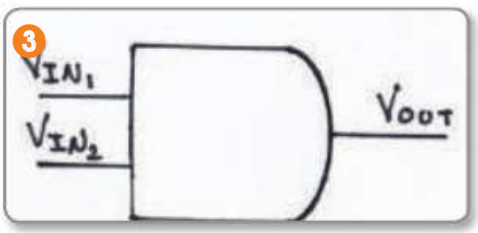
❶



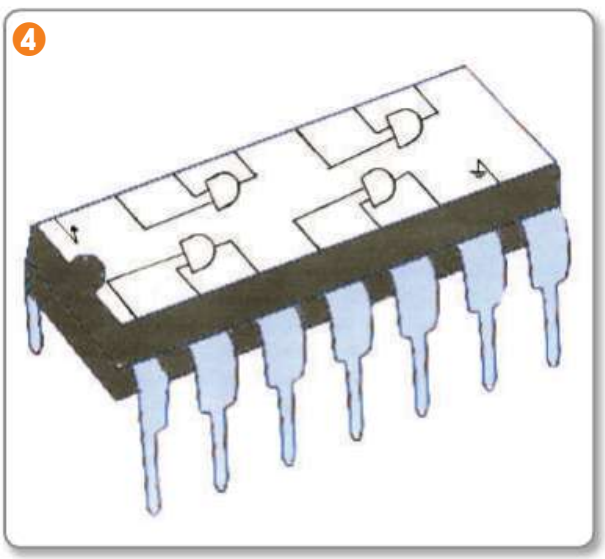
❷

```
void setup() {
  pinMode(13, OUTPUT);
  pinMode(12, INPUT);
  pinMode(11, INPUT);
}

void loop() {
  digitalWrite(13, digitalRead(12) && digitalRead(11));

}
```

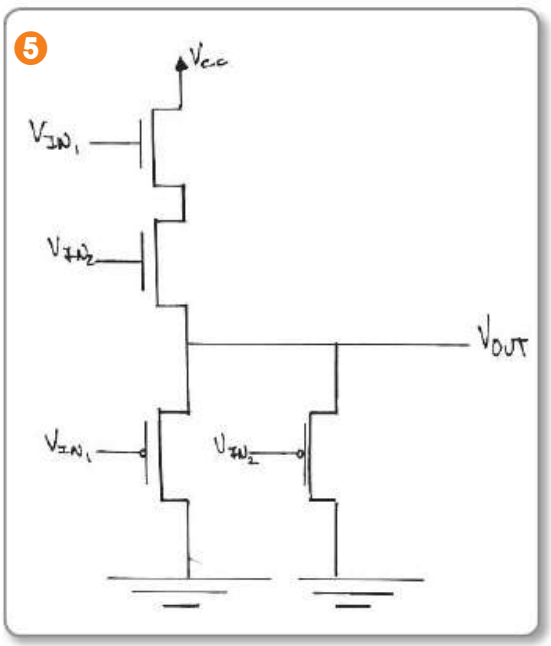
3



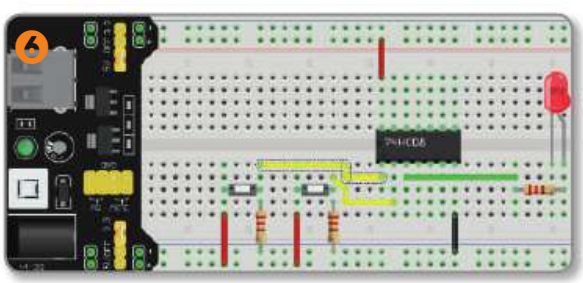
4



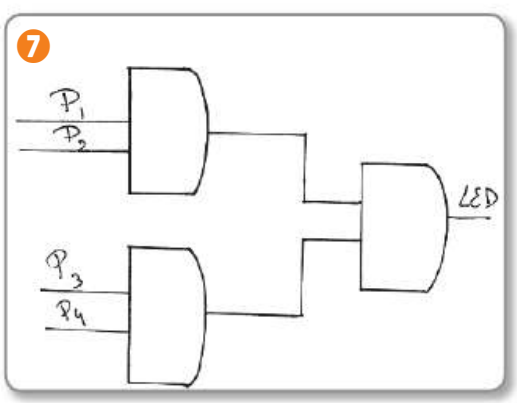
5



6



7



Operador y puerta lógica NAND

Al igual que ocurre en las operaciones matemáticas que podemos agrupar operaciones entre paréntesis, a la hora de realizar operaciones con los operadores de bits podemos hacer lo mismo

- Un operador NAND no es más que la inversa de un operador AND.
- Como es la inversa utilizamos el operador !, para invertir el resultado, pero antes de eso necesitamos agrupar todas las variables entre paréntesis.
- Si cargamos el programa y los probamos, veremos que el led solo se apagará cuando los dos botones están pulsados ❶.

Si combinamos de forma correcta las puertas lógicas que llevamos visto, podemos conseguir crear una puerta lógica NAND .

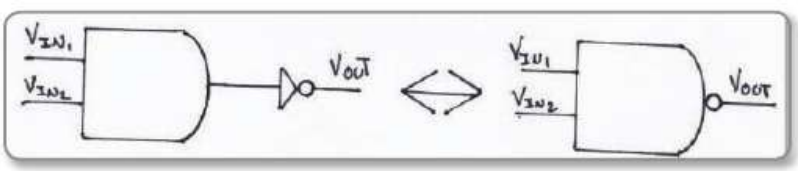
Si a la salida de una puerta AND conectamos una puerta NOT, la combinación de ambas sería una puerta NAND ❷.

Una solución más sencilla sería implementar una puerta lógica NAND.

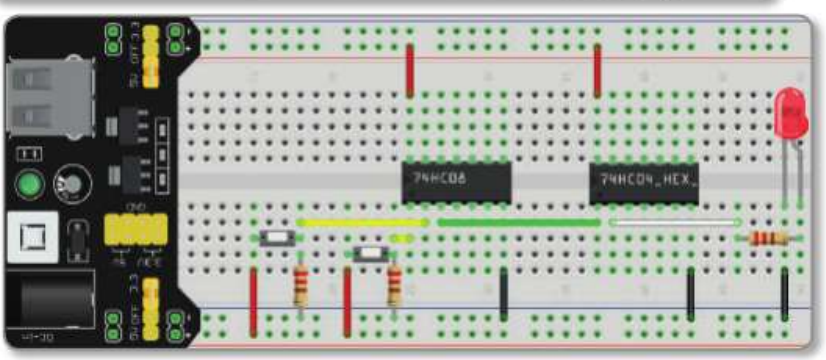
❶

```
void setup() {
  pinMode(13, OUTPUT);
  pinMode(12, INPUT);
  pinMode(11, INPUT);

}

void loop() {

  digitalWrite(13, !(digitalRead(12) && digitalRead(11)) );

}
```

❷

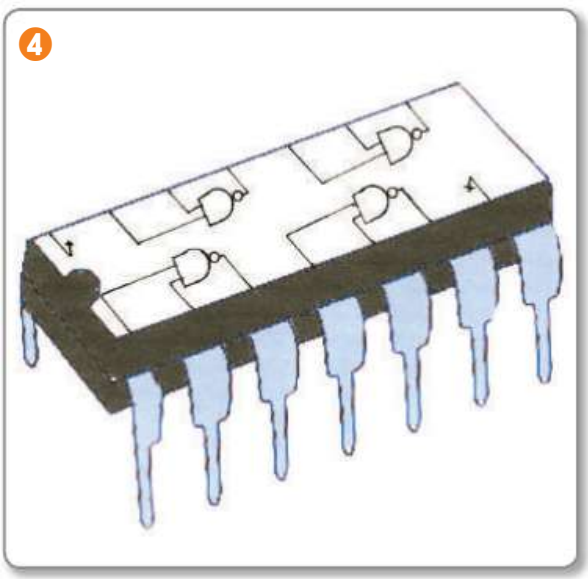


❸

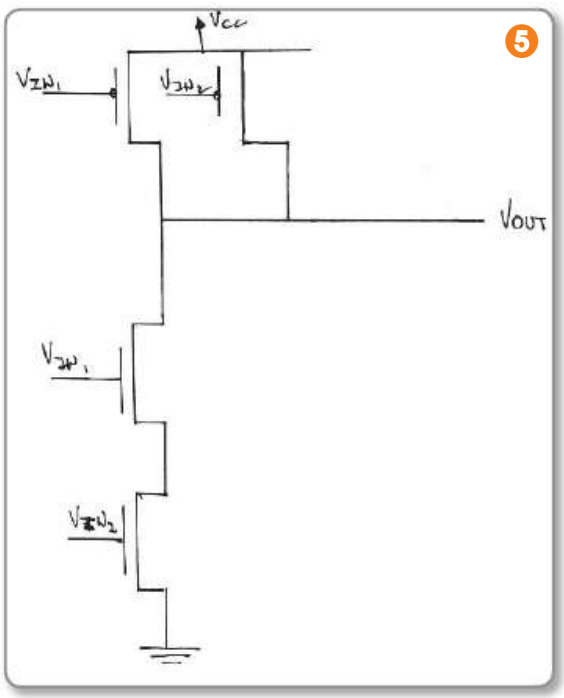


- El CI que utilizaremos es el 74HC132, que integra 4 puertas NAND 3 .
- Una puerta NAND está formada por 4 transistores 4, dos de ellos se encuentran conectados en serie y se "activan" si las dos entradas se encuentran a 5V, si es así Vout se conecta a los OV.
- En caso contrario se "activa" uno o los dos transistores conectados en paralelo, conectando así Vout a 5V 5.
- Si sustituimos los dos CI integrados anteriores, por el 74HC132 conseguiremos el mismo funcionamiento 6.

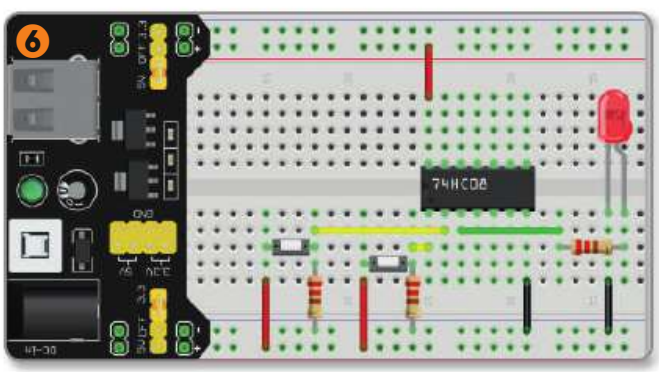
4



5



6



Operador y puerta lógica OR

Una operación OR ya sea por programación o por un CI, implica que el resultado de 2 entradas o más será (en binario) igual a 1 siempre y cuando alguna de las entradas tenga el valor en binario de 1 (señal 5V).

- En cuanto a la programación necesitamos un nuevo operando, que o bien podemos utilizar || o la palabra reservada OR ❶ .
- Si probamos este caso, veremos que el led se encenderá siempre y cuando se pulse al menos un botón.
- Podemos concatenar tantos operadores || como necesitemos.

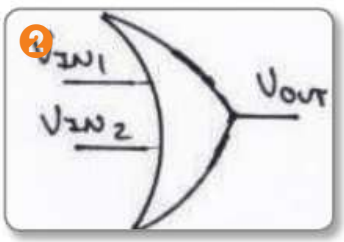
La puerta lógica OR cumple la misma función que la programación que acabamos de realizar, con que alguna de sus entradas se encuentre a 5V el estado de la salida también será de 5V.

- La puerta lógica OR dispone de su propio símbolo ❷ .
- En este caso utilizaremos el CI 74HC32 ❸, que integra 4 puertas lógicas OR.
- Una puerta lógica OR está formada por 4 transistores ❹ asociados a sus dos entradas.
- Dos de esos transistores se encuentran conectados en paralelo, se activan en caso de que la entrada correspondiente se encuentre a 5V, consiguiendo que se conecte Vout a 5V.
- Los otros dos transistores se encuentran conectados en serie, en caso de que las dos entradas se encuentren a 0V, a través de estos transistores se conecta Vout a 0V.
- Si implementamos este nuevo CI, al igual que ocurría por programación el led se encenderá siempre y cuando al menos uno de los dos botones se pulse ❺ .

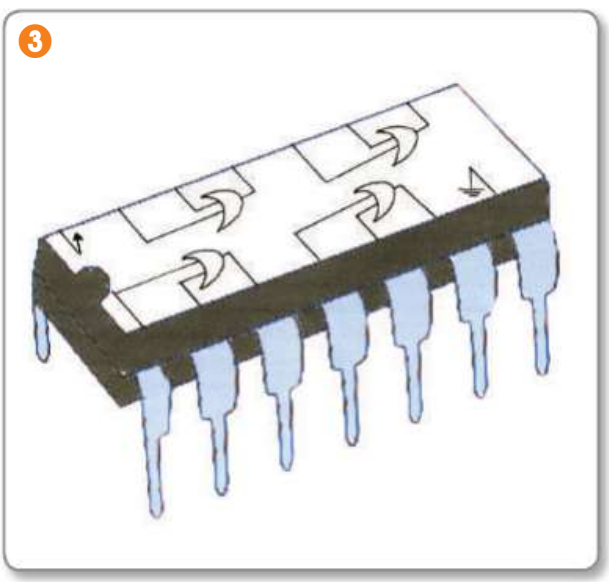
❶

```
void setup() {
  pinMode(13, OUTPUT);
  pinMode(12, INPUT);
  pinMode(11, INPUT);

}

void loop() {

  digitalWrite(13, digitalRead(12) || digitalRead(11) );

}
```

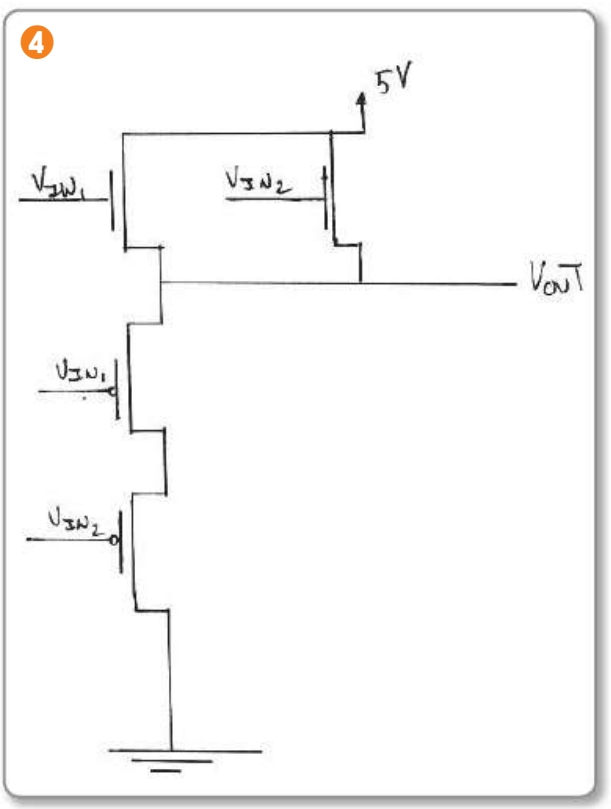
2



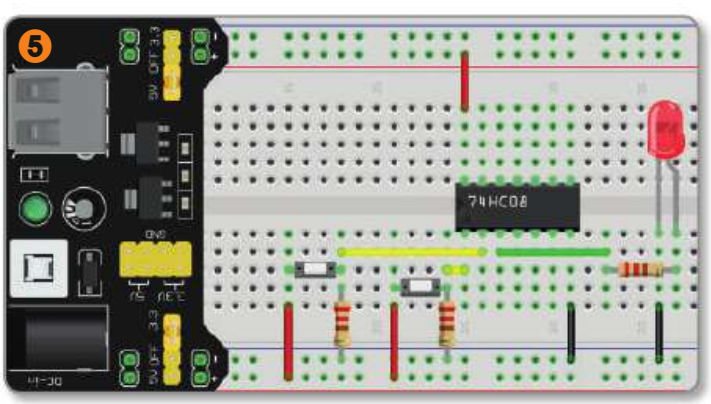
3



4



5



Operador y puerta lógica NOR

El funcionamiento tanto por programación como por CI de una puerta NOR es inverso al de una puerta OR.

- En el caso de la programación no tenemos más que agrupar los operadores entre paréntesis e invertir el resultado con el operador !
- Con este nuevo programa ❶, el led se encenderá cuando no se pulse ningún botón y en caso de que alguno de ellos o los dos se pulsen se apagará.

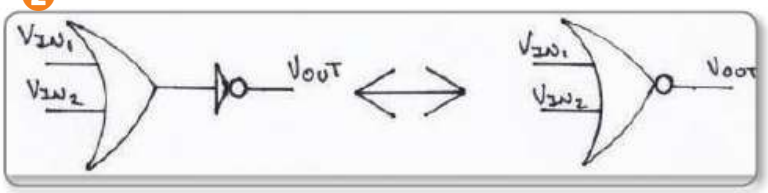
En cuanto al CI, nos encontramos con un caso similar al de la puerta lógica NAND.

- Podemos conectar a la salida de una puerta OR una NOT ❷.
- En ese caso invertiremos el comportamiento del CI del caso anterior ❸.
- La otra solución sería conectar el CI que integre puertas NOR. En este caso usaremos el CI 74HC02 ❹, que integra 4 puertas NOR.
- Una puerta NOR está formada por 4 transistores ❺ asociados a sus dos entradas.
- Dos de ellos conectado en serie, los cuales si las dos entradas se encuentran a 0V se "activan" y conectan Vout a 5V.
- Los otros dos transistores se encuentran conectados en paralelo y con que alguna de las entradas se encuentre a 0V, Vout se conecta a 0V.
- Con que uno de los dos botones se pulse el led se encenderá ❻.

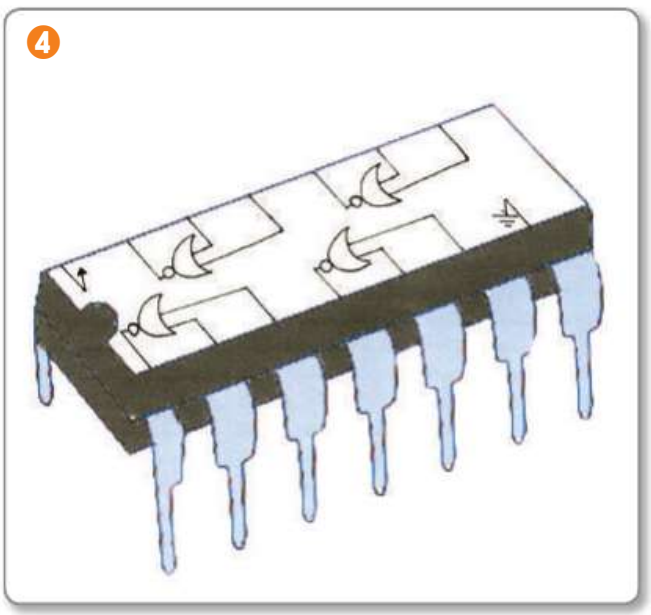
❶

```
void setup() {
  pinMode(13, OUTPUT);
  pinMode(12, INPUT);
  pinMode(11, INPUT);

}

void loop() {

  digitalWrite(13, !(digitalRead(12) || digitalRead(11)) );

}
```

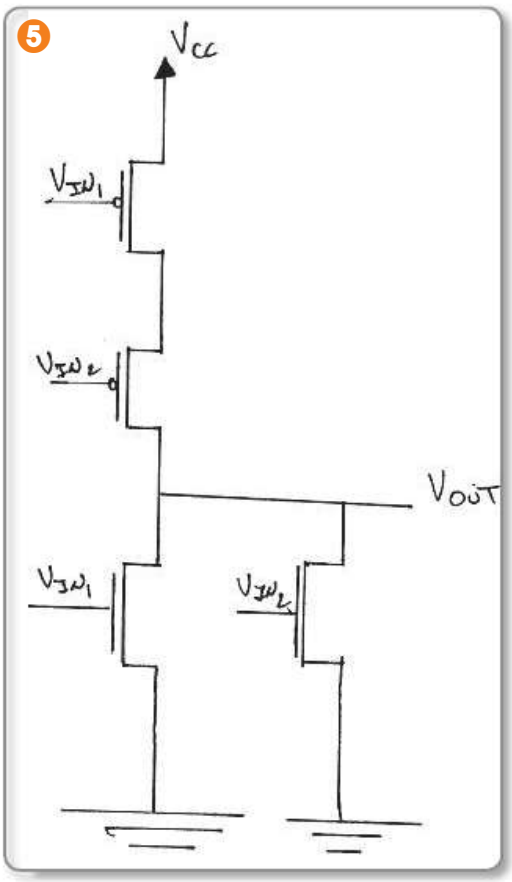
2


VIN1
VIN2
VOUT
VIN1
VIN2
VOUT

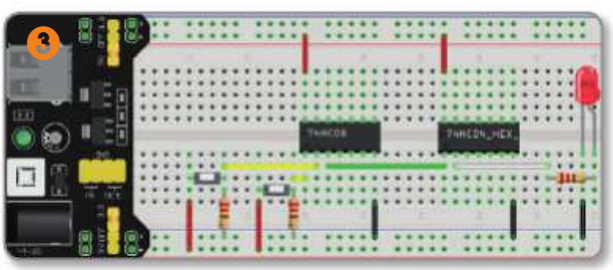
4



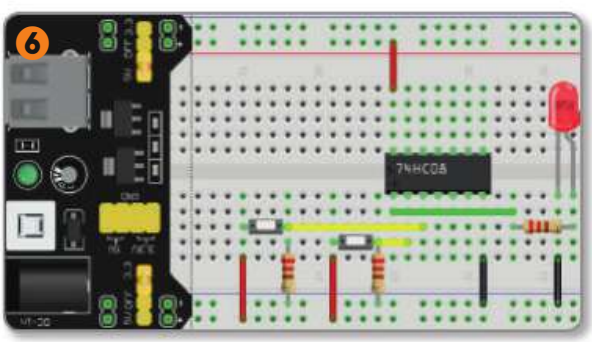
5


VCC
VIN1
VIN2
VOUT
VIN1
VIN2

3



6


74HC08

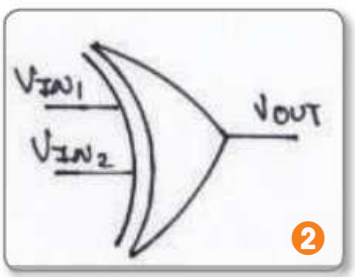
Operador y puerta lógica XOR

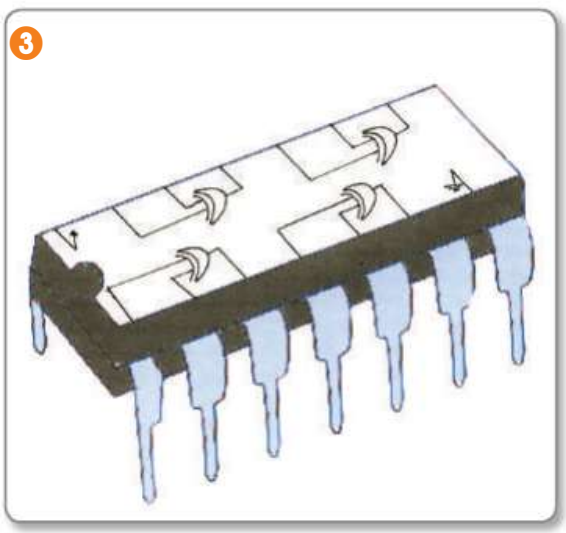
Este nuevo operador es conocido como OR-exclusiva, es un tipo especial de puerta u operador OR, que es capaz de diferenciar cuando dos entradas son iguales y cuando no.

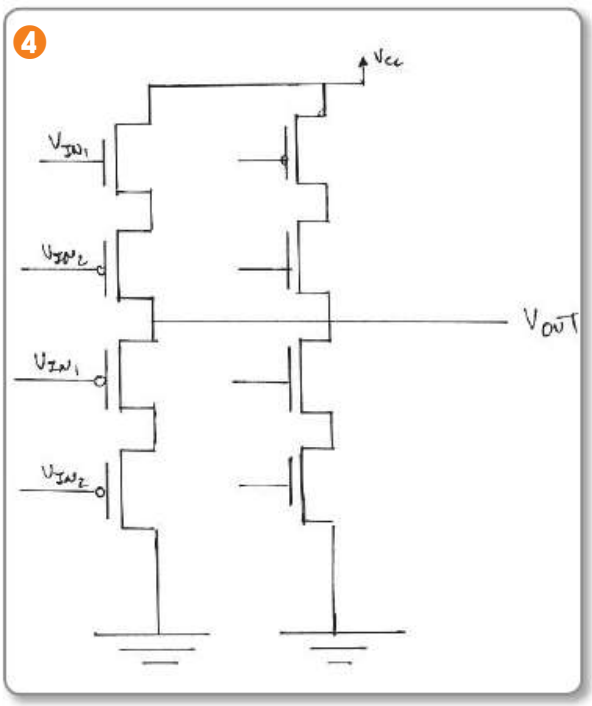
- Se necesita un nuevo operador : ^
- Ante dos bits diferentes (0 y 1) el resultado en binario es 1 y por el contrario si son iguales (0 y 0 o 1 y 1) el resultado es 0.
- Si cargamos este nuevo programa ❶ el led se apagará en el caso que los dos botones se encuentren en el mismo estado, es decir, los dos pulsados o los dos sin pulsar.
- En el momento en que uno se pulse y el otro no el led se encenderá.
- La representación de una puerta XOR es similar a la de una OR ❷.
- En este caso utilizaremos el CI 74HC86 ❸, que integra 4 puertas XOR.
- Una puerta XOR está formada por ocho transistores ❹ asociados a sus dos entradas.
- Se conectan parejas en serie y entre parejas en paralelo.
- Las parejas de transistores de la parte superior del esquema se conectan intercambiando el nivel de cada entrada de tal forma que si las entradas son diferentes una de las dos parejas de transistores se "activa" conectando Vout a 5V.
- Las parejas de transistores de la parte inferior del esquema se conectan igualando el nivel de cada entrada de tal forma que si las entradas son iguales (o bien las dos 5V o las dos 0V) una de las dos parejas de transistores se "activa" conectando Vout a 0V.
- Con este circuito conseguimos el mismo funcionamiento que por programación ❺.

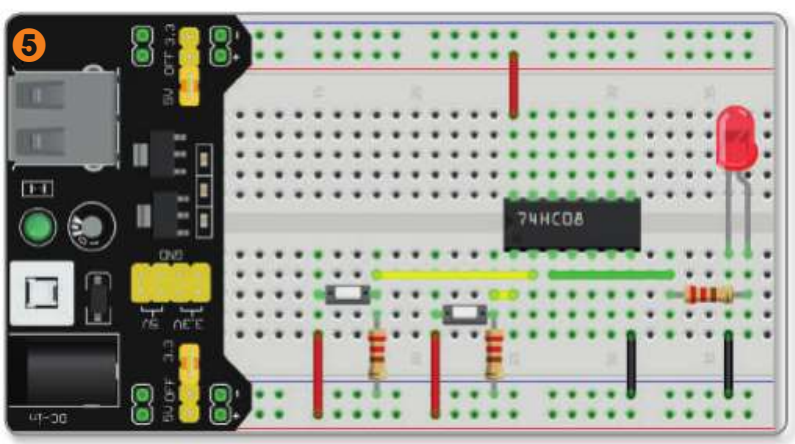
❶

```
void setup() {
  pinMode(13, OUTPUT);
  pinMode(12, INPUT);
  pinMode(11, INPUT);

}

void loop() {

  digitalWrite(13, digitalRead(12) ^ digitalRead(11) );

}
```


VIN1
VIN2
VOUT
2


3


4
Vcc
VIN1
VIN2
VOUT
VIN1
VIN2


5
74HC08

Operador y puerta lógica XNOR

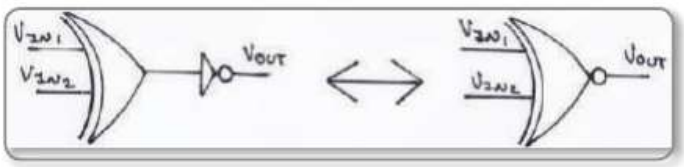
El funcionamiento de esta puerta lógica es inverso al de una puerta XOR.

- Agrupamos todos los operadores ^ entre paréntesis y negamos la operación (!).
- Ante dos bits diferentes (0 y 1) el resultado en binario es 0 y por el contrario si son iguales (0 y 0 o 1 y 1) el resultado es 1.
- Si cargamos este nuevo programa ❶ el led se encenderá en el caso que los dos botones se encuentren en el mismo estado, es decir, los dos pulsados o los dos sin pulsar.
- En el momento en que uno se pulse y el otro no el led se pagará.
- Podemos crear una puerta XNOR ❷ si conectamos a la salida de una puerta XOR una puerta NOT.
- En este caso utilizaremos el CI 74HC266 ❸, que integra 4 puertas XNOR.
- Una puerta XOR está formada por ocho transistores ❹.
- Se conectan parejas en serie y entre parejas en paralelo.
- Las parejas de transistores de la parte inferior del esquema se conectan intercambiando el nivel de cada entrada de tal forma que si las entradas son diferentes una de las dos parejas de transistores se "activa" conectando Vout a 0V.
- Las parejas de transistores de la parte superior del esquema se conectan igualando el nivel de cada entrada de tal forma que si las entradas son iguales (o bien las dos 5V o las dos 0V) una de las dos parejas de transistores se "activa" conectando Vout a 5V.
- Con este circuito conseguimos el mismo funcionamiento que por programación ❺.

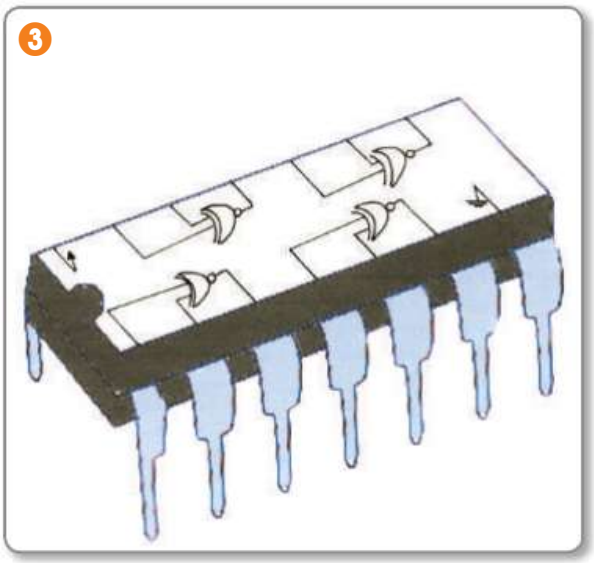
❶

```
void setup() {
  pinMode(13, OUTPUT);
  pinMode(12, INPUT);
  pinMode(11, INPUT);

}

void loop() {

  digitalWrite(13, !(digitalRead(12) ^ digitalRead(11)) );

}
```

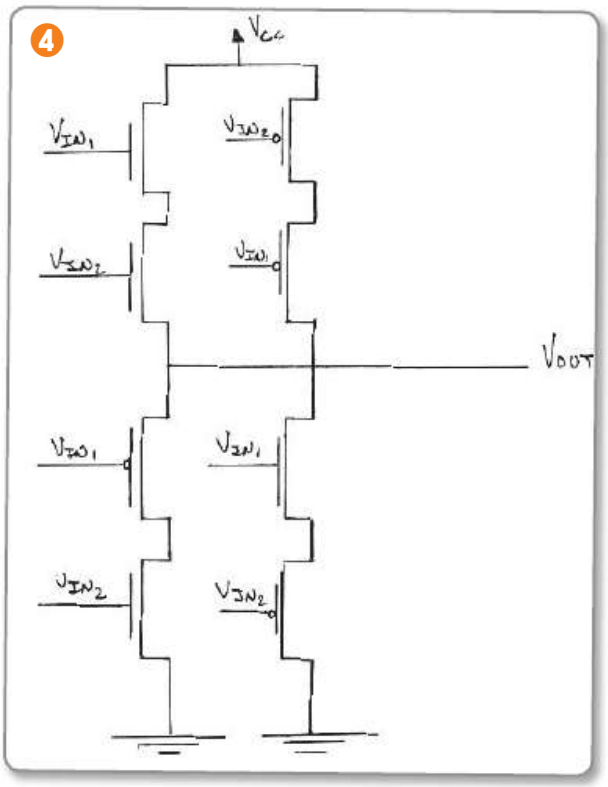
2


VIN1
VIN2
VOUT
VIN1
VIN2
VOUT

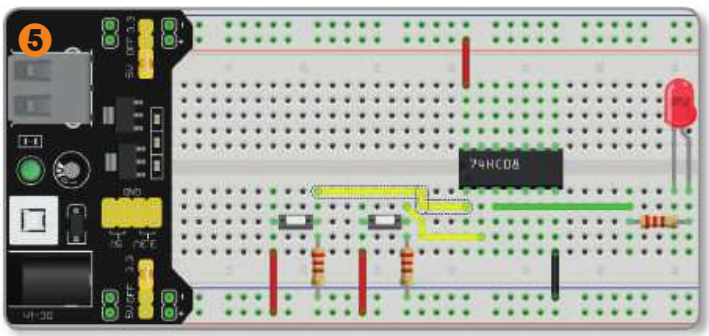
3



4


VCC
VIN1
VIN2
VIN2
VIN1
VOUT
VIN1
VIN1
VIN2
VIN2

5


74HC08

Variables numéricas y sistema binario

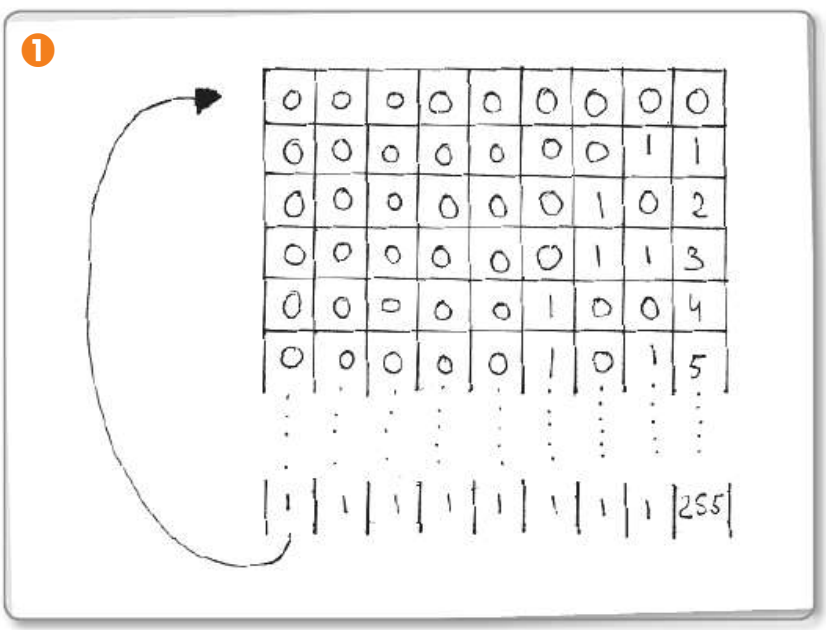
A medida que avancemos en los capítulos necesitaremos trabajar con variables de más de un bit, por ejemplo para almacenar resultados de cuentas o almacenar valores de sensores; es por eso que necesitamos saber cómo opera el sistema binario, las distintas unidades de este y el resto de tipo de variables que disponemos.

- El sistema binario es similar a nuestro sistema numérico, una de las diferencias es que en nuestro sistema número disponemos de hasta 10 dígitos (del 0 al 9) y en binario solo de dos dígitos (0 y 1).
- A la hora de crear valores cada vez más grandes necesitamos más dígitos; en el sistema binario sucede lo mismo, cuantos más dígitos (es decir, bits) podemos representar valores más grandes.
- Un término que puede resultar complicado es el desbordamiento, pero este concepto ya lo tenemos asimilado sin darnos de cuenta.
- El desbordamiento en nuestro sistema numérico equivale a que cuando tenemos, por ejemplo el número 9 y le sumamos una unidad, el dígito 9 desborda (vuelve a 0) y se le suma una unidad al siguiente dígito, con lo cual el resultado será 10.
- En el sistema binario ocurre exactamente lo mismo, lo que pasa es que se desborda antes, solo disponemos de dos digitos (0 y 1).
- Al igual que en el sistema número, cuanto incrementamos un valor en el sistema binario la suma se hace sobre el primer dígito (primer bit) y si hubiese uno o más desbordamientos se iría arrastrando el incremento de una unidad al bit siguiente.
- Por ejemplo si tenemos el número 999 y le sumamos una unidad existen 3 desbordamientos con lo cual el resultado será 1000.
- Si por ejemplo tenemos 4 bits y sus valores son 0111 y le sumamos una unidad, volvería haber 3 desbordamientos y el resultado sería 1000.
- Los números de nuestro sistema numérico se pueden codificar en números binarios. Gracias a esto podemos realizar operaciones numéricas con nuestros ordenadores.
- Una conclusión que podemos deducir es que cuantos más bits tengamos podremos llegar a representar un mayor valor numérico y es verdad, de hecho hay una relación directa que nos dice que en función de la cantidad de bits podemos calcular hasta que valor numérico podemos codificar.
- La fórmula es simple y viene de la expresión de 2^n, donde n es el número de bits.
- Por ejemplo operamos con 8 bits, podemos llegar a representar 2^8 =256, el 256 es el número de combinaciones posibles .
- Pero esto no quiere decir que lleguemos a representar el valor 256, realmente podemos representar hasta 2^n-1, por lo tanto podríamos codificar hasta el valor 255 [1] .
- Siempre debemos restar 1, puesto que valor de 0 también lo codificamos, por lo tanto, con 8 bits podemos codificar hasta 256 valores pero como uno de ellos es el 0 .Realmente codificamos del 0 al 255 (ambos incluidos).

A continuación se hará una lista con los tipos de datos numéricos junto con el rango de valores que codifican.

1. boolean: 1bit, codifica valores del 0 al 1
2. byte: 8bits, codifica valores del 0 al 255
3. int o short: 16bits, codifican valores del -32768 al 32767
4. word o unsigned int o unsigned short: 16bits, codifican valores del 0 al 65535
5. long: 4bit, codifica valores del -2147483648 al 2147483647
6. unsigned long: 32bits, codifica valores del 0 al 4294907295
7. float o double: 32 bytes, codifican valores del -3,4028235E+38 al 3,4028235E+38

Si en algún programa encontramos la variables declaradas como uint8_t, int16_t u otras por este estilo, no es más que una forma diferente de poner byte o int.



IMPORTANTE

Cuando trabajamos con una variable numérica podemos manipular sus valores en formato numérico o en formato binario, por ejemplo:

byte numero=12;

byte numero=B0001100;

En los dos casos anteriores estamos guardando el mismo valor en la variable número.

Para manipular valores binarios debemos utilizar el "prefijo" B.

En las variables nos encontramos algunas que permiten codificar valores negativos eso es porque utilizan uno de sus bits para indicar si es negativo o positivo el valor.

En caso de los valores decimáles es algo más complejo, por lo que no profundizaremos más.

Registros de los pines digitales

Antes de dar por finalizado el contenido sobre los pines digitales, revisaremos ciertos aspectos sobre cómo debe ser la señal de entrada por los pines para que no tengamos errores de lectura y cómo optimizar la programación.

- En cuanto al nivel de tensión de una señal, cuando trabajamos con una entrada digital sabemos que debe encontrarse entre 0 y 5V.
- ¿Qué ocurre cuando la señal es de por ejemplo 4 o 1,25V? Lo primero que tenemos que tener claro es que independientemente del valor de la señal, cuando tomemos una lectura siempre nos devolverá un valor (0 ó 1).
- La duda nos puede surgir cuando el valor no es exactamente 5 ó 0V.
- Quizás lo primero que pensemos es en dividir el rango a la mitad y de mitad para arriba la lectura sería un 1 (equivalente a 5V) y de mitad para abajo 0 (equivalente a 0V).
- Esto nos dejaría dos posibilidades: una señal de entrada entre 0-2,5V y la lectura sería de 0 y una entrada entre 2,5-5V y la lectura sería de 1.
- No iría muy desencaminado este planteamiento, en realidad, tenemos dos rangos fiables y un tercero de incertidumbre.
- Un rango entre 0V y el 30% de la tensión de trabajo del Arduino entre los que la lectura sería igual a 0 (es decir, entre 0-1,5V).
- Entre el 30% y el 60% de la tensión de trabajo del Arduino tenemos un margen de incertidumbre en donde la lectura puede ser aleatoria, por lo que debemos asegurarnos de encontrarnos fuera de esos valores (1,5-3V).
- Y por último tenemos el margen entre el 60% y el 100% donde una señal entre ese rango daría como lectura un 1.

Si revisamos el pineado del MCU (ATMEGA328P) del Arduino UNO, veremos que sus pines aparecen referenciados y al lado el pin correspondiente de la placa de Arduino.

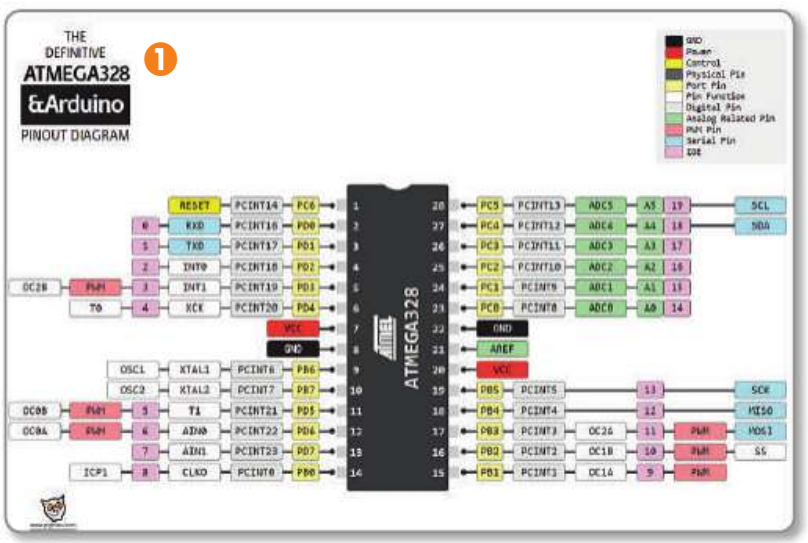
- Una cosa de la que nos damos cuenta es de que la disposición de los pines se redistribuyen en la placa (los han reordenado a través del diseño del hardware).
- Pero también puede llamar la atención que algunos de ellos se refencian con PB,PC o PD 1 .
- Esto es porque los pines se agrupan en lo que se conocen como puertos (B,C,D).
- En el puerto D se agrupan los pines del 0 al 7.
- En el puerto B se agrupan los pines del 8 al 13.
- En el porte C se agrupan los pines analógicos de la A0 a la A5.
- Para manipular los pines disponemos de tres registros: PORTx,PINx y DDRx. (en vez de la x pondríamos D, B o C según el puerto a manipular).
- El registro PORT permite establecer el valor del pin (5 ó 0V)
- El registro PIN devuelve el valor del pin
- El registro DDR permite configurar los pines como entradas o salidas.

- Los pines individuales de cada puerto se nombran PB0,PB1,PD0,PD7, etc.
- Los pines digitales se sitúan por orden según el puerto, el PD0 es el pin digital 1, el PB0 es el pin 8, etc.

Un caso en el que podemos aprovechar el uso de lo puertos sería por ejemplo si quisiéramos configurar todos los pines como salida, de momento (a falta de conocer los bucles) necesitaríamos una línea por cada pin (pinMode(,);) con lo que sería 14 líneas frente a dos si lo configuramos directamente el puerto ❷.

Los pines digitales (0-13) no se encuentran todos agrupados en un mismo puerto del 0-7 en el D y del 8-13 en el B es por eso que si revisamos el hardware veremos que hay una ranura entre los pines 7 y 8.

Para configurar un pin como INPUT_PULL debemos configurarlo como entrada en el DDR y activarlo en el PORT.



❶

```
void setup() {
  DDRD = B11111111;//0-7 salida
  DDRB = B111111;//8-13 salida
  PORTB = B100000;//Activar led 13
}
void loop() {}
```

❷

IMPORTANTE

Los pines analógicos se pueden comportar como digitales si los configuramos adecuadamente.

Por ejemplo, para trabajar con el pin A0 como pin digital utilizaríamos la instrucción :

pinMode(14,___);

Podemos configurar el A0 si lo referenciamos como 14 y a partir de ahí utlizarlos como pin digital (entrada o salida) y programarlo de la misma forma que los digitales.

A0-14, A1-15, A2-16, A3-17, A4-18, A5-19

Por el contrario, los pines digitales no se pueden configurar como analógicos.

El trabajar con los puertos nos permitirá optimizar nuestro programa, con lo que su ejecución sería más rápida y el programa ocuparía menos en la memoria del FLASH (si el programa se empieza hacer largo ,ir cojiendo estas costumbres nos puede salir muy rentable).

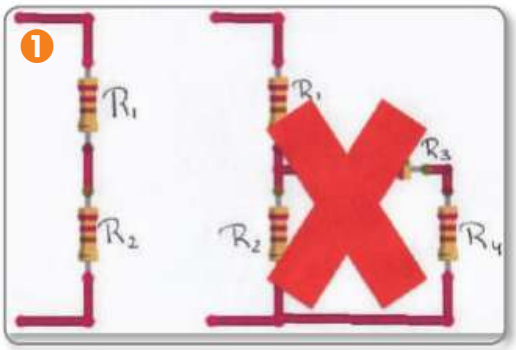
Resistencias en serie o en paralelo

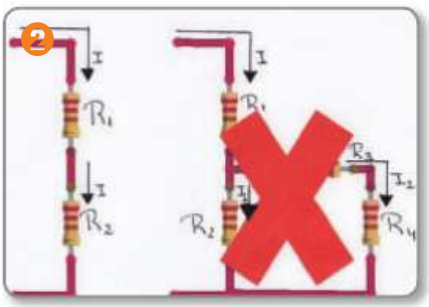
IMPORTANTE

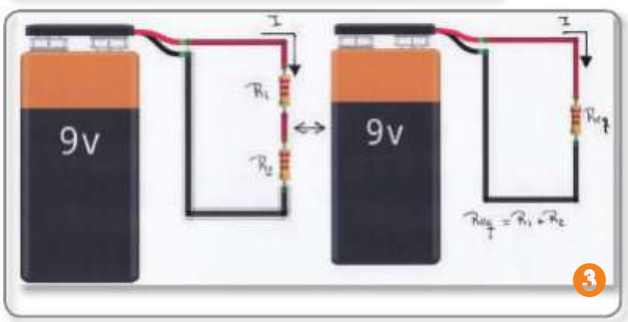
Puede ser que necesitemos el valor de una resistencia y que ese valor no sea habitual y nos cueste encontrarlo. Una solución sería combinar resistencias en serie y en paralelo para consiguir una Requ con el valor que estamos buscando.

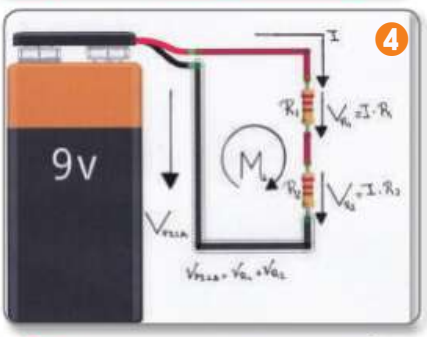
Revisaremos en este capítulo cómo podemos conectar resistencias (esto también se aplica a cualquier otro componente) entre sí. tenemos dos posibilidades: en serie o en paralelo.

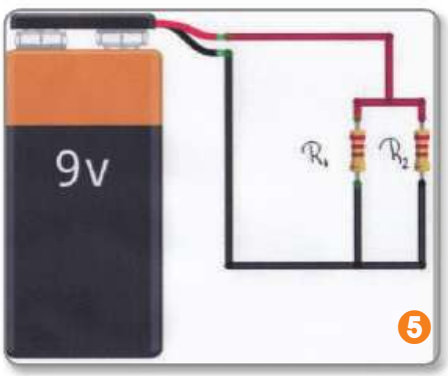
- Se dice que dos componentes (en nuestro caso resistencias) se encuentran conectados en serie cuando están conectados entre sí por un único terminal y sin existir un nodo entre ellos (de ser así necesitaríamos calcular la resistencia equivalente con el resto de componentes del nodo o nodos) ❶.
- Entre dos componentes que se encuentren conectados en serie circula la misma intensidad ❷.
- Podemos obtener la resistencia equivalente entre dos o más resistencias conectadas en serie ❸.
- La resistencia equivalente (Requ) sería la suma del valor de las resistencias conectadas en serie ❹.
- Con la Requ podemos calcular la intensidad que circula por ambas resistencias: I=V/Requ.
- Con el valor de esa intensidad volvemos al circuito inicial con las resistencias en serie y calculamos la caída de tensión en cada una.
- Dos componentes conectados en paralelo son aquellos que tienen sus dos terminales conectados directamente entre sí ❺.
- En el caso de las resistencias, la Requ saldría de dividir el resultado de multiplicar los valores de las resistencias entre su sumatorio: Requ=R1*R2/R1+R2 ❻.
- Si los componentes están conectados en paralelo, la caída de tensión en todos ellos es la misma.
- Para calcular la intensidad total hay dos formas:
- Calcular la resistencia equivalente y después, a través de la ley de Ohm, calcular la intensidad ❼.
- O, como sabemos la caída de tensión en cada resistencia, con la ley de Ohm calcular la intensidad que circula por cada una y luego sumarlas para obtener la intensidad total.

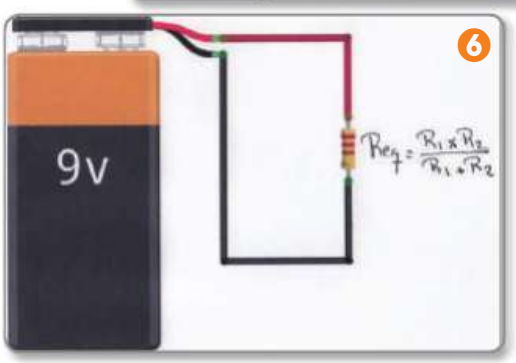

1
R1
R2
R1
R3
R2
R4

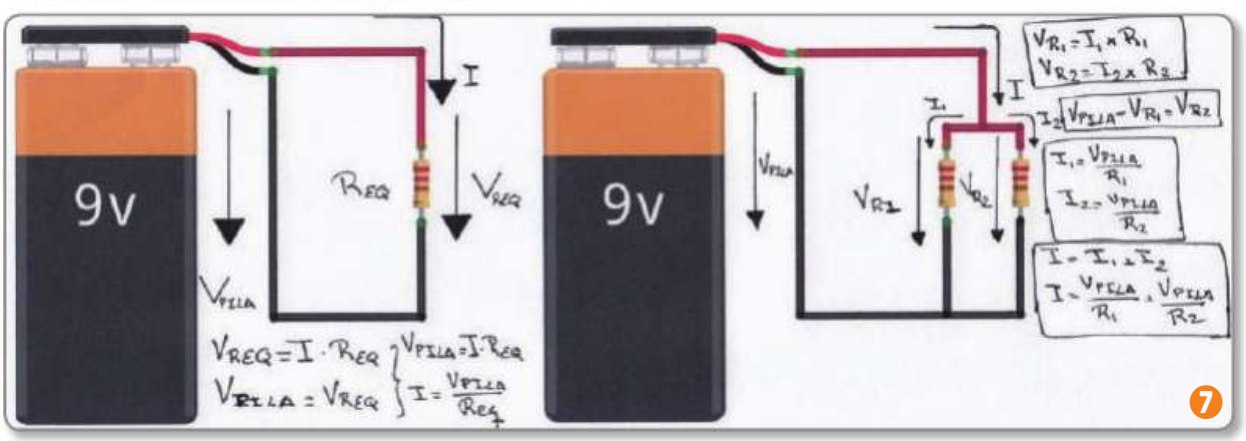

2
I
R1
I
R2
I
R1
R3
I2
R2
R4


I
9v
R1
R2
I
9v
Req
Req = R1 + R2
3


4
I
9v
R1
VR1 = I · R1
M
R2
VR2 = I · R2
VPILA
VPILA = VR1 + VR2


9v
R1
R2
5


6
9v
Req = R1 x R2 / R1 + R2


9v
I
REQ
VREQ
VPILA
VREQ = I · REQ
VPILA = VREQ
VPILA = I · REQ
I = VPILA / Req
9v
VPILA
I
I1
I2
VR1
VR2
VR1 = I1 x R1
VR2 = I2 x R2
VPILA = VR1 = VR2
I1 = VPILA / R1
I2 = VPILA / R2
I = I1 + I2
I = VPILA / R1 + VPILA / R2
7

Potenciómetro

IMPORTANTE

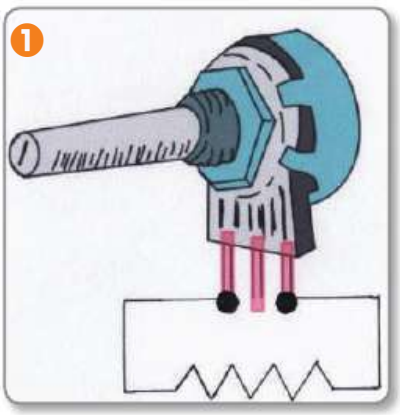
Un potenciómetro se puede considerar como un sensor que es capaz de cuantificar el movimiento de un mecanismo.

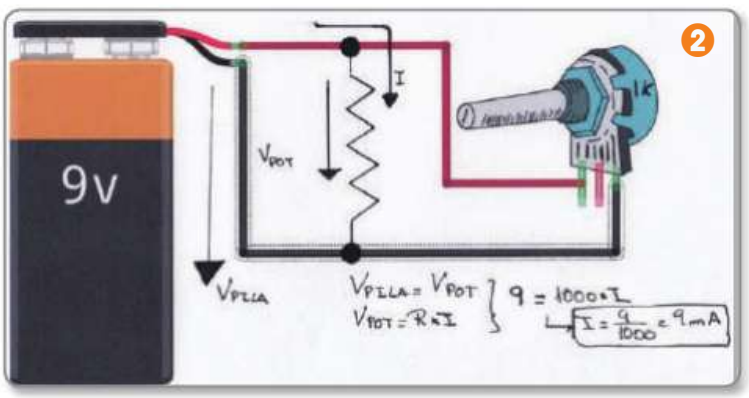
Existen diferentes tipos de potenciómetros y entre sus clasificaciones podemos distinguir por el tipo de señal que generan (proporcional, logarítmica...) o por el tipo de movimiento que detectan (rotatorio o deslizamiento).

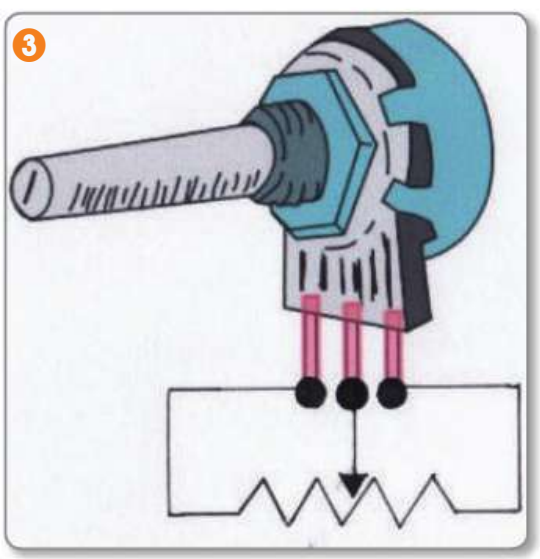
- Los potenciómetros se escogen del mismo modo que una resistencia: el valor de un potenciómetro se mide en ohmios.
- Un potenciómetro dispone de tres patillas: las de los extremos se utilizan para alimentar el potenciómetro y entre ellas se encuentra el valor de la resistencia del potenciómetro ❶ .
- La patilla central genera una señal proporcional a la posición del mecanismo (en nuestro caso de rotación).
- Si, por el momento, consideramos un potenciómetro como una resistencia, podemos estudiar con la ley de Ohm su consumo (consideramos que el potenciómetro es de 1k = 1000 ohmios). Si la pila es de 9V, la intensidad que circula es de 9mA ❷ .
- La patilla central ❸ se encuentra en contacto con la resistencia del potenciómetro; el punto de contacto cambia en función del movimiento de rotación del potenciómetro, es decir, es solidario al movimiento.
- En función del giro del potenciómetro, el contacto de esta patilla con la resistencia varía a lo largo de esta, siendo sus extremos la conexión directa; en este caso, con el polo positivo y negativo de la pila, respectivamente.
- Podemos ver esto como que esa patilla divide la resistencia en dos partes y que por lo tanto tenemos dos resistencias en serie.
- Lo que hace entonces esa patilla es variar el valor de las esas dos resistencias; si, por ejemplo, el mecanismo de rotación se encuentra en el centro, el valor de cada resistencia será la mitad del valor total del potenciómetro, con lo que cada resistencia tendría un valor de 500 ohmios y, como están en serie, la intensidad que circula por ambas sabemos que es la misma (9mA). Podemos así calcular la caída de tensión en cada una, que en este caso sería de 4,5V ❹ .
- Esta tercera patilla obtiene la caída de tensión de la resistencia inferior, por lo tanto en el caso anterior daría una tensión de 4,5V.
- Si giramos el potenciómetro hacia la patilla del positivo, por ejemplo hasta un 75%, el valor de la resistencia superior es 250, de la inferior es 750, y saliendo por la patilla

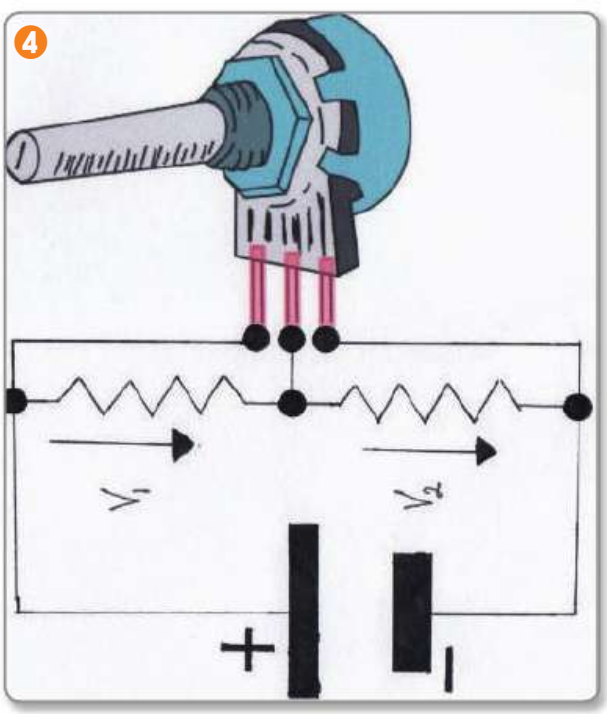
central 6,75V.

- Conclusión: a medida que giremos hacia el polo positivo, el valor de la tensión se irá acercando a 9V (en el extremo se conectaría directamente a 9V) y girando al negativo a 0V.









Señales y entradas analógicas

IMPORTANTE

A la hora de escoger un sensor con salida analógica, tenemos que tener en cuenta ciertas características: rango de medida, precisión y resolución.

Precisión: a mayor precisión menor margen de error habrá en la medida. Es un parámetro que nos dice cómo de fiable es el sensor. Posiblemente esto cambie en función de las condiciones de trabajo.

Resolución: indica cuál es la variación más pequeña que detecta el sensor. Por ejemplo, si la resolución es de 0,2º en un sensor de temperatura, significa que es capaz de generar una señal diferente para cada incremento de 0,2º.

Es importante tener clara la resolución con la que trabajan tanto el sensor como el Arduino, han de ser iguales o la del Arduino mayor para no perder información del sensor.

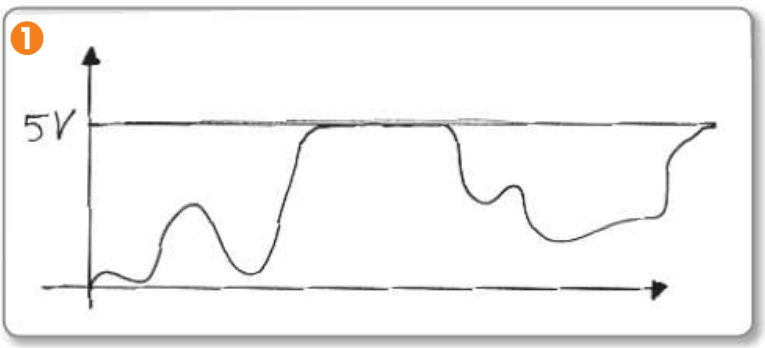
Por último recordar, que los pines analógicos se pueden comportar como digitales pero no a la inversa.

Las señales analógicas tienen más de dos estados: son señales que quizás entendamos mejor si las vemos como salidas de sensores ❶.

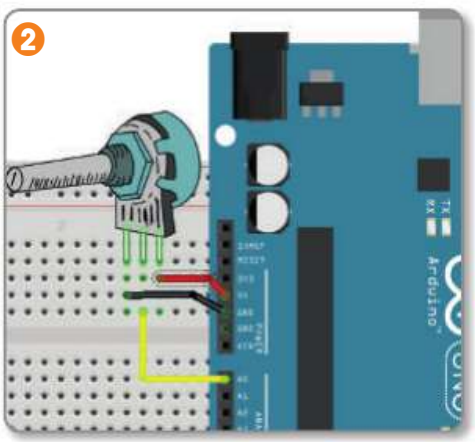
- Un sensor es un dispositivo electrónico capaz de transformar una magnitud física o química en un señal electrónica.
- Por ejemplo, un sensor de temperatura es capaz de transformar una magnitud física como es la temperatura en un señal eléctrica.
- Para no complicarnos, vamos a suponer que trabajamos con sensores cuya salida es proporcional a la entrada. Por ejemplo, tenemos un sensor de temperatura que mide entre un rango de 0 a 50º y su salida es proporcional entre 0 y 5V(no puede entrar una señal de más de 5V al Aduino).
- Nos tendremos que ir a los pines analógicos, que sí pueden cuantificar tensión de entre 0 a 5V.
- Disponemos de 6 pines digitales, referenciados de A0 a A5 (por programación podemos escoger esta referencia o nombrarlos del 0 al 5).
- Su resolución es de 10 bits, lo cual significa que nos devolverá un valor entre 0 y 1023 (2^10-1). Estos valores serán proporcionales a la tensión de entrada.
- Si la tensión como máximo puede ser de 5V, podemos detectar incrementos de unos 5mV (5/1023).
- Con todo esto podemos monitorizar el movimiento y la posición del mecanismo de un potenciómetro ❷, ya que este proporcionará una tensión en función de la posición de su eje.
- La instrucción que emplearemos será: analogRead(parámetro) ❸, que solo dispone de un parámetro y hace referencia a la entrada analógica que leemos. No es necesario configurarlo como entrada puesto que solo hay entradas analógicas y no salidas.
- Esta instrucción nos devolverá valores del 0 al 1023 en función de la señal que entre por el pin analógico ❹.

045

1



2



```
void setup() {
  Serial.begin(9600);
}

void loop() {
  Serial.println(analogRead(0));
  //También valdría analogRead(A0)
  delay(1000);

}
```

3

4

```
0
0
526
1023
527
175
527
1008
1023
1023
527
527
```

Autoscroll

Señales PWM

IMPORTANTE

Aunque una señal PWM en ciertos casos nos permita llegar a una solución igual que una señal analógica, debemos tener clara la gran diferencia: que en nigún momento generará un nivel de señal diferente a 0 ó 5V.

Por lo tanto si tenemos un sistema que se controla con una señal de 3,3V, una señal PWM no sería la solución ya que en los intervalos a nivel alto puede dañar ese equipo.

Los Timers son el mecanismo que utiliza el MCU para generar un señal PWM . Tenemos 3 Timers, el 0,1 y 2.

El timer0 de 980 Hz asociado a los pines 5 y 6

El timer1 de 490 Hz asociado a los pines 9 y 10.

El timer3 de 490 Hz asociado a los pines 3 y 11.

A pesar de existir unos pines propios para generar señales PWM, podemos mediante programación (con un nivel más avanzado) generar señales PWM por cualquiera de los pines.

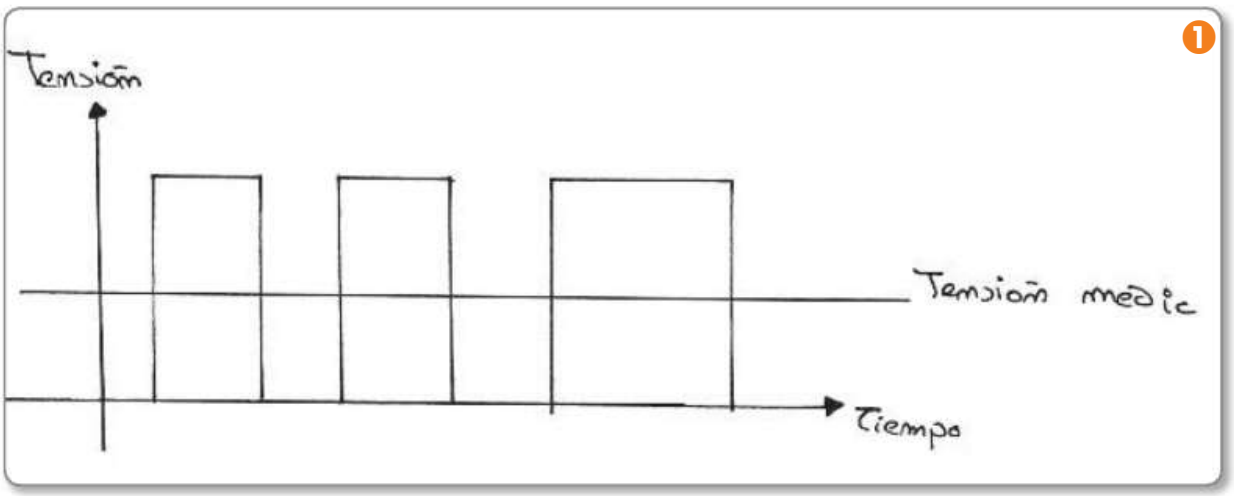
Siendo capaces de tomar la lectura de un potenciómetro, en este capítulo veremos cómo regular un led, en función de la lectura.

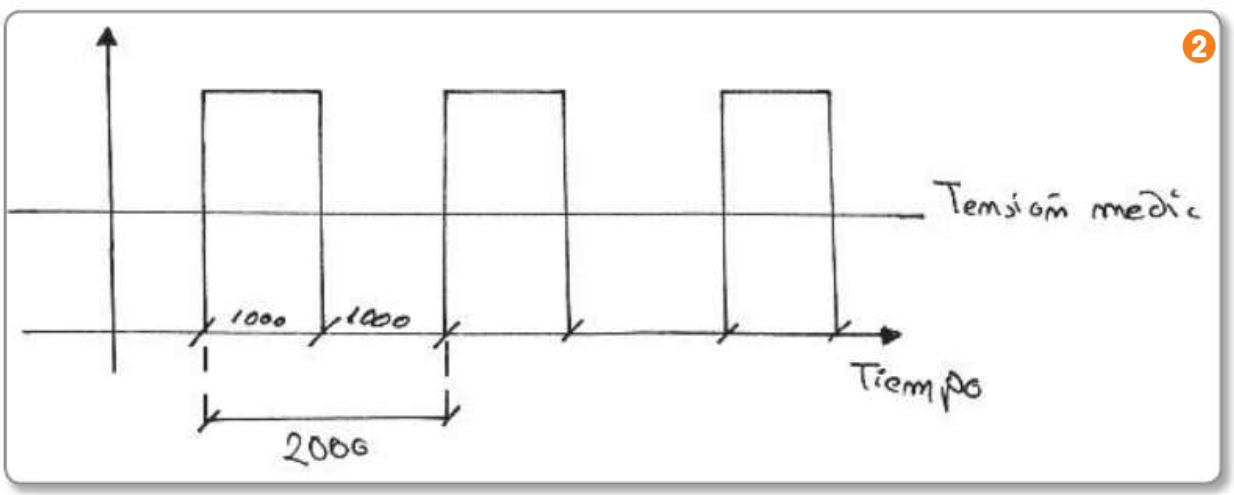
Si el Arduino UNO dispusiera de salidas analógicas sería muy sencillo. Si, por ejemplo, la entrada fuera de 3,5V, regularíamos una salida a 3,5 V; pero no disponemos de salidas analógicas, así que tendremos que solucionarlo con señales PWM.

- Una señal PWM ❶ es una señal que varía periódicamente entre sus dos posibles estados: encendido o apagado. Esos dos valores, en caso de un Arduino, son 0 o 5 V.
- Antes de empezar a generar señales PWM debemos ver previamente dos términos: «frecuencia» y «periodo».
- La frecuencia es el número de veces que se repite un evento en un segundo, y su unidad de medida es el hercio (Hz). Por ejemplo, si digo que, escribiendo con el teclado, mi frecuencia de escritura es de 5 Hz, esto quiere decir que escribo cinco palabras por segundo.
- Hay pines digitales que permiten la programación de señales PWM: 3, 5, 6, 9, 10 y 11. Los podemos identificar en el hardware por el símbolo ~.
- El periodo es el tiempo que dura una señal (se encuentre o no activada), y lo podemos conocer por la fórmula:

  T(período) = 1/f(frecuencia)

  T(segundos) = 1/hercios
- En consecuencia, cada señal que genera el Arduino (por los Timers de 490) por los pines de duran T = 1/490 = 0,002040816 segundos, redondeamos y consideramos que cada señal dura 2 milisegundos, o lo que es lo mismo, 2000 microsegundos.
- Vamos a suponer el siguiente caso: tomamos una lectura de la entrada analógica y esta nos devuelve el valor de 512, lo que significa que tenemos 2,5V. Para que el led se regule a una tensión similar a 2,5 voltios jugaremos con los tiempos de encendido y apagado de cada señal.
- Por lo tanto, teniendo en cuenta que una señal analógica tiene un valor de 0 o 5 V, si genero unas 500 señales por segundo y cada una de ellas dura 2000 microsegundos, si programo que cada señal se encuentre activa 1000 microsengundos y desactivada otros 1000 microsegundos ❷ la tensión media con la que trabajará el led es de 2,5V.


1
Tensión
Tensión media
Tiempo


2
Tensión media
1000
1000
Tiempo
2000

Generar señales PWM

IMPORTANTE

El resultado de una instrucción map es de tipo entero (sin decimales).

Antes de regular el led con una señal PWM, debemos analizar un último término con el que nos podemos encontrar: el ciclo de trabajo.

- Relaciona el tiempo que cada una de las señales de una PWM se encuentra a nivel alto sobre el tiempo total que dura la señal.
- En el caso de una PWM de frecuencia aprox 500, si cada señal dentro de su periodo dura 1000 microsegundos a nivel alto, el ciclo de trabajo es del 50% ([1000/2000]*100%).
- La resolución también afecta a la generación de señales PWM, en este caso tenemos una resolución de 8 bits; por el contrario, para la lectura tenemos 10.
- Lo que significa que tenemos registros de 8 bits (256 combinaciones del 0 al 255) en los cuales configuraremos el ciclo de trabajo.
- Un ciclo de trabajo del 100% equivale a un valor de 255, y los restantes a un valor proporcional entre 0 y 255.

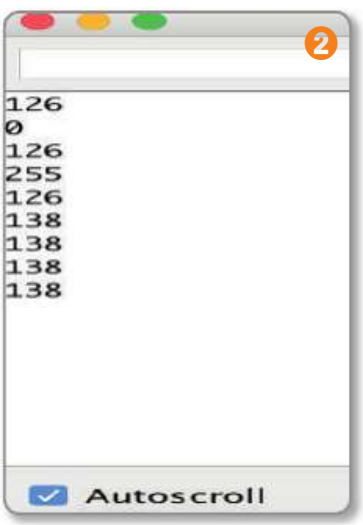
Necesitamos hacer operaciones ❶, por un lado tomaremos una lectura con un valor entre 0 y 1023 y nosotros debemos calcular el ciclo de trabajo con un rango entre 0 y 255 ❷.

- Tenemos una instrucción que nos facilita esta operación: map(,,,,);. Esta instrucción tiene cinco parámetros: el primero es el valor sobre el que queremos realizar la operación (en nuestro caso el valor de una entrada analógica); el segundo y tercer parámetro indican el rango de valores que puede alcanzar ese valor (en nuestro caso de 0 a 1023); y el cuarto y quinto parámetro indican entre qué rango de valores debe quedar el resultado (en nuestro caso 0 y 255).
- La instrucción analogWrite (,); permite generar señales PWM. Como primer parámetro indicamos sobre qué pin se genera la señal, y en el segundo parámetro introducimos el valor del ciclo de trabajo ❸.
- Con todo esto ya podemos regular un led con el potenciómetro ❹.

1

```
void setup() {
  Serial.begin(9600);

}

void loop() {
  int A0 = analogRead(0);
  byte pwm = map(A0, 0, 1023, 0, 255);
  Serial.println(pwm);
  delay(1000);

}
```

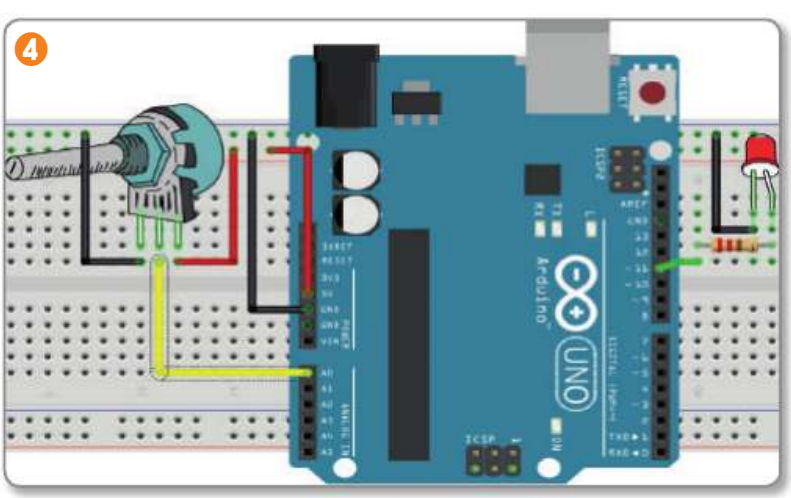
2



3

```
void setup() {
  pinMode(11, OUTPUT);
}

void loop() {
  int A0 = analogRead(0);
  byte pwm = map(A0, 0, 1023, 0, 255);
  analogWrite(11, pwm);
}
```

4



Estructuras de control

Hasta ahora nuestros programas estaban formados por un conjunto de líneas de código que se ejecutaban en su totalidad y de forma secuencial, es decir, desde la primera línea hasta la última.

- Según empecemos a crear proyectos más complejos necesitaremos tomar decisiones a lo largo del programa, y en función de ciertas condiciones necesitaremos que se ejecuten unas líneas de código u otras.

- Para ello programaremos estructuras de control, que nos permiten cambiar la ejecución del programa y, si las representáramos gráficamente, tendrían una ejecución ramificada.

- Existen diferentes tipos de estructuras de control de flujo [1], que podemos dividir en dos grupos: repetitivas y no repetitivas. Todos ellas son condicionales, es decir, permiten establecer una o varias condiciones que, al cumplirse, ejecutan las líneas de código de dicha estructura. La diferencia entre ambas radica en que las repetitivas ejecutan una y otra vez el código mientras se cumplan las condiciones, y las no repetitivas solo lo ejecutan una vez con cada evaluación de las condiciones establecidas (si se cumplen).

- Una buena práctica a la hora de programar es dibujar un boceto de cómo debe ser nuestro programa; a esto se le conoce como diagrama de flujo.

- Un diagrama sirve para organizar nuestras ideas y plantear el programa más óptimo. Si definimos un buen diagrama el siguiente paso será traducir ese diagrama en un programa.

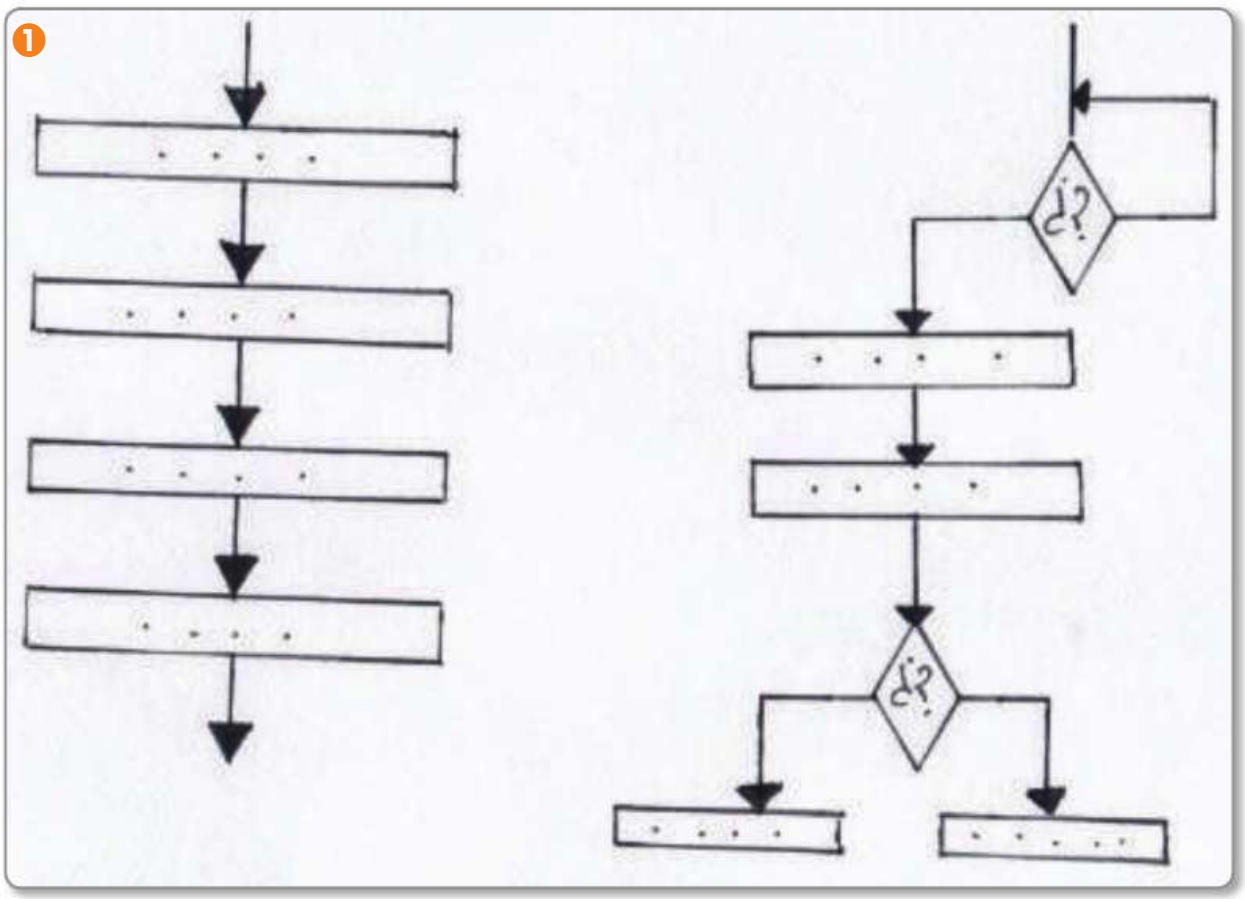
- Al igual que ocurría con el resto de recomendaciones (comentar programas, variables con nombres que se asocien a su uso), los diagramas de flujo son importantes para documentarnuestro trabajo.

- Si tratamos de modificar un programa antiguo y en su creación nos preocupamos de documentarlo y de hacerlo fácil de interpretar, no nos supondrá un gran esfuerzo volver a entenderlo y luego modificarlo.

- De hecho, para trabajos en grupo será imprescindible seguir esta serie de pautas.

- Y al final un diagrama de flujo tampoco consiste en un dibujo técnico difícil de realizar, está formado por rectángulos ,flechas, rombos ,elipses, etc.

048



Elementos básicos de un diagrama de flujo

En un diagrama de flujo los elementos básicos que siempre nos vamos a encontrar son: flechas, rectángulos y elipses.

- Las elipses se utilizan en un diagrama de flujo para indicar el inicio y fin del diagrama perteneciente a lo que vamos a programar dentro de: void loop, void setup, funciones ointerrupciones (estas dos últimas posibilidades las veremos en los capítulos correspondientes).

- Utilizaremos una elipse de inicio y fin para delimitar el contenido, y pondremos como título el nombre de la parte del programa que vamos a "dibujar" ❶ .

- En los rectángulos definiremos la operación u operaciones que vamos a realizar. No tenemos ningún leguaje especifico para cubrir cualquiera de estos campos; lo haremos de tal forma que nos resulte lo suficientemente explícito ❷ .

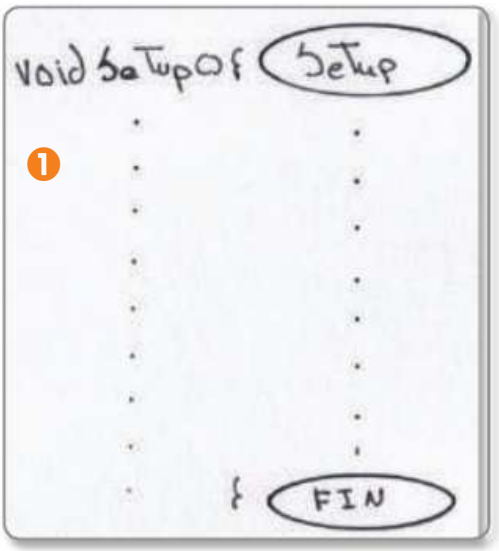
- En este caso podemos ver que tenemos varias opciones de dibujar una misma parte del programa, eso queda en función del criterio de cada uno. Al final también podemos ver un diagrama de flujo como un resumen o una ideal inicial sobre la que partir, y en la que posiblemente, una vez que empecemos a programar, tengamos que realizar cambios.

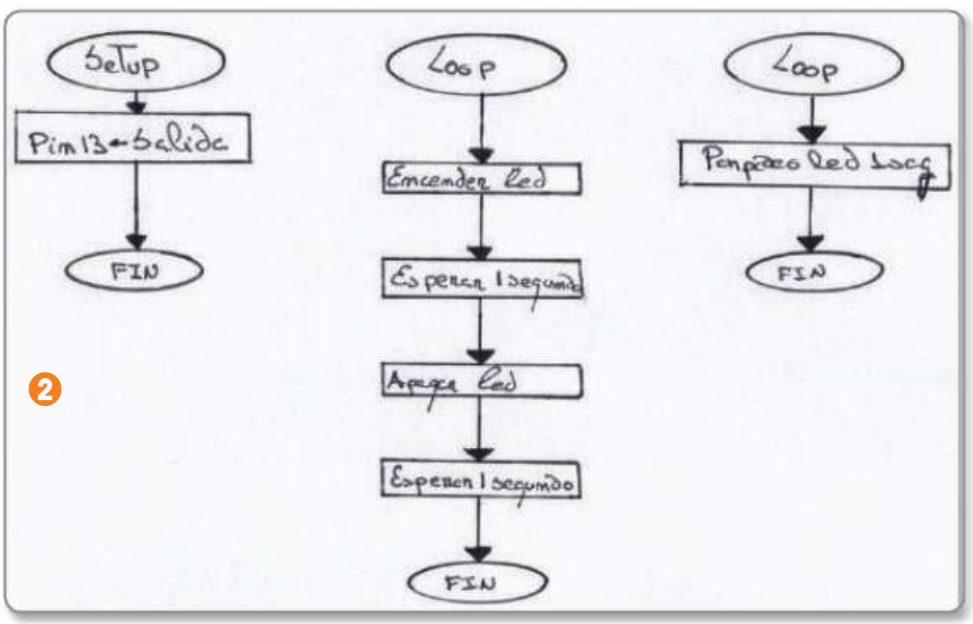
- Por último tenemos las flechas, que nos sirven para entender el camino que sigue el programa.

- Como por el momento no hemos definido ninguna estructura de control, quizás no veamos las flechas de gran utilidad, pero según se empiece a ramificar el programa nos serán de gran ayuda, y en ciertos casos imprescindibles.

- Obviamente para casos tan sencillos como el que hemos dibujado en este capítulo no tendría sentido dibujar un diagrama de flujo, pero en casos en los que ya nos suponga cierta dificultad serán una buena herramienta.

- Cada estructura de control tiene su propio "esquema", por eso según vayamos estudiando en cada capítulo una nueva estructura la acompañaremos de su esquema correspondiente.


void SetupO{
Setup
1
}
FIN


Setup
Pin13 → Salida
FIN
Loop
Encender led
Esperar 1 segundo
Apagar led
Esperar 1 segundo
FIN
Loop
Parpadeo led 1seg
FIN
2

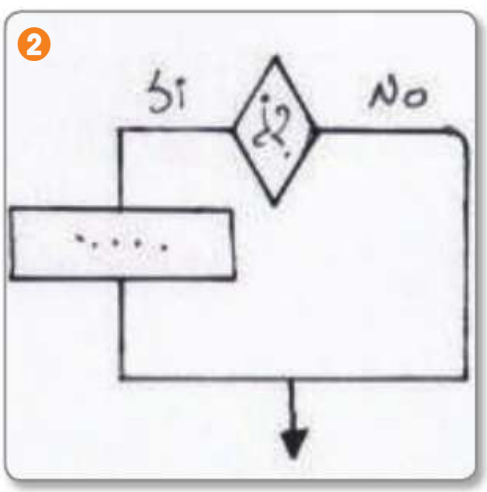
Estructura de Control if

Esta estructura nos va a permitir, cada vez que el Arduino la ejecute, que se evalúe una condición (o conjunto de ellas) y, si la condición se cumple, ejecute las instrucciones que engloba la estructura.

- Con los rombos podemos indicar cuándo se toma una decisión y, en función de si se cumple, que el programa siga un camino u otro.
- Esta estructura estará formada por la palabra reservada if ❶ , seguida entre paréntesis de la condición o condiciones y, por último, una llave de apertura y otra de cierre entre las cuales escribiremos las líneas de código que queremos que se ejecuten si se cumple la condición o condiciones.
- Del esquema de una estructura if ❷ podemos deducir también su funcionamiento. Se cumpla o no la condición, las líneas de código que se escriban a continuación de la llave de cierre se ejecutarán siempre.
- Primero se comprueba la condición y, si se cumple, se ejecutan las instrucciones que se encuentran dentro de las llaves (de forma secuencial) y se continúa con la ejecución del programa. Si no se cumple la condición, no se ejecutan las instrucciones que se encuentran entre las llaves y se «salta» a la instrucción que se encuentre después de la llave de cierre.
- ¿Qué es esto de una condición? ¿Qué quiere decir «evaluar condición»? Como ya hemos visto, en el sistema binario podemos hacer un símil; imaginemos que, al escribir una estructura if, estamos creando una celda en la memoria del MCU.
- «Evaluar» y «cumplir» una condición significa que, cada vez que se llegue a la línea de ejecución del if, accedemos a esa celda, en la cual sabemos que solo puede haber un 0 o un 1. Si hay un 1, significa que se cumple la condición, por lo tanto se ejecutan las instrucciones que se encuentran dentro de las llaves del if. Si por el contrario hay un 0, significa que no se cumple y saltamos a la línea en la que se encuentra la llave de cierre.
- Esto quizás se vea mejor con un ejemplo ❸ , en el que, si el botón está pulsado indicamos que se ejecute la instrucción de encender el led.
- Si se pulsa el botón se encenderá el led y si no, se apagará ❹.

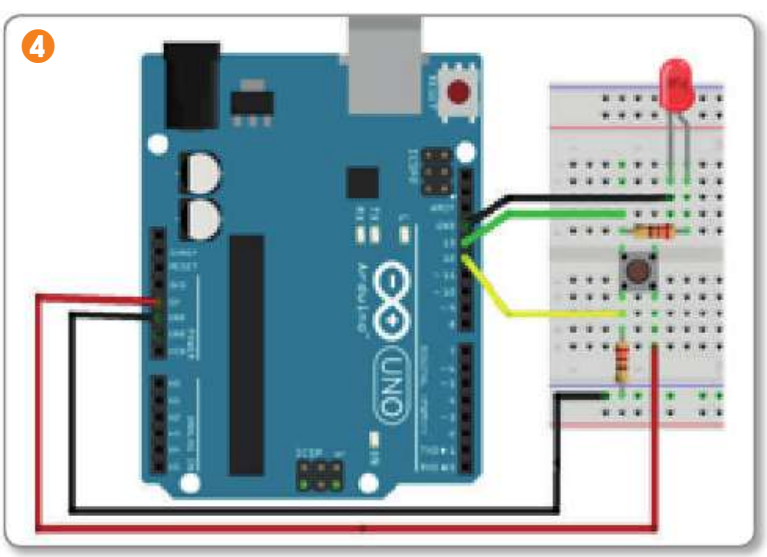
❶
```
if (condición) {

}
```

❷



❸
```
void setup() {
  pinMode(13, OUTPUT);
  pinMode(12, INPUT);

}

void loop() {

  if (digitalRead(12)) {
    digitalWrite(13, HIGH);
  }
  digitalWrite(13, LOW);

}
```

❹



if... else

En realidad, el programa del capítulo anterior no sería la mejor solución, puesto que, si probamos a añadir un delay de apenas 1 milisegundo al final del programa ❶, veremos cómo se enciende muy débilmente el led o ni eso.

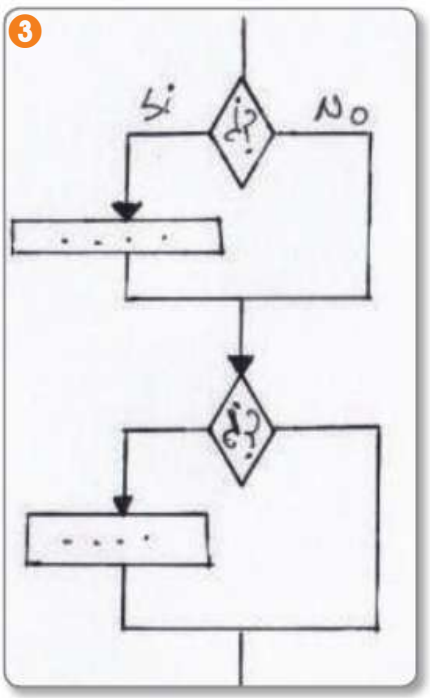
Esto es por que la instrucción de apagar el led se ejecuta siempre, independientemente de si se ha pulsado el botón o no.

- Una solución, con lo que ya sabemos, sería incluir dos estructuras if ❷ que comprueben si está o no pulsado el botón, y en función de eso ejecutar una de las dos instrucciones.
- Pero comprobar si está pulsado para luego comprobar si no lo está no parece una solución lógica ❸.
- En programación tenemos otra estructura ligada a la estructura if, esta sería la estructura else ❹.
- Esta estructura ejecuta las líneas de código que engloba, siempre y cuando no se cumpla la condición de la estructura if a la que se encuentra ligada.
- Ya no necesitamos por lo tanto hacer dos comprobaciones; definimos una condición, que en este caso sería que el botón se pulse y así se encienda el led. En caso de no cumplirse, se ejecutarían las líneas de código del else, que en este ejemplo apagarían el led ❺ .
- Una estructura else puede ejecutarse simplemente en caso de no cumplirse la condición de la estructura if definida justo en la línea anterior.
- Pero también se puede definir otra estructura if acompañando al else, por lo que empezamos a anidar estructuras. En el próximo capítulo veremos cómo las definimos, y ciertas condiciones a tener en cuenta
- En conclusión, una estructura if...else, permite evaluar una condición y programar dos operaciones diferentes en función de si se cumple la condición o no.

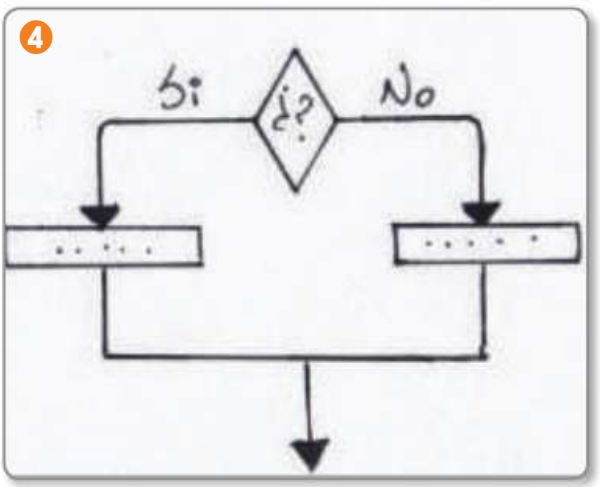
❶
```
void setup()
{
  pinMode(13, OUTPUT);
  pinMode(12, INPUT);
}

void loop()
{
  if (digitalRead(12))
  {
    digitalWrite(13, HIGH);
  }
  digitalWrite(13, LOW);
  delay(1);
}
```

❷
```
void setup()
{
  pinMode(13, OUTPUT);
  pinMode(12, INPUT);
}

void loop()
{
  if (digitalRead(12))
  {
    digitalWrite(13, HIGH);
  }
  if (!digitalRead(12))
  {
    digitalWrite(13, LOW);
  }
}
```

3



4



5

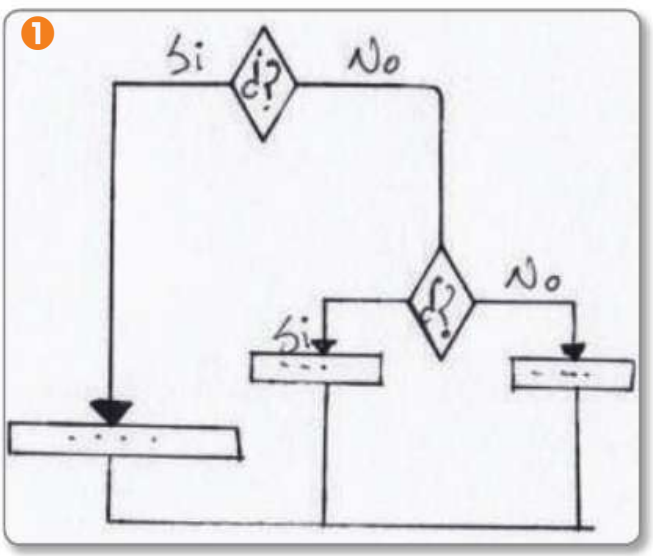
```
void setup()
{
  pinMode(13, OUTPUT);
  pinMode(12, INPUT);
}

void loop()
{
  if (digitalRead(12))
  {
    digitalWrite(13, HIGH);
  }else{
    digitalWrite(13, LOW);
  }
}
```

if... else anidados

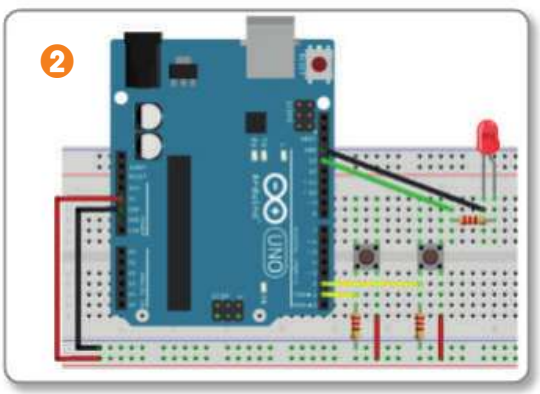
Hasta ahora podíamos evaluar una condición y realizar dos operaciones diferentes en caso de cumplirse o no. No obstante, puede darse la circunstancia de que, si no se cumple una condición, debamos evaluar una o más condiciones adicionales, puesto que existe una relación entre todas ellas.

- El programa cada vez puede seguir más caminos ❶. De esta forma, en caso de no cumplirse la primera condición podemos evaluar una segunda y, en función de esta, realizar dos operaciones diferentes.
- Podemos agrupar tantos if else if como necesitemos; lo que tenemos que tener claro es que, una vez que se cumpla una de las condiciones, el programa no evaluará las restantes.
- En el siguiente caso probaremos esta nueva estructura de control. Para ello, montamos un circuito nuevo❷ cuyo funcionamiento es el siguiente: si alguno de los dos botones está pulsado, el led debe encenderse y, de no ser así, debe apagarse.
- Se comprueba si uno de los dos botones está pulsado, de ser así se enciende el led y no se hacen más comprobaciones. Si ese primer botón no está pulsado se comprueba el segundo, si este está pulsado se enciende el led y en caso contrario se apaga el led puesto que ninguno de los dos se ha pulsado❸ .
- Como ya conocemos los operadores lógicos, podemos simplificar el programa anterior con una única estructura if...else en donde el if englobe dos condiciones ❹ .
- Si volvemos al caso de anidar if..else..if, en ciertos casos es importante el orden en el cual definimos condiciones.
- A esto se le conoce como prioridades, tendremos ciertas condiciones en las cuales prevalecen unas sobre otras, entonces unas son más prioritarias que otras.
- Por ejemplo❺ , en caso de que el botón conectado a la entrada 1 se pulse, el led siempre deberá permanecer encendido; si se pulsa el botón conectado a la entrada 2 el led deber parpadear cada segundo (siempre que no se pulse el botón 1); y por último, si ninguno de los dos se ha pulsado el led tiene que estar apagado.
- De este caso deducimos que es prioritaria una pulsación en el botón conectado a la entrada 1 frente a la entrada 2.
- Reflejamos esta prioridad consultando primero la condición de la entrada 1 y, de no cumplirse, consultamos la entrada 2.

1



2



3

```
void setup()
{
  pinMode(13, OUTPUT);
  pinMode(1, INPUT);
  pinMode(2, INPUT);
}

void loop()
{
  if (digitalRead(1))
  {
    digitalWrite(13, HIGH);
  } else if (digitalRead(2))
  {
    digitalWrite(13, HIGH);
  } else {
    digitalWrite(13, LOW);
  }
}
```

4

```
void setup()
{
  pinMode(13, OUTPUT);
  pinMode(1, INPUT);
  pinMode(2, INPUT);
}

void loop()
{
  if (digitalRead(1) or digitalRead(2) )
  {
    digitalWrite(13, HIGH);
  } else {
    digitalWrite(13, LOW);
  }
}
```

5

```
void setup()
{
  pinMode(13, OUTPUT);
  pinMode(1, INPUT);
  pinMode(2, INPUT);
}
void loop()
{

  if (digitalRead(1))
  {
    digitalWrite(13, HIGH);
  }  else if (digitalRead(2))
  {
    digitalWrite(13, HIGH);
    delay(1000);
    digitalWrite(13, LOW);
    delay(1000);
  } else {
    digitalWrite(13, LOW);
  }

}
```

Ampliación operadores

Los operadores de comparación permiten realizar operaciones con un único bit (0 o 1), lo que en ciertos casos no nos va a permitir evaluar correctamente una condición.

- == Igualdad: permite comparar dos valores: si son iguales, devuelve un 1 y, si son diferentes, un 0.
- != Diferencia: permite comparar dos valores: si son diferentes, devuelve un 1 y, si son iguales, un 0.
- > Mayor que: devuelve un 1 si el primer término es mayor que el segundo; de lo contrario, devuelve un 0.
- >= Mayor o igual que: devuelve un 1 si el primer término es mayor o igual que el segundo; de no ser así, devuelve un 0.
- < Menor que: devuelve un 1 si el primer término es menor que el segundo; de lo contrario, devuelve un 0.
- <= Menor o igual que: devuelve un 1 si el primer término es menor o igual que el segundo; de no ser así, devuelve un 0.

Utilizaremos por ejemplo el comparador >, este ejemplo consiste en: conectar un led y un potenciómetro ❶ . En caso de que el potenciómetro genere una señal menor a 2,5V el led se tiene que apagar y si por el contrario es mayor el led se encenderá.

Con una estructura if ...else será más que suficiente para solucionar este caso, la comprobación lo hacemos con 512 puesto que esto equivale a una señal de 2,5V ❷ .

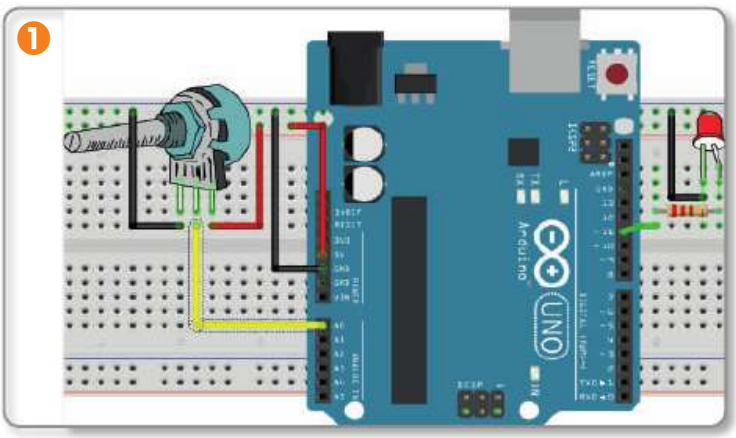
Tenemos otro tipo de operadores, los aritméticos:

- + Suma; - Resta; * Multiplicación; / División; % Módulo de una división

De los operadores aritméticos surgen los compuestos:

- x++; -> x = x + 1; x--; -> x = x - 1; x+=3; -> x = x + 3; x*=3; -> x = x*3; x/=3; -> x = /3;
- Con estos operadores podemos empezar a realizar operaciones básicas.
- Podemos incrementar el valor de una variable y monitorizar su valor por el monitor serie ❸ .
- Comprobamos que una variable tipo byte al llegar al valor de 255 se resetea ❹, puesto que todos sus bits desbordan y vuelven a cero, con lo que volvería a empezar el conteo.

1



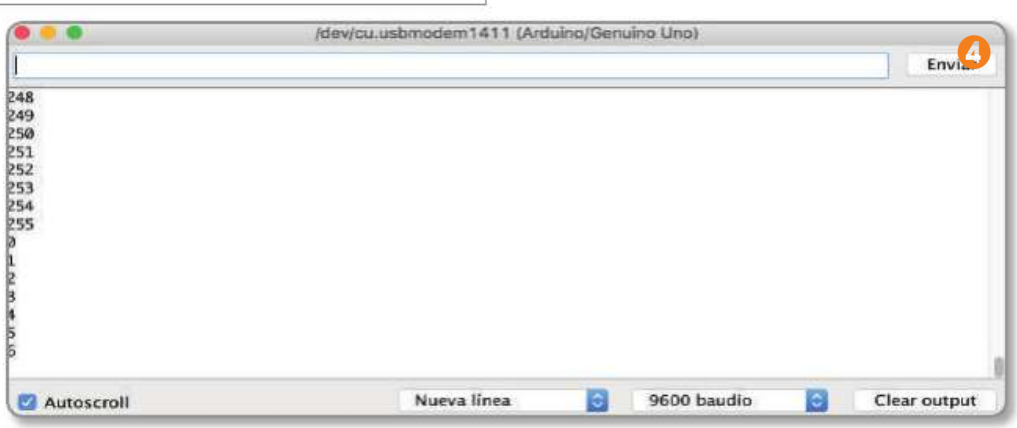
2

```
void setup()
{
  pinMode(11, OUTPUT);
  Serial.begin(9600);
}
void loop()
{
  Serial.println(analogRead(A0));
  if (analogRead(A0) > 512)
  {
    digitalWrite(11, HIGH);
  }
  else {
    digitalWrite(11, LOW);
  }

}
```

3

```
byte contador;
void setup()
{
  Serial.begin(9600);
}
void loop()
{
  Serial.println(contador++);
  delay(100);
}
```

4



switch

Una estructura switch equivale a dos o más estructuras if ❶ ❺ , en las cuales se realiza un operador de comparación (==).

Si tenemos que hacer muchas comparaciones, el programa podrá llegar a ser demasiado grande y difícil de interpretar. La estructura switch compara el valor de la expresión switch (expresión) con cada valor que acompañe a los case.

En el momento en que sea igual a uno de ellos, se ejecutarán las líneas de código que engloben el case y su break correspondientes ❷.

Al ejecutar las instrucciones, dejaría de realizar las comparaciones con el resto de case, saliendo de la estructura del switch. En caso de no ser igual a ninguno de los case, se ejecutarían las líneas de código escritas a continuación de default.

Vamos a realizar un ejemplo con esta nueva estructura, en el cual, si el potenciómetro genera una señal de 0 V, el led se apagará y no se encenderá hasta generar una salida de 5 V, y viceversa ❸.

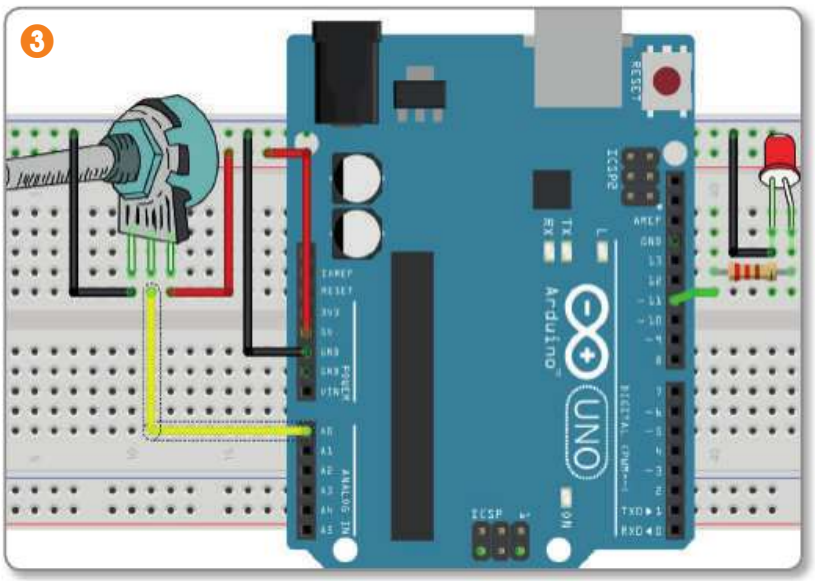
Para conseguir este funcionamiento, necesitamos dos comparaciones ❹. Una condición para la cual la señal es de 0V y otra para cuando son 5V. En cada una de ellas actuaremos sobre el led encendiéndolo o apagándolo. Y en la parte de default no se hace nada dejando el estado del led sin modificar.

Conclusión: la estructura switch es la idónea para cuando necesitemos tomar decisiones en función de un valor numérico, ya venga de una operación o lectura.

❶
```
if(variable==1){
  //acción
}else if(variable==2){
  //acción
}else if(variable==3){
  //acción
}else if(variable==4){
  //acción
}else
```

❷
```
switch (expresión) {
    //La expresión puede ser una variable o una función.
  case valor1:
    /*Si la expresión es igual a este valor, ejecuta las
      instrucciones hasta que encuentre*/
    break; /*no se evalúan más valores,
    se sale del bucle.*/
  case valor2:
    /*Si la expresión es igual a este valor,
      ejecuta las instrucciones hasta que encuentre*/
    break; //no se evalúan más valores se sale del bucle.
    default;
    /*si la expresión no es igual a ningún valor,
      se ejecutan estas instrucciones y se sale del bucle.*/
}
```

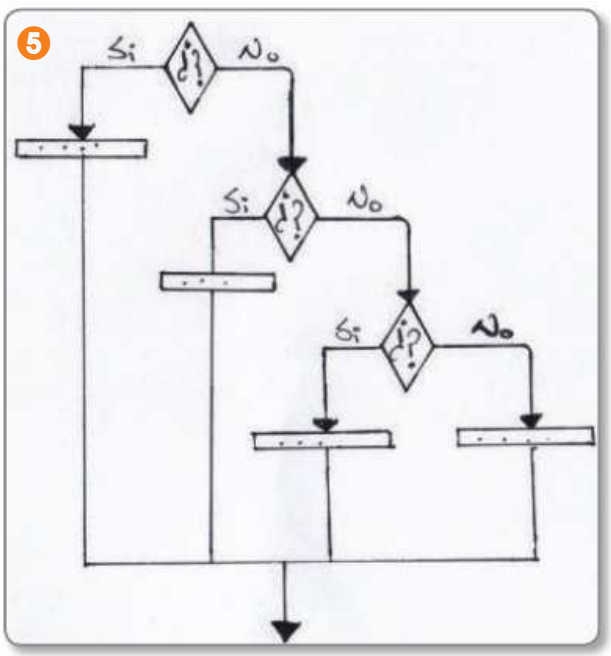
3



4

```
void setup()
{
  pinMode(11, OUTPUT);
}

void loop() {

  switch (analogRead(A0)) {
    case 0:
      //En caso de que la lectura sea igual a 0
      //se apagará el led
      digitalWrite(11, LOW);
      break ;
    case 1023:
      //En caso de que la lectura sea igual a 1023
      //se encenderá el led
      digitalWrite(11, HIGH);
      break ;
    default:
      //No hacemos nada en caso de que la lectura
      //sea diferente de 0 y 1023
      break ;
  }
}
```

5



while

Esta estructura ejecuta las instrucciones que se encuentran entre sus llaves de apertura y cierre. ¿Cuántas veces? Se ejecutarán mientras la condición especificada se cumpla, sin ejecutarse ninguna otra línea de código que se encuentre fuera de la estructura ❶.

Una estructura while es similar a una if, la diferencia entre ambas es que una estructura if comprueba unas condiciones que, de cumplirse, ejecutan las líneas de código que engloba y, una vez finalizada su ejecución, el programa sale de la estructura y continúa con la ejecución de otras líneas de código si las hubiese; mientras que una estructura while no sale de la ejecución de las líneas de código que engloba hasta que dejen de cumplirse las condiciones.

1. Montamos un nuevo circuito ❷.
2. Cargamos el siguiente programa ❸ para comprobar el funcionamiento de esta nueva estructura.

En este caso, mientras el botón se encuentre pulsado el led estará encendido. Ya que se estaría ejecutando una y otra vez la instrucción que activa la salida, en el momento que no se cumpla la condición (botón no pulsado) el programa sale de la estructura y apaga durante al menos 2 segundos el led.

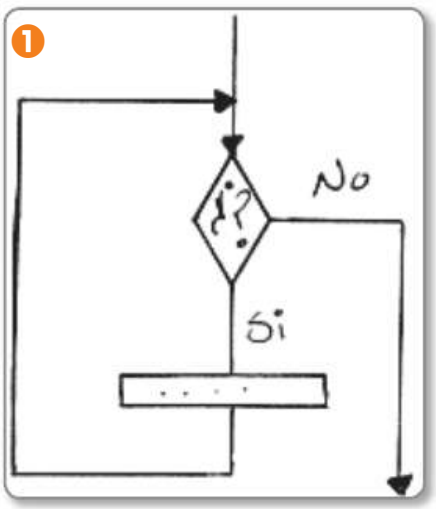
La programación de condiciones de una estructura while es exactamente igual que un if; podemos definir diferentes condiciones ayudándonos de los operadores vistos en capítulos anteriores.

A la hora de utilizar un while debemos tener precaución puesto que se puede bloquear el programa.

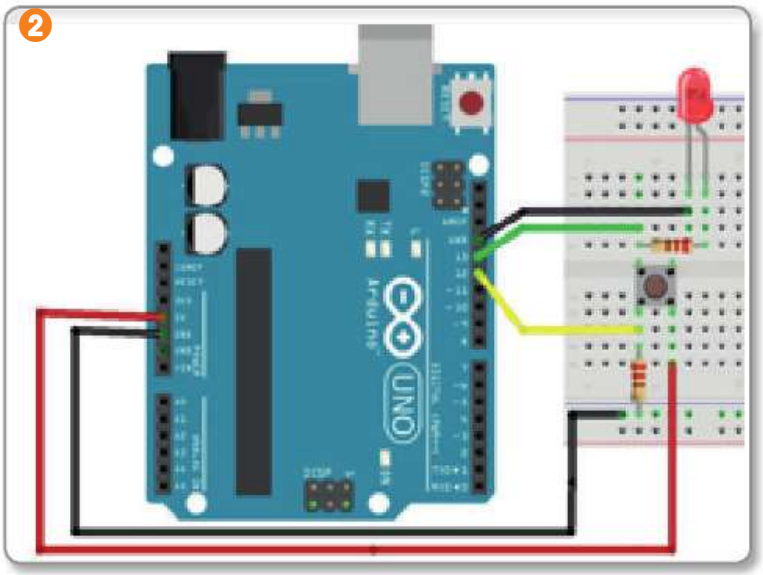
Cargando este último programa ❹ y manteniendo el circuito anterior, comprobamos que el led nunca se llegará a encender ya que el programa se queda bloqueado en la linea while(1); y no se llegaría a ejecutar en ningún caso la línea del void loop.

Es por eso que debemos tener cuidado a la hora de programar un while, parándonos a analizar como se definen las condiciones.

1



2



3

```
void setup()
{
  pinMode(13, OUTPUT);
  digitalWrite(13, LOW);
  pinMode(12, INPUT);
}

void loop()
{
  while (digitalRead(12)) {
    digitalWrite(13, HIGH);
  }
  digitalWrite(13, LOW);
  delay(2000);
}
```

4

```
void setup()
{
  pinMode(13, OUTPUT);
  digitalWrite(13, LOW);
  while (1);

}
void loop()
{
  digitalWrite(13, HIGH);
}
```

do while

La estructura do while funciona de una forma similar a una while. La diferencia es que las líneas de código de esta nueva estructura se ejecutan al menos una vez ❶; después de esto, se evalúa la condición y, si se cumple, se vuelven a ejecutar mientras se siga cumpliendo.

1. Conservar el montaje del capítulo anterior.
2. Cargar un nuevo programa ❷.

La condición de la estructura do while es que se pulse el botón conectado al pin 12, de no ser así el led estaría encendiéndose un segundo y apagándose cinco continuamente.

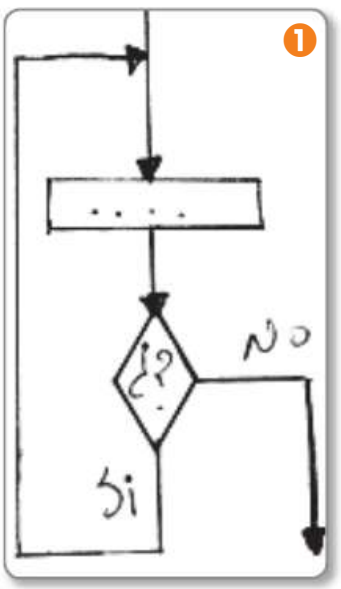
Puesto que cuando el programa llega a la línea del do no evalúa ninguna condición y ejecuta las líneas de código que hay dentro de la estructura, y luego comprueba la condición, al no pulsarse en este supuesto el botón el programa continúa con su ejecución apagando el led y deteniéndose durante 5 segundos.

En el supuesto de que el botón se encuentre pulsado, el programa al llegar a la línea del do al igual que antes vuelve a ejecutar las líneas de código que hay dentro de la estructura. Una vez ejecutadas evalúa la condición; como esta vez sí se cumple, el programa "salta" a la línea del do para volver a ejecutar una vez más las líneas de código que hay dentro de la estructura.

Este proceso se repetirá todo el tiempo mientras el botón se encuentre pulsado, por lo tanto el led estaría continuamente encendido.

Al igual que ocurría con la estructura while, debemos tener precaución para no bloquear la ejecución del programa ❸ debido a una estructura do while, tal y como sucede en el siguiente programa.

Como la condición siempre se cumple, el led estaría continuamente encendido, de esta forma podemos comprobar que una estructura do while mal programada puede bloquear un programa.



2

```
void setup()
{
  pinMode(13, OUTPUT);
  digitalWrite(13, LOW);
  pinMode(12, INPUT);

}
void loop()
{
  do {
    digitalWrite(13, HIGH);
    delay(1000);
  } while (digitalRead(12));
  digitalWrite(13, LOW);
  delay(5000);
}
```

3

```
void setup()
{
  pinMode(13, OUTPUT);
  digitalWrite(13, LOW);
  pinMode(12, INPUT);

}
void loop()
{
  do {
    digitalWrite(13, HIGH);
    delay(1000);
  } while (1);
  digitalWrite(13, LOW);
  delay(5000);
}
```

for

Esta estructura permite fijar el número de veces que se repite la ejecución de las líneas de código que engloba ❶.

La condición de esta estructura viene definida por tres parámetros:

- Valor inicial: puede empezar por cualquier valor numérico.
- Valor final contador: cada vez que se ejecuten las instrucciones de la estructura, se incrementa el valor del contador conforme se defina en el campo de operación, y se comprueba si se alcanza el valor final; de ser así, se da por finalizado el bucle y se continúa con la ejecución del resto del programa.
- Operación: define qué operación se realiza sobre el valor del contador del bucle (suma, resta, multiplicación, división...).

Se puede hacer una comparación entre una estructura for y una while; la estructura for viene siendo una estructura while en la cual la condición es un número de repeticiones ❷.

1. Realizar una nueva conexión ❸.
2. Cargar un nuevo programa ❹.

Lo que se consigue con este nuevo programa es que el led parpadea cada segundo 10 veces, esto mismo lo podemos conseguir con una estructura while ❺.

Cuando se trata de realizar una serie de operaciones un número determinado de veces, por lo general se suele emplear la estructura for frente a la while.

Independientemente de cuál de las dos estructuras se utilice, lo que es evidente es que permiten optimizar el código de nuestro programa y hacerlo más comprensible.

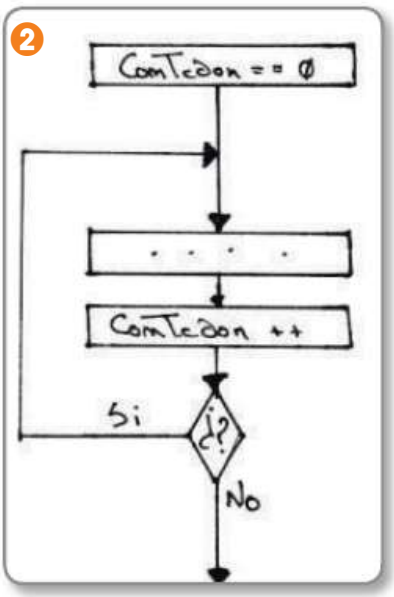
- Podemos configurar todos los pines como salida en un par de líneas ❻.
- O por ejemplo muestrear todas las entradas analógicas ❼.

❶

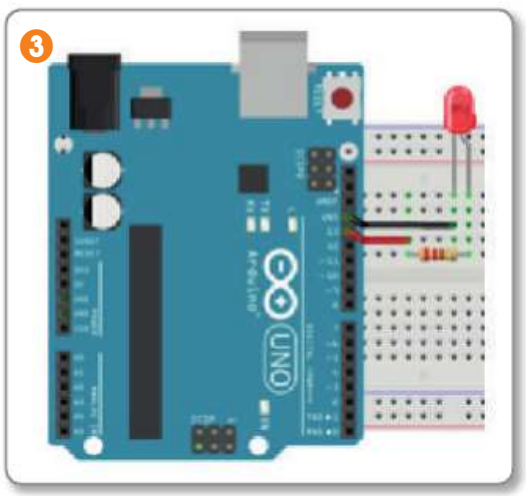
```
for (valor_inicial_contador ; valor_final_contador ; operación)
{
  //Instrucciones que se repetirán un número determinado de veces
}
```

057

2



3



4

```
void setup()
{
  pinMode(13, OUTPUT);
  for (byte contador = 0; contador < 10; contador++)
  {
    digitalWrite(13, HIGH);
    delay(1000);
    digitalWrite(13, LOW);
    delay(1000);
  }

}
void loop()
{

}
```

5

```
void setup()
{
  pinMode(13, OUTPUT);
  byte contador = 0;
  while ( contador < 10)
  {
    digitalWrite(13, HIGH);
    delay(1000);
    digitalWrite(13, LOW);
    delay(1000);
    contador++;
  }
}
void loop(){}
```

6

```
void setup()
{
  for (byte i = 0; i < 10; i++)
  {
    pinMode(i, OUTPUT);
  }
}
void loop() {}
```

7

```
void setup()
{
  Serial.begin(9600);
  for (byte i = 0; i < 6; i++)
  {
    Serial.println(analogRead(i));
  }
}
void loop() {}
```

Directivas

Al realizar un programa, este es "traducido" a un lenguaje que puede interpretar el MCU de un Arduino. Este proceso de traducción se llama compilación, y el software encargado de realizar la traducción se denomina compilador (IDE de Arduino).

Las directivas no son más que unas instrucciones que el programador necesita o quiere que el compilador ejecute antes de realizar la traducción y posterior carga del programa.

- La primera directiva que trataremos será << #define >> ❶; esta directiva permite asignar un nombre a un dato para que así esta no ocupe espacio de memoria.
- Esto nos permite definir operaciones ❷.
- Por lo general, este tipo de variables se escriben en mayúsculas para diferenciarlas del resto.
- Existen otras directivas que permiten "activar" código de nuestro programa en función de una condición. En función del valor de la variable ACTIVAR ❸ se incluyen en el programa que se vaya a cargar al Arduino las líneas de código que se encuentran entre el if y el endif.
- Las directivas condicionales nos permiten tener escritas diferentes líneas de código ❹ en nuestro programa y de forma rápida realizar una gran cantidad de cambios.
- Existen otras directivas condicionales cuya condición consiste en si existe una determinada variable.
- En este caso ❺ lo importante no es el valor de la variable ACTIVAR, sino si existe; por eso a pesar de que su valor es 0 el led se enciende.
- Si comentamos la declaración de la variable ❻, el led se apagará
- Existen más directivas con las que, llegado el momento, trabajaremos; pero con lo que ya hemos visto podemos considerar que las directivas son una herramienta que tiene el programador.

```
#define NUMERO_PI 3.14                    ❶
void setup()
{
  Serial.begin(9600);
  Serial.println(NUMERO_PI);
}
void loop() {}
```

```
#define NUMERO_PI (3.14*3)+5
void setup()
{
  Serial.begin(9600);
  Serial.println(NUMERO_PI);
}
void loop() {}
```

2

```
#define ACTIVAR 1
void setup()
{
#if ACTIVAR
  pinMode(13, OUTPUT);
  digitalWrite(13, LOW);
#endif
}
void loop() {}
```

3

```
#define ACTIVAR 0
void setup()
{
  pinMode(13, OUTPUT);
#if ACTIVAR
  digitalWrite(13, HIGH);
#else
  digitalWrite(13, LOW);
#endif
}
void loop() {}
```

4

```
#define ACTIVAR 0

void setup()
{
  pinMode(13, OUTPUT);
#ifdef ACTIVAR
  digitalWrite(13, HIGH);
#else
  digitalWrite(13, LOW);
#endif
}
void loop() {}
```

5

```
//#define ACTIVAR 0
void setup()
{
  pinMode(13, OUTPUT);
#ifdef ACTIVAR
  digitalWrite(13, HIGH);
#else
  digitalWrite(13, LOW);
#endif
}
void loop() {}
```

6

break continue

Son instrucciones que permiten modificar la ejecución de una estructura.

Con la instrucción break, se da por finalizada la ejecución de una estructura de control.

- Este programa hace que el led parpadee cuatro veces ❶, ya que al llegar el contador a 4 se cumple la condición que hay definida en la estructura if y se ejecuta la instrucción break .
- Al ejecutarse el break, el programa sale de la estructura contenedora principal, que en este caso es el bucle for, por lo que no habrá ningún parpadeo más.

La instrucción continue fuerza a volver a la primera instrucción de la estructura, sin evaluar las condiciones ni ejecutar las líneas de código restantes.

- En ese caso ❷ en el momento en que el contador del bucle llegue al valor de 4 se cumple la condición de la estructura if, con lo que se ejecuta la instrucción continue.
- Esta instrucción lo que hace es que el programa salte a la última línea del bucle sin ejecutarla, con lo que no se realizan las líneas de código del parpadeo cuanto el contador vale 4.
- En vez de parpadear 10 veces, el led parpadeará 9 veces (cuando el contador vale 4 no parpadea).
- Con la instrucción continue podemos hacer un caso similar al de la instrucción continue ❸.
- Si en este caso la condición de la estructura if es que el contador se encuentre entre los valores 4 y 10, veremos que, al igual que ocurría con el programa de la instrucción break, el led parpadeará 4 veces.
- Lo que sucede es que entre los valores 4 y 10 se ejecuta la instrucción continue, con lo que el programa vuelve a la primera línea de la estructura sin ejecutar el resto de instrucciones.

1

```
void setup()
{
  pinMode(13, OUTPUT);
  for (byte i = 0; i < 10; i++) {
    if (i == 4)
    {
      break;
    }
    digitalWrite(13, HIGH);
    delay(1000);
    digitalWrite(13, LOW);
    delay(1000);
  }

}
void loop() {}
```

2

```
void setup()
{
  pinMode(13, OUTPUT);
  for (byte i = 0; i < 10; i++) {
    if (i == 4)
    {
      continue;
    }
    digitalWrite(13, HIGH);
    delay(1000);
    digitalWrite(13, LOW);
    delay(1000);
  }

}
void loop() {}
```

3

```
void setup()
{
  pinMode(13, OUTPUT);
  for (byte i = 0; i < 10; i++) {
    if (i >= 4 and i < 10)
    {
      continue;
    }
    digitalWrite(13, HIGH);
    delay(1000);
    digitalWrite(13, LOW);
    delay(1000);
  }

}
void loop() {}
```

goto

Esta instrucción permite "saltar" del flujo habitual del programa. Permite ejecutar unas líneas de código que vienen precedidas por un "título".

Es algo así como poner una referencia a una línea del programa para poder "saltar" a ella cuando queramos.

Para trabajar con saltos de línea debemos tener en cuenta ciertas normas:

- La instrucción empleada para realizar un salto de línea es goto que irá acompañada del nombre que le hemos asignado a una línea en concreto del programa; esta llamada debe finalizar en punto y coma.
- No puede haber más de una línea con el mismo nombre, puesto que el programa no sabría a cuál de las líneas debe "saltar" (el IDE ya no nos permite definir más de una línea con el mismo nombre).
- Los nombres de los saltos de línea se rigen por las mismas normas que los de las variables.
- Para poner el nombre a una línea esta debe acabar en dos puntos y punto y coma.

Con estas reglas ya podemos hacer una prueba con los saltos de líneas, en este caso ❶ el led se encontrará encendido en todo momento.

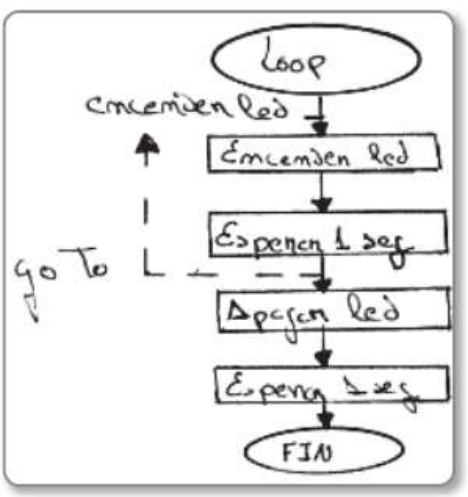
Ya que el goto hace que cada vez que el programa llega a su altura salta a la línea que tiene el nombre encender_led, con lo que estaría continuamente encendiendo el led.

Los saltos de línea se pueden realizar hacia "delante" o hacia "atrás", en este caso ❷ el led volvería a estar encendido continuamente, ya que nos estamos saltando otra vez las líneas que apagarían el led durante 1 segundo.

Los saltos de línea son sencillos de usar, pero no son un recurso muy usado en programación.

1
```
void setup()
{
  pinMode(13, OUTPUT);
}
void loop()
{
encender_led:;
  digitalWrite(13, HIGH);
  delay(1000);
  goto encender_led;
  digitalWrite(13, LOW);
  delay(1000);
}
```

2
```
void setup()
{
  pinMode(13, OUTPUT);
}
void loop()
{
  digitalWrite(13, HIGH);
  delay(1000);
  goto encender_led;
  digitalWrite(13, LOW);
  delay(1000);
  encender_led:;
}
```



Caracteres tabla ASCII

Los MCU solo operan con 0 y 1. ¿Cómo podemos entonces trabajar con caracteres? En realidad, un MCU no entiende más allá de 0 y 1; por lo tanto, lo que se hizo fue una tabla para codificar caracteres en dígitos binarios. Es algo similar a lo que ya hemos visto con respecto a los números naturales y su correspondiente codificación en binario. Para ello disponemos de la tabla ASCII; es una tabla que consta de tres partes: la primera parte se conoce como «caracteres no imprimibles», la segunda, «imprimibles» y, la tercera, «extendido». Nosotros vamos a centrarnos en la segunda parte.

Si buscamos en Internet «tabla ASCII» [1], encontraremos una gran cantidad de resultados; en la siguiente tabla solo se presenta la parte de la tabla ASCII que se refiere a los caracteres imprimibles.

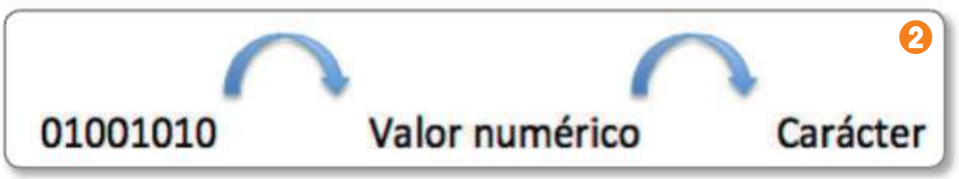
Lo que se hace con la tabla ASCII es asignar valores numéricos a caracteres [2]; es decir, los codificamos. Aún así, el MCU hace una segunda codificación, ya que pasa esos números al sistema binario. Vamos a suponer que enviamos el mensaje «Arduino», según la tabla sería: 65-114-100-117-105-110-111. Esta sería la codificación numérica que se envía, pero como el Arduino solo trabaja con 0 y 1.

La información con la que realmente trabajaría es la siguiente: 01000001-01110010-01100100-01110101-01101001-01101110- 01101111.

Es necesario tener esto en cuenta para cuando trabajemos con valores numéricos enviados por el puerto serie.

La variable más básica a la hora de trabajar con caracteres es la variable tipo char (8 bits), la cual puede almacenar un único carácter.

La tabla ASCII está dividida en tres partes, un Arduino puede trabajar con los caracteres de control y los imprimibles, ya que si miramos el valor numérico que es capaz de almacenar iría del -128 al 127 y esto deja fuera la parte ASCII extendido.



Código numérico	Carácter	Código numérico	Carácter	Código numérico	Carácter
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_	127	DEL
64	@	96			

Caracteres

- La palabra reservada para almacenar caracteres es char ❶, para asignarle un valor o bien se hace con el carácter entre comillas simples o bien por asignación del valor devuelto por una función u otra vara.
- Después de lo visto en el capítulo anterior, sabemos que cada carácter tiene su equivalente en valor numérico, por lo que asignamos el valor de una variable tipo char a una variable numérica, en este caso ❷ tipo byte, y obtenemos su posición en la tabla ASCII.
- Y esto era porque la codificación en binario de un carácter con su valor numérico equivalente en la tabla ASCII es el mismo; esto lo podemos comprobar si hacemos que el Arduino también envíe la codificación en binario de cada uno ❸.
- Por último, podemos programar nuestro Arduino para que recorra la tabla ASCII, devolviendo los caracteres imprimibles junto con su posición en la tabla ❹.
- En caso de almacenar un conjunto de caracteres, tenemos la opción de las cadenas de caracteres (String). Para asignar un valor este debe ir entre comillas dobles ❺.

❶
```
char caracter = 'A';
void setup()
{
  Serial.begin(9600);
  Serial.println(caracter);
}
void loop()
{
}
```

❷
```
char caracter = 'A';
byte valor_tabla_ASCII;
void setup()
{
  Serial.begin(9600);
  valor_tabla_ASCII = caracter;
  Serial.print(caracter);
  Serial.print("   ");
  Serial.println(valor_tabla_ASCII);
}
void loop()
{
}
```

```
A   65
```

3

```
char caracter = 'A';
void setup()
{
  Serial.begin(9600);
  for (byte i = 33; i < 128; i++)
  {
    caracter = i;
    Serial.print(caracter);
    Serial.print("->");
    Serial.println(i);
  }
}
void loop(){}
```

```
!->33
"->34
#->35
$->36
```

4

```
char caracter = 'A';
byte valor_tabla_ASCII;
void setup()
{
  Serial.begin(9600);
  valor_tabla_ASCII = caracter;
  Serial.print(caracter);
  Serial.print("(");
  Serial.print(caracter, BIN);
  Serial.print(")-");
  Serial.print(valor_tabla_ASCII);
  Serial.print("(");
  Serial.print(valor_tabla_ASCII, BIN);
  Serial.print(")");
}
void loop()
{
}
```

```
A(1000001)-65(1000001)
```

5

```
String caracteres = "Cadena de caracteres";
void setup()
{
  Serial.begin(9600);
  Serial.print(caracteres);
}
void loop() {}
```

```
/dev/cu.usbmoder
Cadena de caracteres
```

Caracteres de control

- Los caracteres de control no son imprimibles, no tienen una representación gráfica.
- Se utilizan para el de control de dispositivos, por ejemplo, impresoras.
- En la siguiente tabla ❶ se encuentran los caracteres de control junto con su codificación correspondiente en números enteros.
- Existen combinaciones de caracteres que permiten generar los caracteres de control.
- Salto de línea: \n ❷: si comparamos el enviar un mensaje desde un Arduino por el puerto serie a escribir un texto en Word, equivaldría a pulsar la tecla Enter.
- Con este caracter podemos substituir la instrucción Serial.println por Serial.print si añadimos al final un salto de línea; también podemos añadir saltos de línea a lo largo de un texto.
- Tabulación: \t ❸ : haciendo la misma comparación que en el caso anterior, equivaldría a una pulsación de la tecla de tabulación.
- Conociendo la codificación de cada carácter de control no es necesario conocer la cadena de caracteres equivalente.

❶

```
void setup()
{
  Serial.begin(9600);
  Serial.print("Añadir \n Salto de linea");
}
void loop() {}
```

/dev/cu.usbmode

```
Añadir
 Salto de linea
```

❷

```
void setup()
{
  Serial.begin(9600);
  Serial.print("Añadir \t tabulacion");
}
void loop() {}
```

/dev/cu.usbm

```
Añadir    tabulacion
```

0	Carácter Nulo	16	Escape Vínculo de Datos
1	Inicio Encabezado	17	Control de Dispositivo 1
2	Inicio Texto	18	Control de Dispositivo 2
3	Fin de Texto	19	Control de Dispositivo 3
4	Fin de Transmisión	20	Control de Dispositivo 4
5	Consulta	21	Acuse de Recibo Positivo
6	Acuse de Recibo	22	Sincronía de Espera
7	Timbre	23	Fin del Bloque de Transición
8	Retroceso	24	Cancelar
9	Tabulación Horizontal	25	Fin del Medio
10	Salo de línea	26	Substitución
11	Tabulación Vertical	27	Escape
12	Avance Página	28	Separador de Archivo
13	Retorno de Carro	29	Separador de Grupo
14	Desactivar Mayúsculas	30	Separador de Registro
15	Activar Mayúsculas	31	Separador de Unidad
		127	Suprimir

Transmisión de datos, comunicación serie

Para enviar datos desde el Arduino existen tres instrucciones:

- Serial.println(); Envía datos añadiendo un salto de línea.
- Serial.print(); Envía datos sin salto de línea.
- Serial.write(); Envía un único carácter sin salto de línea.

El envío de un dato del Arduino al exterior se hace carácter a carácter; cuando se ejecuta la instrucción Serial.print();, Serial.println(); o Serial.write();, lo que se hace es «trocear» el mensaje en caracteres y cada uno de ellos se almacena en una memoria (buffer) ❶, la cual se va vaciando en función de la velocidad a la que hayamos configurado el puerto serie.

El buffer tiene una capacidad de 63 bytes o, lo que es lo mismo, 63 caracteres.

Lo entenderemos mejor con un ejemplo ❷ ; esto no solo es importante para entender el envío de datos, sino la recepción, puesto que el comportamiento es el mismo.

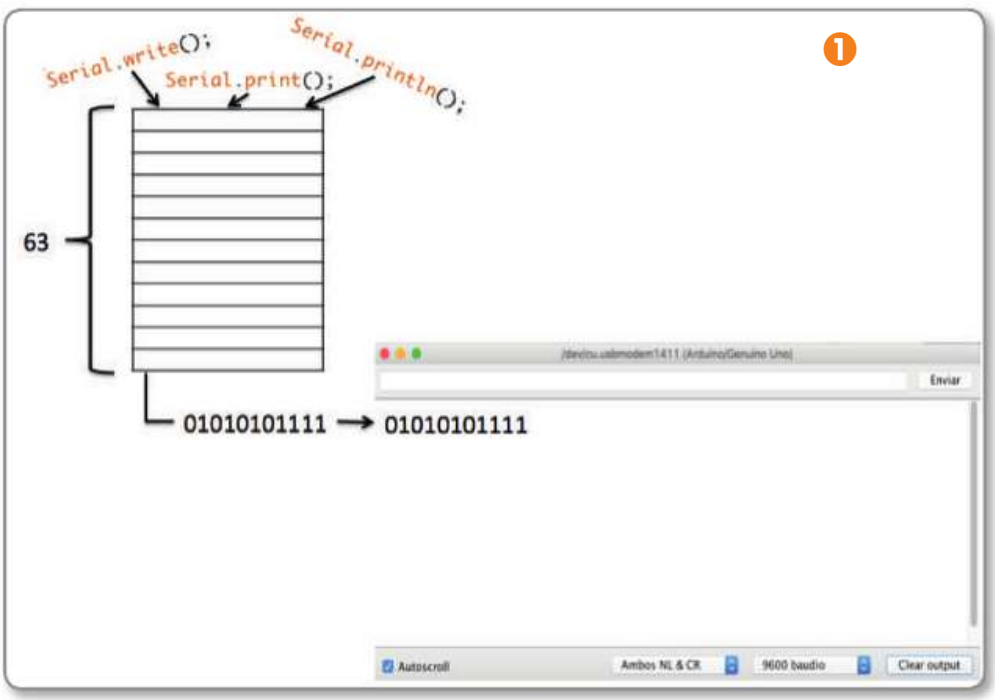
En caso de querer enviar más de 63 caracteres, pongamos por ejemplo 100, lo que hace la instrucción Serial.print o println es: escribir los 63 primeros caracteres en el buffer y según se vaya vaciando el buffer (por el envío de esos caracteres) la instrucción irá "soltando" uno a uno los 37 caracteres que quedaron pendientes de almacenar en el buffer.

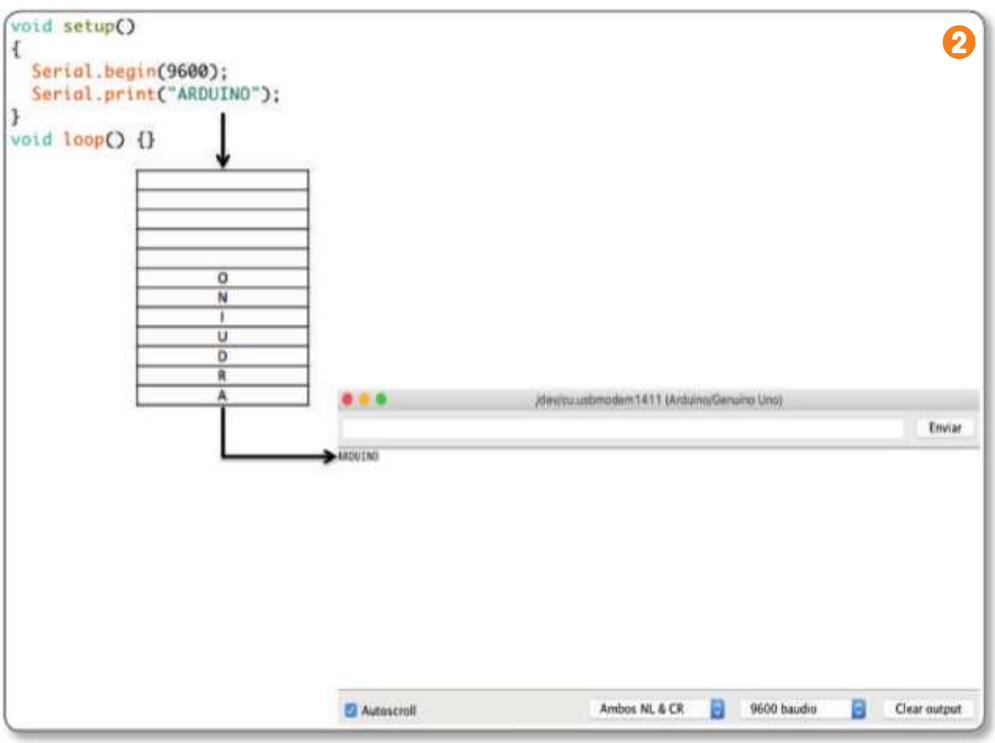
Hasta aquí quizás sea fácil de entender, lo que puede resultar un poco más complicado es el envío de números. Cuando se envía un número, lo que el Arduino hace es traducir cada dígito con la tabla ASCII y, luego, enviar cada dígito traducido ❸.

Es importante tener esta cuestión en cuenta a la hora de recibir valores del IDE, puesto que, si son números, antes de operar con ellos, debemos realizar la conversión correcta.

❸

Código numérico	Carácter
48	0
49	1
50	2
51	3
52	4
53	5
54	6
55	7
56	8


1
Serial.write();
Serial.print();
Serial.println();
63
01010101111
01010101111
Enviar
Autoscroll
Ambos NL & CR
9600 baudio
Clear output


2
void setup()
{
Serial.begin(9600);
Serial.print("ARDUINO");
}
void loop() {}
O
N
I
U
D
R
A
ARDUINO
Enviar
Autoscroll
Ambos NL & CR
9600 baudio
Clear output

Recepción datos, comunicación serie

Para recibir datos realizamos el proceso inverso, por lo que disponemos de una segunda memoria, el buffer de entrada, con otras 63 posiciones. Por lo tanto, para saber si nos han enviado datos, debemos comprobar si en esa memoria hay algún dato ❶.

Serial.availble();

- Esta instrucción no lleva parámetros y nos devuelve un 0 si el buffer de entrada está vacío, lo que significa que no hemos recibido nada. Si hay datos en el buffer, nos devuelve el número de posiciones ocupadas (nº de caracteres recibidos).

Una vez que sepamos si hay datos o no, debemos leerlos y almacenarlos en una variable, para que se eliminen del buffer de entrada y dejen espacio a otros datos que se puedan recibir. Si no lo hacemos, una vez que se llene el buffer de entrada (con 63 datos), cada carácter que nos envíen se perdería.

Serial.read();

- Esta instrucción nos devuelve el dato que se encuentre en la primera posición del buffer de entrada, es decir, lee el primer dato. Al leerlo, lo elimina del buffer y, automáticamente, si hay más datos, suben todos una posición, quedando siempre ocupada la primera posición por el siguiente dato recibido ❷.

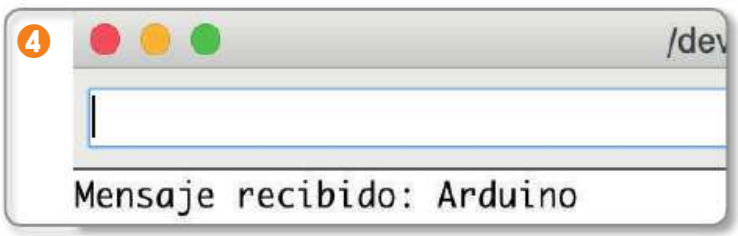
Con esto entendido, ya podemos enviar datos del IDE al Arduino y empezar a procesarlos.

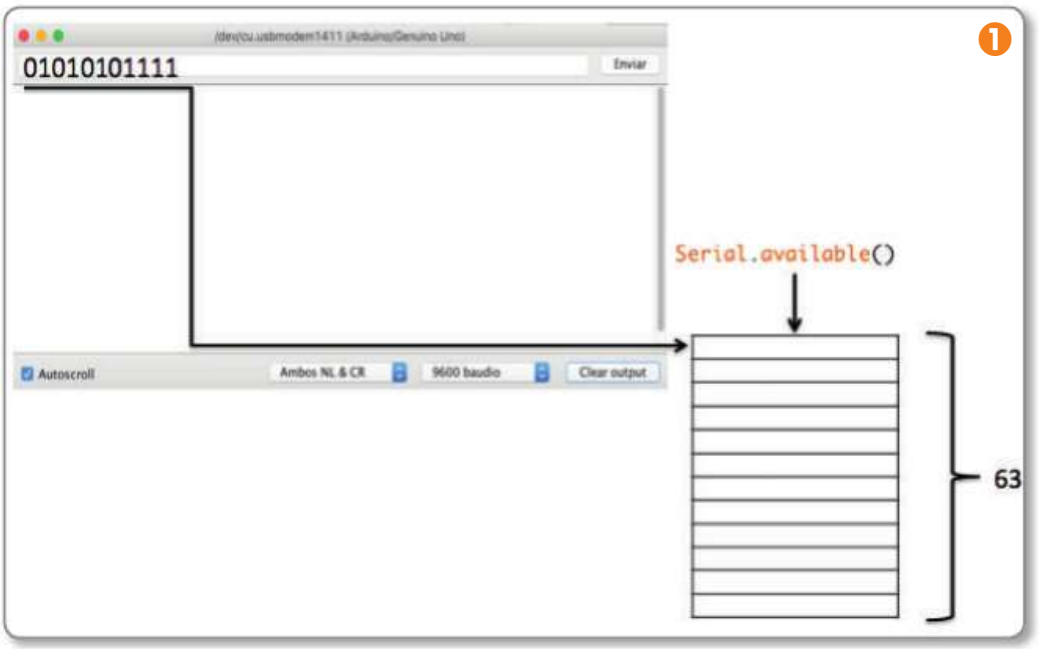
En el siguiente caso ❸, lo que hará el Arduino es reenviar las cadenas de texto que se envían desde el IDE, devolviéndolas con un texto añadido: «Mensaje recibido», más el mensaje.

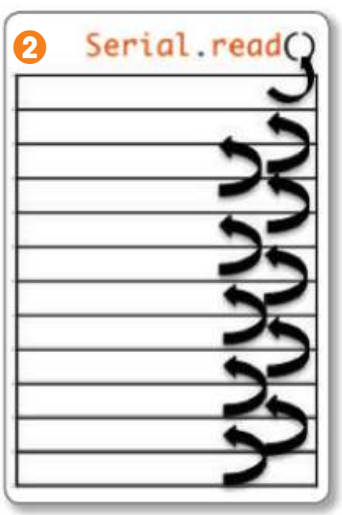
La estructura if comprueba si hay datos en el buffer; de ser así se ejecuta un bucle while para ir capturando cada uno de los datos. Para su correcto almacenamiento se utiliza una variable tipo char, que a continuación añade el valor obtenido a la cadena que contendrá todo el mensaje. Entre la lectura de cada carácter se pone un tiempo de espera para permitir que todos los caracteres lleguen al buffer.

Por último, se devuelve al IDE el mensaje recibido, empezando por la cadena <<Mensaje recibido >> ❹y, una vez enviado, se borra la cadena donde se almacena el mensaje para cuando se almacene el mensaje recibido no se tengan en cuenta los anteriores.

❹







3

```
String mensaje;
void setup() {
  Serial.begin(9600);
}
void loop()
{
  if (Serial.available() > 0)
  {
    while (Serial.available() > 0)
    {
      char caracter = Serial.read();
      mensaje += caracter;
      delay(3);
    }
    Serial.print("Mensaje recibido: ");
    Serial.println(mensaje);
    mensaje="";
  }
}
```

/dev/cu.usbmod

Arduino

Control de Arduino por puerto serie

Siendo capaces de almacenar datos que recibimos del exterior por el puerto serie, podemos controlar el funcionamiento del Arduino mediante el envío de órdenes.

Conectamos un led al pin 13 y, si el Arduino recibe el mensaje «ON», el led se encenderá y, si recibe «OFF», se apagará ❶.

Podemos programar un menú de control ❷, en el cual mostramos al usuario el abanico de órdenes que puede enviar.

1. Mostramos los mensajes del menú con todas las órdenes posibles, que en este caso son 4.
2. Para no estar continuamente mostrando la lista de órdenes, bloqueamos la ejecución del programa hasta que se reciba algún carácter, es decir, alguna orden.
3. No programamos un bucle que almacene una cadena de caracteres, puesto que cada orden se ejecuta con un único comando.
4. La estructura switch opera con valores numéricos, por eso no se pueden hacer 4 case comparando cada uno con un carácter.
5. Pero si nos vamos a la tabla ASCII, podemos saber el valor numérico equivalente a cada caracter.
6. Posibilidades : A(65), E(69), a(97) y d(100).
7. Con cada case asociado al comando correspondiente, solo queda efectuar la orden recibida.
8. En caso de que, por error, el usuario enviase un comando no válido, no se ejecuta ninguna orden y se envía un mensaje mostrando el error (operación realizada en el default).

En lo referente al control de un Arduino a través de un software mediante comunicación serie, lo visto en este capítulo es más que suficiente para un control simple.

El control simple permite que el Arduino reciba órdenes y las ejecute; en caso de que fuese un control que requiriese realizar algún tipo de cálculo, sería un poco más complejo puesto que ya nos obliga a realizar entre otras tareas conversiones de datos.

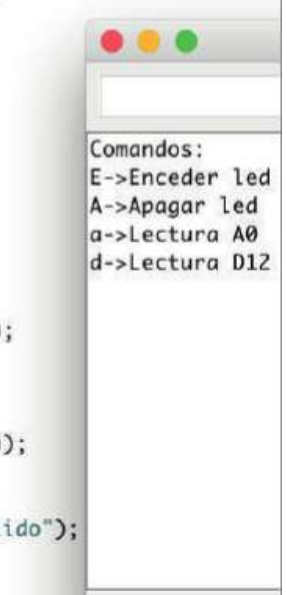
Pero de cara a un control simple, la programación de un Arduino no varía en absoluto, independientemente del software de control.

1

```
String orden;
void setup() {
  pinMode(13, OUTPUT);
  digitalWrite(13, LOW);
  Serial.begin(9600);
}
void loop()
{
  if (Serial.available() > 0)
  {
    while (Serial.available() > 0)
    {
      char caracter = Serial.read();
      orden += caracter;
      delay(3);
    }
    if (orden == "ON")
    {
      digitalWrite(13, HIGH);
    } else if (orden == "OFF")
    {
      digitalWrite(13, LOW);
    }
    orden="";
  }
}
```

2

```
void setup() {
  pinMode(13, OUTPUT);digitalWrite(13, LOW);
  pinMode(12,INPUT);
  Serial.begin(9600);
}
void loop()
{
  Serial.println("Comandos:");
  Serial.println("E->Enceder led\nA->Apagar led");
  Serial.println("a->Lectura A0 \nd->Lectura D12");
  while (Serial.available() == 0);
  char orden = Serial.read();
  switch(orden)
  {
    case 65:
      digitalWrite(13,LOW);
    break;
    case 69:
      digitalWrite(13,HIGH);
    break;
    case 97:
    Serial.print("Valor A0 ");
    Serial.println(analogRead(A0));
    break;
    case 100:
    Serial.print("Valor D13 ");
    Serial.println(digitalRead(12));
    break;
    default:
    Serial.println("Comando no válido");
  }
}
```



Conversión de datos

IMPORTANTE

La función serial.SetTimeout(); permite configurar el tiempo de espera para instrucciones como Serial.ParseInt();.

Vamos a regular un led a través del monitor serie; aquí es donde debemos tener en cuenta que sean letras, símbolos o números todo lo que se reciba por el puerto serie; se recibe carácter a carácter y codificado en la tabla ASCII.

No obstante, recordemos antes cómo trabajaba la instrucción analogWrite(): el primer parámetro indica sobre qué pin se genera la señal PWM, y el segundo parámetro, el valor numérico entre 0 y 255 (con un tamaño de 1 byte). Sabemos cómo se codifican números en el sistema binario.

Por ejemplo: el número 127 correspondería en binario a: 01111111. El problema radica en que por el puerto serie lo vamos a recibir dígito a dígito y codificado según la tabla ASCII.

Por lo tanto, si recibimos por el puerto serie el valor 127, ya no es un byte, sino tres bytes (00110001 00110010 00110111). ¿Cómo solucionamos esto? Lo que tenemos que hacer es almacenarlo en una cadena de caracteres y luego convertirlo en un número entero.

- toInt();

Esta función transforma una cadena de caracteres (en la que los caracteres solo pueden ser números) a su dato numérico equivalente❶.

Otra solución sería utilizar la función Serial.ParseInt() ❷;

- Esta instrucción muestrea el puerto serie en busca de valores numéricos.
- Por defecto espera durante 1 segundo (configurable), tras el cual si no detecta ningún número devuelve un 0.
- En el momento en el que detecta un número empieza a almacenar el resto de números que sigan llegando hasta encontrar un valor no número.
- Puede llegar a almacenar un valor tipo long.
- En caso de no encontrar valores numéricos, devuelve un 0.

Tenemos más tipos de conversiones, todas ellas necesitan recibir como parámetro el nombre de la variable a convertir: char(), byte(), ubt(), word(), long(), float().

```
String mensaje;                                    ❶
int valor_numerico;
void setup() {
  Serial.begin(9600);
  pinMode(11, OUTPUT);

}

void loop()
{

  if (Serial.available() > 0) {

    while (Serial.available() > 0) {

      char c = Serial.read();

      mensaje += c;

      delay(3);
    }
    valor_numerico=mensaje.toInt();
    mensaje="";
    analogWrite(11,valor_numerico);
  }
}
```

```
void setup() {                                     ❷
  pinMode(11, OUTPUT); digitalWrite(11, LOW);
  Serial.begin(9600);
}
void loop()
{
  while (Serial.available() == 0);
  byte pwm = Serial.parseInt();
  analogWrite(11, pwm);
}
```

Comunicación serie entre Arduinos

IMPORTANTE

Esperaremos a cargar el programa en ambos Arduinos para luego conectarlos entre sí. Con esto evitamos que aparezcan problemas a la hora de cargar un programa.

En este capítulo enviaremos datos entre Arduinos por comunicación serie. Para ello es necesario conectar el pin RX de uno con el pin TX del otro ❶. Asimismo, si no comparten la misma fuente de alimentación es necesario conectar los pines GND de cada uno.

- Lo que vamos a realizar en este caso es el control del led del Arduino A en función del mensaje enviado por el Arduino B.
- El Arduino que ejece el control envía cada segundo una orden de encendido y apagado (e y a respectivamente) ❷.
- El Arduino que recibe la orden tiene que diferenciar qué tarea debe realizar ❸.

Estableciendo este tipo de comunicación podemos tener un Arduino que recopile datos del exterior (sensores, módulos...) y que envíe las órdenes a otro Arduino que controla un entorno (actuadores). Una característica de este tipo de comunicación es la problemática de tener más de dos dispositivos conectados por el mismo puerto serie.

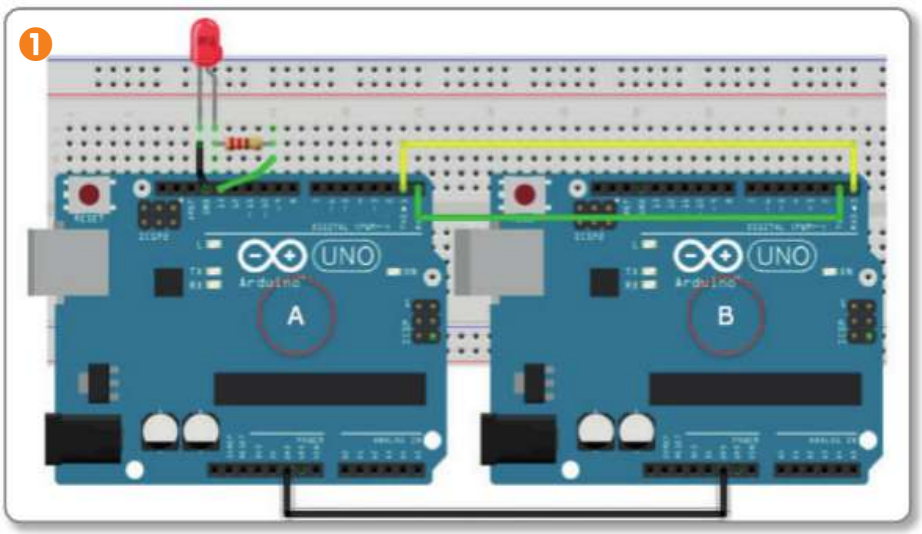
En próximos capítulos se creará un chat entre dos ordenadores mediante sus puertos series y un Arduino para cada PC. Con la solución que acabamos de realizar no podríamos llevar a cabo esta aplicación.

Para poder realizar un chat, la función que debe desempeñar cada Arduino es la recogida del mensaje escrito por su IDE, el envío al otro Arduino y la devolución al monitor serie, para luego reenviar el mensaje escrito desde el otro PC (a través de la comunicación entre Arduinos), reenviándolo al monitor serie.

El problema viene si solo trabajamos con un puerto para enviar y recibir mensajes, ya que por el canal de recepción estarían llegando mensajes tanto del IDE como del otro Arduino, sin posibilidad de poder diferenciar cuál es cuál.

La solución que se desarrollará en próximos capítulos consistirá en añadir otro puerto de comunicación con el que comunicarse entre Arduinos y dejar el puerto 0 para la comunicación con el ordenador . Con esta solución, el Arduino es capaz de diferenciar quién envía el mensaje puesto que los recibe por puertos diferentes.

1



2

```
void setup()
{
  Serial.begin(9600);
}
void loop()
{
  Serial.print("e");
  delay(1000);
  Serial.print("a");
  delay(1000);
}
```

3

```
void setup() {
  pinMode(13, OUTPUT);
  digitalWrite(13, LOW);
  Serial.begin(9600);
}
void loop()
{
  if (Serial.available() > 0)
  {
    char orden = Serial.read();
    if (orden == 'e')
    {
      digitalWrite(13, HIGH);
    } else if (orden == 'a')
    {
      digitalWrite(13, LOW);
    }

  }

}
```

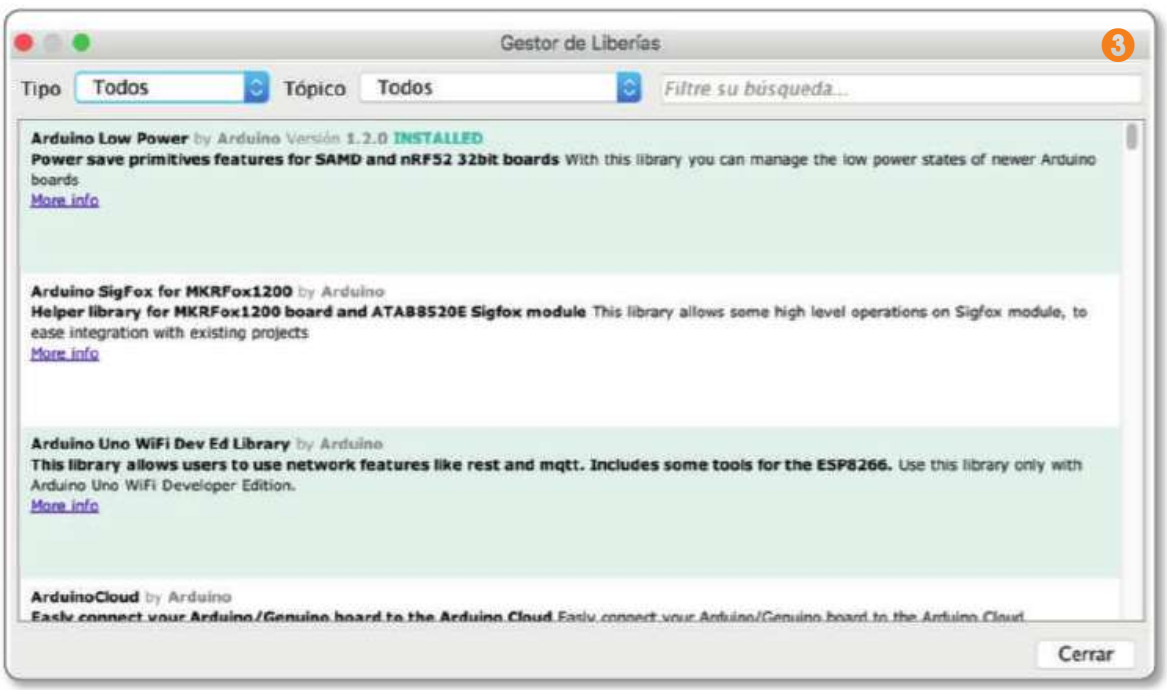
Librerías

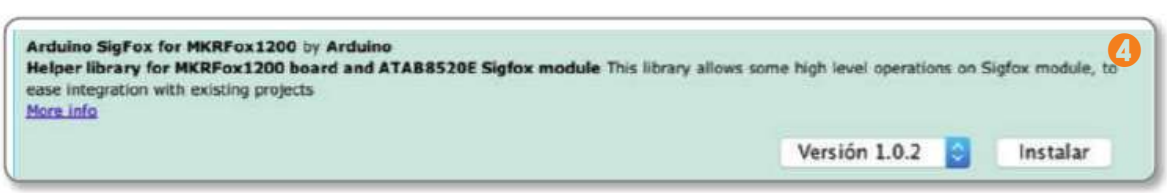
Una librería pone a nuestra disposición una serie de funciones con una programación compleja, que podemos utilizar para facilitar la tarea de programación. Vamos a ver qué operaciones podemos hacer con una librería; para ello, necesitamos ver otro apartado del IDE de Arduino.

- Si nos vamos a la barra de menú y al apartado «Programa» ❶, se abre un desplegable. En él, encontramos un apartado denominado «Incluir Librería».
- Si abrimos ese segundo desplegable ❷ encontramos tres apartados: el primero, «Gestionar Librerías», abre un repositorio que nos permite buscar una librería en concreto; si hemos encontrado algún caso práctico en una web y vemos que necesitamos una librería, podemos probar a buscarla en este apartado ❸.
- Disponemos de dos filtros de búsqueda e incluso un campo para escribir la librería que necesitamos. La instalación es sencilla: simplemente seleccionamos una librería, en el margen inferior derecho nos aparecen las versiones disponibles, en un desplegable, y un botón para instalar la versión seleccionada.
- Con pulsar el botón de instalar se inicia el proceso de instalación ❹, que tardará unos pocos segundos en finalizar. Si, por el contrario, queremos instalar una librería que hemos descargado de alguna web, seleccionamos el apartado de «Añadir librería .ZIP...»; buscamos el archivo descargado a través de un directorio y, al seleccionarlo y aceptar, se instala la librería.
- Independientemente de la forma de instalación, la consola del IDE nos informará de si se ha realizado la instalación.
- Por último nos aparece un listado con todas las librerías instaladas; muchas de ellas ya vienen instaladas al descargar el IDE de Arduino; son las que se conocen como librerías oficiales de Arduino.
- Las librerías que instalemos por nuestra cuenta se incluirán en esta lista de librerías, pero con un encabezado diferente. Ahora que sabemos cómo instalar librerías, tenemos que saber utilizarlas para tener acceso a sus funciones.

❶


2
Gestionar Librerías
Añadir librería .ZIP...
Arduino librerías
Arduino Low Power
Bridge
EEPROM
Esplora
Ethernet


Gestor de Liberías
3
Tipo
Todos
Tópico
Todos
Filtre su búsqueda...
Arduino Low Power by Arduino Versión 1.2.0 INSTALLED
Power save primitives features for SAMD and nRF52 32bit boards With this library you can manage the low power states of newer Arduino boards
More info
Arduino SigFox for MKRFox1200 by Arduino
Helper library for MKRFox1200 board and ATAB8520E Sigfox module This library allows some high level operations on Sigfox module, to ease integration with existing projects
More info
Arduino Uno WiFi Dev Ed Library by Arduino
This library allows users to use network features like rest and mqtt. Includes some tools for the ESP8266. Use this library only with Arduino Uno WiFi Developer Edition.
More info
ArduinoCloud by Arduino
Easly connect your Arduino/Genuino board to the Arduino Cloud Easly connect your Arduino/Genuino board to the Arduino Cloud
Cerrar


Arduino SigFox for MKRFox1200 by Arduino
Helper library for MKRFox1200 board and ATAB8520E Sigfox module This library allows some high level operations on Sigfox module, to ease integration with existing projects
More info
4
Versión 1.0.2
Instalar

SoftwareSerial

IMPORTANTE

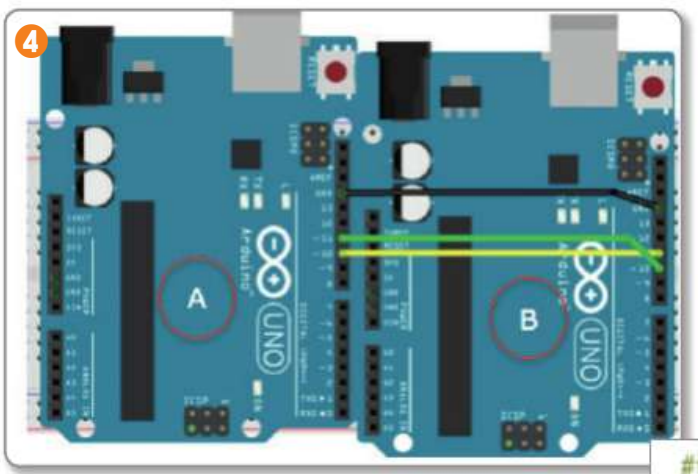
El motivo por el cual el capítulo anterior trató sobre librerías es por que son necesarias para solucionar el caso de un chat entre ordenadores.

El #include es una directiva para el compilador, de tal forma que pueda buscar el archivo referenciado por esta directiva, necesario para el uso de las funciones de la librería.

Cada librería implementa sus propias funciones o instrucciones, por lo que cada librería «es un mundo». Como al principio puede resultar complejo entender cada librería, disponemos de ejemplos para asimilar su funcionamiento ❶.

1. Creamos un puerto virtual con la librería <<SoftwareSerial>> ❷.
2. Para ello, debemos declarar la librería.
3. Si la seleccionamos y volvemos a nuestro programa, veremos cómo se hace referencia a ella al inicio.
4. Empecemos con la librería SoftwareSerial ❸; esta librería nos permite definir uno o varios puertos de comunicación serie adicionales a los que incorpore cada Arduino. Podemos entonces solventar el problema de utilizar un puerto del Arduino UNO para la comunicación con el IDE, y otro, para la comunicación con otro Arduino.
5. Lo primero que tenemos que hacer es definir qué pines se van a comportar como un puerto serie.
6. Para una correcta comunicación ❹ se debe conectar el pin 10 del Arduino A con el 11 del B y el 11 del A con el 10 del B, ya que por el 11 cada uno de ellos transmite datos, y por el 10 cada uno de ellos recibe datos.
7. Con este paso ya podemos trabajar con este puerto igual que hemos hecho con el Serial; la diferencia es que el prefijo es mySerial: mySerial.print();, mySerial.println();, mySerial.available();.
8. Con este programa ❺ conseguiremos un funcionamiento similar a un chat de WhatsApp: los mensajes enviados aparecerán en el margen derecho del monitor y los recibidos en el derecho.

4



2

3

```
#include <SoftwareSerial.h>

SoftwareSerial mySerial(10, 11); // RX, TX
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

5

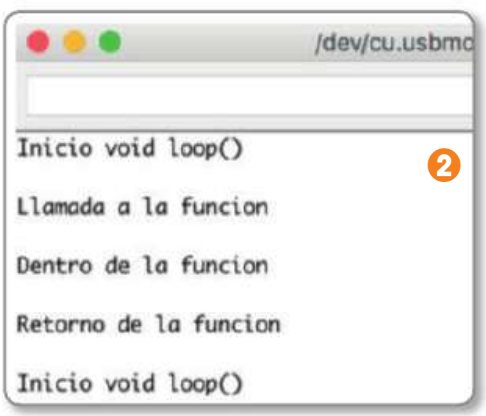
```
#include <SoftwareSerial.h>
SoftwareSerial  mySerial(10,11);
String mensaje_monitor;
String mensaje_otro_arduino;
void setup() {
  Serial.begin(9600);
  mySerial.begin(9600);
}

void loop() {
  if(Serial.available()>0)
  {
    while(Serial.available()>0)
    {
      delay(3);
      char c=Serial.read();
      mensaje_monitor+=c;
    }
    Serial.println("\t\t\t"+ mensaje_monitor);
    mySerial.print(mensaje_monitor);
    mensaje_monitor="";
  }
  if(mySerial.available()>0)
  {
    while(mySerial.available()>0)
    {
      delay(3);
      char c=mySerial.read();
      mensaje_otro_arduino+=c;
    }
    Serial.println( mensaje_otro_arduino);
    mensaje_otro_arduino="";
  }
}
```

Funciones

- Una función no es más que un trozo de código que se identifica con un nombre y que, para que se ejecute, debe ser «llamado». Con esto podemos escribir líneas de código fuera del void setup y void loop, que deben estar identificadas por un nombre y que se ejecutarán si se las «llama» desde otra parte del programa ❶.
- Si abrimos el monitor serie, veremos la siguiente secuencia de mensajes ❷.
- Lo que vemos con este primer caso es cómo, al «llamar» a una función, el programa «sale» de la ejecución de líneas de código del programa principal (void loop) y ejecuta otra parte de código.
- Para indicar que se llama a una función se utiliza un rectángulo con doble rayado y se escribe el nombre de la función.
- Las funciones nos permiten «trocear» un programa en partes e identificarlas según la operación que se realice en cada una de ellas ❸. Esto nos permite escribir programas mucho más legibles, además de optimizar la programación. Si, a lo largo de un programa, repetimos una serie de operaciones, podemos englobarlas todas en una función y «llamar» a la función que sea necesaria.
- Por consiguiente, las funciones son un elemento importante para optimizar un programa y hacerlo más legible.
- En cuanto al nombre que pueden llevar las funciones, seguiremos las mismas reglas en relación a la declaración de variables.

Las funciones, junto con los comentarios, asignación de nombres identificativos a las variables según su uso, y diagramas de flujo son un conjunto de técnicas o herramientas importantes de conocer para desarrollar un programa.

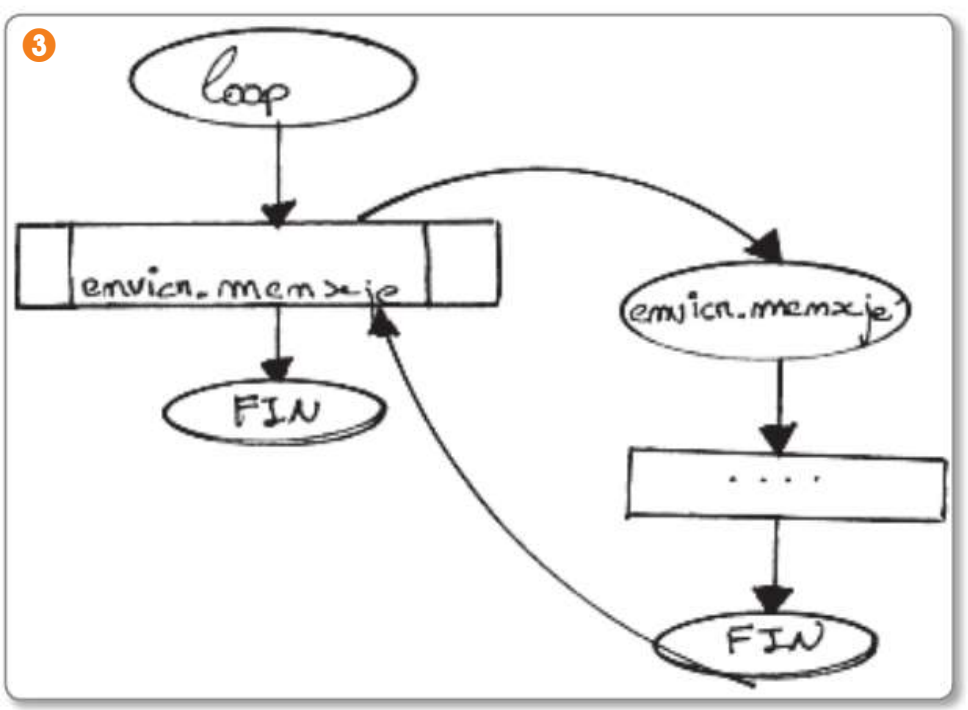
En próximos capítulos veremos que las funciones dan mucho de sí ya que tendremos tipos de funciones, se le podrán enviar datos e incluso pueden devolver el resultado de las operaciones que lleven a cabo.



1

```
void setup() {
  Serial.begin(9600);
}
void loop() {
  /*Mostraremos la ejecución del programa mediante mensajes
    por el puerto serie*/
  Serial.println("Inicio void loop()\n");
  delay(2000);
  Serial.println("Llamada a la funcion\n");
  delay(2000);
  enviar_mensaje();
  Serial.println("Retorno de la funcion\n");
  delay(2000);
}
void enviar_mensaje() {
  Serial.println("Dentro de la funcion\n");
  delay(2000);
}
```

3



Funciones con valor de retorno

La función tipo void loop ejecutaba el código que había entre sus llaves y, al finalizar, volvía al inicio. Las funciones permiten devolver un valor (tipo string, int...) en la línea desde la que son «llamadas».

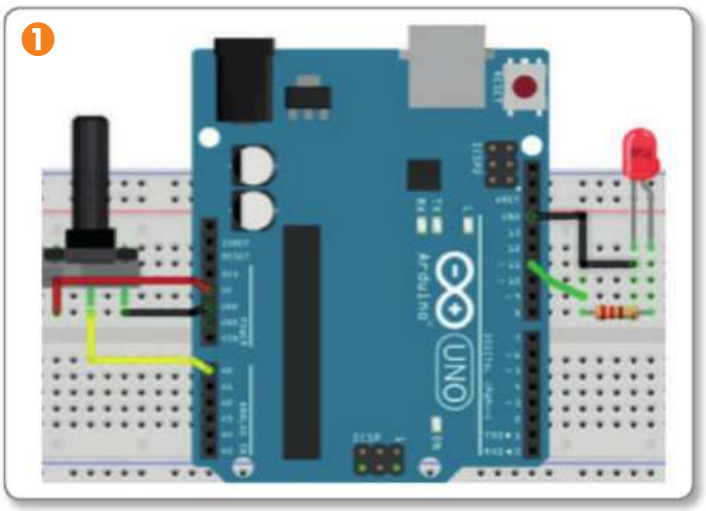
Para ello, las funciones no deben ser declaradas tipo void, sino que deben ser declaradas como algún tipo de variable, es decir: tipo int, string, etc.

Al declarar así una función, el valor que devuelve es del tipo que sea la función, es decir: una función tipo int devuelve un valor tipo int. Es «obligación» del desarrollador almacenar el valor que devuelve la función.

Por último, ¿cómo se hace para devolver un valor? Para ello, se necesita la instrucción return, la cual indica que el valor que la acompaña es el que se devuelve. Veamos todo esto en un ejemplo.

- Conectamos el siguiente esquema ❶.
- Cargando este programa ❷ el led se regulará en función del valor del potenciómetro.
- Este no es el programa que mejor optimiza el programa ya que en capítulos anteriores hemos hecho la misma aplicación con menos programación, pero para entender cómo una función devuelve un valor resulta útil.
- Al inicio del void loop se almacena el valor de potenciómetro en la variable entrada_A0; este valor viene de una función.
- La función lectura_A0, ya no es de tipo void sino de tipo int para que así puede devolver un resultado.
- Como este caso es sencillo la función no tiene más que una línea de código que ya es la línea que devuelve el valor de la función.
- Al lado de return se toma la lectura de la entrada analógica para que esta sea el valor que devuelve la función.
- El tipo de función debe ser acorde con el tipo de valor que se devuelve, por ejemplo, en este caso devolviendo un número entero la función no puede ser de tipo String.
- Teniendo claro que el valor que se devuelve es tipo int (como mínimo podría ser long, unsigned long...) la función será de tipo int y también lo debe ser el valor donde se almacena, por eso entrada_A0 es de tipo int.

1



2

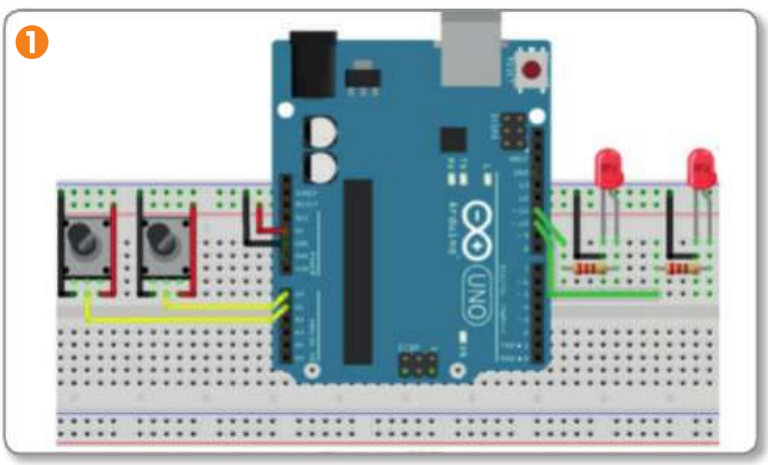
```
void setup()
{
  pinMode(11, OUTPUT);
  digitalWrite(11, LOW);
}
void loop()
{
  int entrada_A0 = lectura_A0();
  analogWrite(11,map(entrada_A0, 0, 1023, 0, 255));
}

int lectura_A0()
{
  return analogRead(A0);
}
```

Pasar parámetros a una función

Una función no solo puede devolver datos, sino que al ser llamada, se le pueden pasar datos. Para ello debe haber una concordancia entre el valor que se envía y la variable en la que se almacena, es decir, ambos deben ser del mismo tipo.

- En este caso se regularán dos leds en función de una entrada analógica diferente para cada uno ❶. El led 11 se regula con la entrada A0 y el 10 con la entrada A1.
- En el programa ❷ se define una función que calcula y devuelve el valor de la PWM correspondiente a la entrada analógica que se le indica.
- Desde el programa principal se "llama" a la función PWM indicándole sobre qué entrada analógica debe calcular la PWM.
- En el caso del led 11 se le pasa como parámetro el valor 0 y en el caso del 10 el valor 1.
- Esos valores serán recibidos por la función y almacenados en la variable entrada. Como los valores posibles que puede tener ese parámetro son 0 ó 1, con almacenarlos en una variable tipo boolean es más que suficiente
- Cambiando de caso ❸, podemos pasar a una función el número de parámetros que necesitemos y todos los datos pueden ser, o no, del mismo tipo. Mantenemos el mismo circuito y cargamos el siguiente programa.
- Sí se envían varios parámetros es importante tener en cuenta el orden, puesto que se almacenan también en orden dentro de la función.
- Otro punto a tener en cuenta es que no es necesario que una función devuelva un valor para que pueda recibir valores, es decir, puede ser de tipo void y recibir valores.
- En este caso los dos parámetros que se envían son del mismo tipo, pero pueden ser de tipos diferentes; eso sí, las variables en las que se almacene cada valor deberá concordar con el tipo correspondiente.

❶



2

```
void setup()
{
  pinMode(11, OUTPUT);
  digitalWrite(11, LOW);
  pinMode(10, OUTPUT);
  digitalWrite(10, LOW);
}
void loop()
{
  analogWrite(11, pwm(0));
  analogWrite(10, pwm(1));
}

byte pwm(boolean entrada)
{
  return map(analogRead(entrada), 0, 1023, 0, 25
}
```

3

```
void setup()
{
  pinMode(11, OUTPUT);
  digitalWrite(11, LOW);
  pinMode(10, OUTPUT);
  digitalWrite(10, LOW);
}
void loop()
{
  pwm(analogRead(A0), analogRead(A1));
}

void pwm(int entrada_A0, int entrada_A1)
{
  analogWrite(11, map(entrada_A0, 0, 1023, 0, 255));
  analogWrite(10, map(entrada_A1, 0, 1023, 0, 255));
}
```

Pestañas IDE Arduino

El IDE de Arduino dispone de una herramienta que permite escribir funciones en otras «pestañas» y así dejar la «hoja» en la que escribimos el programa principal para declarar librerías, variables y las partes básicas del programa (loop y setup).

1. Seleccionamos, en el desplegable, «Nueva Pestaña» ❶,
2. En la parte inferior del IDE, antes de la consola, surge una ventana ❷.
3. Cuando creamos una pestaña, el nombre de esta sigue las mismas reglas que las de los nombres de las variables ❸.
4. Entonces se crea una pestaña en blanco con el nombre que le asignamos ❹. El único funcionamiento de esta herramienta es separar el programa en «hojas» diferentes para hacerlo más legible; en realidad no cambia el funcionamiento del programa.
5. En las nuevas pestañas no se pueden utilizar directivas (constantes, librerías..) y tampoco declarar variables globales. Todo esto se debe realizar en la pestaña principal.
6. Dentro de cada pestaña se pueden programar una o más funciones, y dentro de ellas sí es posible declarar variables locales.
7. Es conveniente crear pestañas en el momento en el que el programa empiece a ser demasiado extenso .
8. Por cada pestaña nueva se crea un archivo asociado con extensión .ino, estos archivos se encontrarán en la carpeta del programa.
9. Será el compilador el encargado en empaquetar todos esos archivos y cargarlos como si fueran un único programa al MCU.


1
Nueva Pestaña ⇧⌘N
Renombrar
Borrar
Pestaña Anterior ⌥⌘←
Pestaña Siguiente ⌥⌘→
sketch_oct12e


2
Nombre del nuevo fichero:
Ok
Cancelar


3
Nombre del nuevo fichero:
Ventana_funciones
Ok
Cancelar


4
sketch_oct12e §
Ventana_funciones §

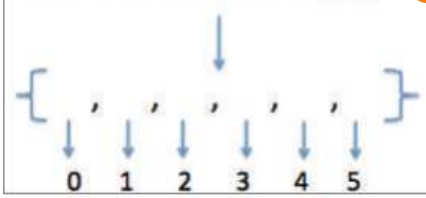
Vectores

Hasta ahora, si almacenábamos datos en diferentes espacios de memoria era necesario declarar una variable por cada dato; con los arrays, este proceso se agiliza, ya que nos permite estructurar los datos. Un array es un vector que puede almacenar más de un dato; sin embargo, en un mismo array (vector) solo puede haber datos del mismo tipo.

- Al igual que una variable, los vectores deben ser declarados. Su declaración es prácticamente idéntica a la de una variable; por ejemplo, vamos a declarar un vector de números enteros ❶.
- En el caso anterior, declaramos un vector en el cual se pueden almacenar datos tipo «int», y en concreto se pueden almacenar 6 datos.
- Para poder trabajar con los datos (modificarlos o leerlos) trabajamos con el índice del vector; ❷ es decir, debemos saber en qué posición hemos almacenado cada dato para después acceder a esa posición y operar con él.
- Por defecto, en cada posición, si no «escribimos» nada, muestra un cero. Podemos declarar de diferentes formas un vector, pero para escribir datos en una posición determinada solo existe una forma: seleccionar la posición del vector y asignar un valor.
- Se puede asignar valor a todas las posiciones o bien posición a posición, indicándolo mediante el índice de la posición al igual que se hacía con la lectura ❸.
- Por último, al igual que ocurre en matemáticas, los vectores pueden tener varias dimensiones, lo que los convierte en matrices.
- Por lo tanto, si queremos crear tres filas debemos crear tres campos (de 0 a 2 posiciones) en el vector y, dentro de cada campo, definiremos el valor de las columnas ❹.

```
int vector_enteros[6];                                    ❶

void setup() {
  Serial.begin(9600);
  Serial.println("Recorremos el vector");
  for (int indice;indice<6;indice++){
    Serial.println(vector_enteros[indice]);
  }

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

```
int vector_enteros[6];
```




```
int vector_enteros[]={0,1,2,3,4,5};
```

```
int vector_enteros[3][3] = {{0, 0, 0}, {1, 1, 1}, {2, 2, 2}};
int fila;
int columna;
void setup() {
  Serial.begin(9600);
  Serial.println("Recorremos la matriz");
  for (fila = 0;  fila < 3 ; fila++) {

    for (columna = 0; columna < 3; columna++) {
      Serial.print(vector_enteros[fila][columna]);
    }
    Serial.println();
  }


}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Gráficas puerto serie

IMPORTANTE

No se puede tener abierto el monitor serie y el serial plotter al mismo tiempo.

En muchas aplicaciones nos resultará más útil visualizar los datos monitorizados por el Arduino en gráficas que directamente su valor numérico (a través del monitor serie).

El IDE de Arduino dispone de una herramienta para representar los datos que reciba por el puerto serie <<Serial Plotter>>. Esta extensión la encontramos en el desplegable de la opción de herramientas del IDE❶.

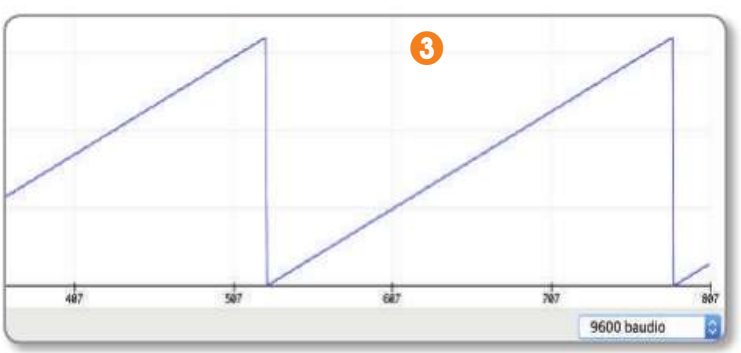
1. Empezaremos por graficar un único dato❷.
2. Como el canal de comunicación sigue siendo la comunicación serie, debemos inicializarla y, para que el dato se represente, se tiene que enviar con la instrucción Serial.println.
3. A continuación abrimos el Serial Plotter❸ y al igual que ocurría con el monitor serie se abre una ventana.
4. La ventana del Serial Plotter es más sencilla que la del monitor serie, simplemente dispone de un desplegable para configurar la velocidad de comunicación y la parte en la que se grafica la información.
5. La escala de valores se ajusta automáticamente en función de los valores que se vayan recibiendo.
6. Para representar varios datos la programación se complica un poco, siendo necesario enviar cada dato con Serial,print, entre dato y dato una coma y el último dato se enviará con Serial.println.
7. Con el siguiente programa❹ se enviará el valor de dos contadores, los cuales cambian de valor de forma inversa.
8. Cada dato se representará con un color diferente❺, para saber qué color corresponde a cada valor es importante saber en qué orden se envían.
9. En este caso se envía primero el valor del contador1, por lo tanto se le asigna el color azul y al contador2 el color rojo.
10. En el margen superior derecho encontramos el orden de los colores.

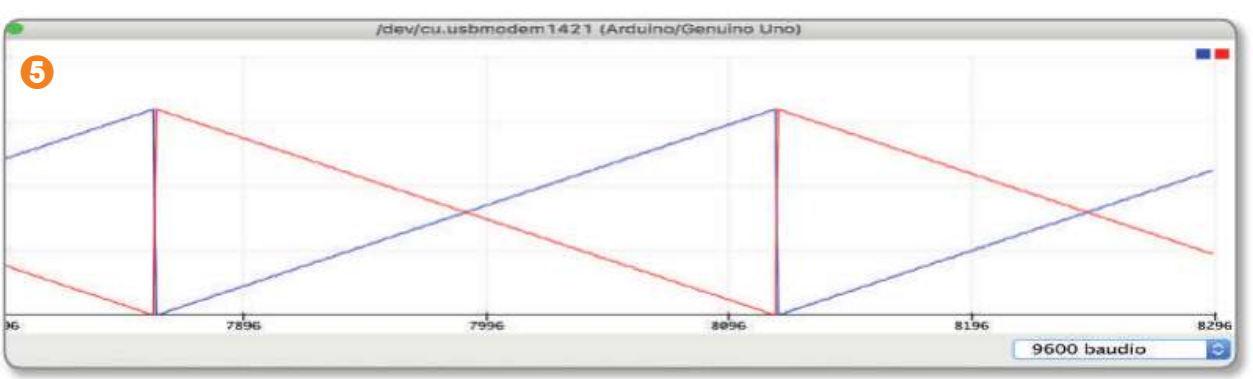
2

```
byte contador;

void setup()
{
  Serial.begin(9600);
}
void loop()
{
  Serial.println(contador);
  contador++;
  delay(10);
}
```

4

```
byte contador1;
byte contador2 = 255;
void setup()
{
  Serial.begin(9600);
}
void loop()
{
  Serial.print(contador1);
  Serial.print(",");
  Serial.println(contador2)
  contador1++;
  contador2--;
  delay(10);
}
```





Instrucciones matemáticas

Para generar una señal PWM en función del valor de una entrada analógica utilizamos la instrucción map, que permitía mapear el valor de entrada y así obtener su correspondiente valor proporcional entre un nuevo rango de valores

El lenguaje de programación de Arduino dispone de más instrucciones para poder realizar cálculos:

- abs(); devuelve el valor absoluto de un número ❶.
- min(); devuelve el valor mínimo de dos números ❷.
- max(); devuelve el valor máximo de dos números ❸.
- sqrt(); opera con un parámetro y devuelve la raíz cuadrada de este ❹.
- constrain(); opera con 3 parámetros, el primero es el dato con el que se realiza la operación. Permite forzar que el dato se encuentre entre un rango, ese rango lo definen el parámetro 2 y 3. En caso de que el dato se encuentre dentro del rango, la instrucción devuelve el propio valor del dato; si por el contrario se encuentra por debajo o por encima, devolvería el valor mínimo y máximo del rango respectivamente ❺.
- pow(); opera con dos parámetros y se utiliza para realizar potencias, el primer parámetro sería la base y el segundo el exponente ❻.
- sq(); opera con un parámetro y devuelve el valor de este elevado al cuadrado ❼.

❶
```
void setup()
{
  Serial.begin(9600);
}
void loop()
{
  Serial.println(abs(-100));
}
```

❷
```
void setup()
{
  Serial.begin(9600);
}
void loop()
{
  Serial.println(min(-5,10));
}
```

3
```
void setup()
{
  Serial.begin(9600);
}
void loop()
{
  Serial.println(max(-5,10));
}
```

4
```
void setup()
{
  Serial.begin(9600);
}
void loop()
{
  Serial.println(sqrt(25));
}
```

5
```
void setup()
{
  Serial.begin(9600);
}
void loop()
{
  Serial.print(constrain(-5, 0, 20));
  Serial.print("    ");
  Serial.print(constrain(10, 0, 20));
  Serial.print("    ");
  Serial.println(constrain(25, 0, 20));
  delay(1000);
}
```

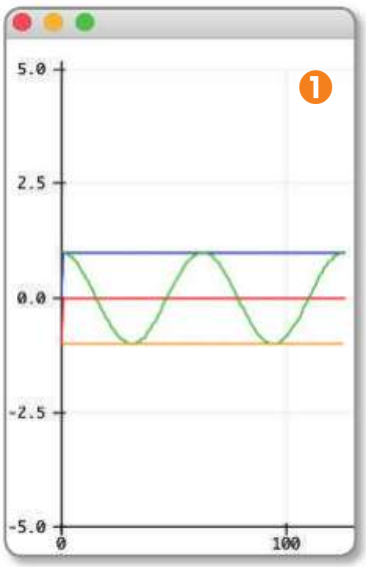
6
```
void setup()
{
  Serial.begin(9600);
}
void loop()
{
  Serial.println(pow(2,5));
}
```

7
```
void setup()
{
  Serial.begin(9600);
}
void loop()
{
  Serial.println(sq(10));
}
```

Instrucciones trigonométricas

IMPORTANTE

Los ángulos para cualquiera de las funciones deben ir expresados en radianes



Para realizar operaciones trigonométricas disponemos de las instrucciones cos, sin y tan.

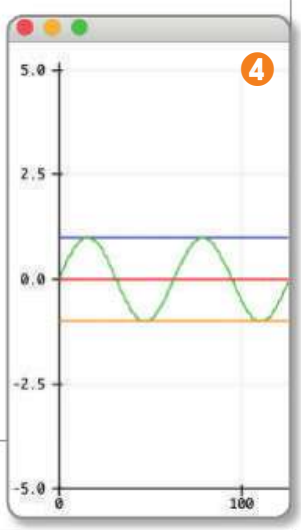
- cos(); devuelve el coseno del ángulo pasado como parámetro ❶.
- En este caso visualizaremos el coseno recorriendo sus posibles valores desde 360 a 0º, pero en radianes e incrementando el valor inicial de 0.1 en 0.1 ❷.
- sin(); devuelve el seno del ángulo pasado como parámetro ❸.
- En este caso visualizaremos el seno recorriendo sus posibles valores desde 360 a 0º, pero en radianes e incrementando el valor inicial de 0.1 en 0.1 ❹.
- tan(); devuelve la tangente del ángulo pasado como parámetro ❺.
- En este caso visualizaremos la tangente recorriendo sus posibles valores desde 360 a 0º, pero en radianes e incrementando el valor inicial de 0.1 en 0.1 ❻.

❷

```
void setup()
{
  Serial.begin(9600);

  for (float angulo = -PI * 2; angulo <= PI * 2; angulo = angulo + 0.1)
  {
    Serial.print(1);
    Serial.print(",");
    Serial.print(0);
    Serial.print(",");
    Serial.print(cos(angulo));
    Serial.print(",");
    Serial.println(-1);
    delay(10);
  }
}
void loop() {  }
```

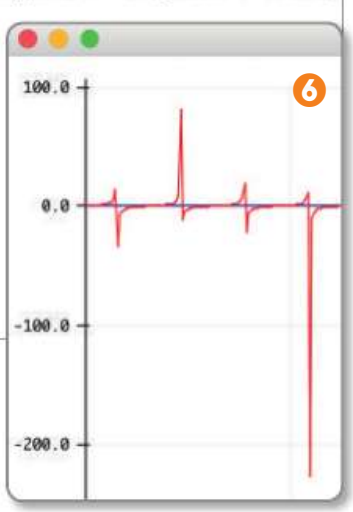
```
void setup()
{
  Serial.begin(9600);

  for (float angulo = -PI * 2; angulo <= PI * 2; angulo = angulo + 0.1)
  {
    Serial.print(1);
    Serial.print(",");
    Serial.print(0);
    Serial.print(",");
    Serial.print(sin(angulo));
    Serial.print(",");
    Serial.println(-1);
    delay(10);
  }
}
void loop() {  }
```

3




```
void setup()
{
  Serial.begin(9600);

  for (float angulo = -PI * 2; angulo <= PI * 2; angulo = angulo + 0.1)
  {

    Serial.print(0);
    Serial.print(",");
    Serial.println(tan(angulo));
    delay(10);
  }
}
void loop() {  }
```

5



Generar números pseudoaleatorios

El lenguaje de programación de Arduino, dispone de varias instrucciones para generar números pseudoaleatorios.

- randomSeed(); esta instrucción iniciliza el generador de números pseudoaleatorios, fija un valor inicial (tipo long) trás el cual se empezarán a generar números Siempre que inicializamos un generador de números pseudoaleatorios con el mismo valor la secuencia de números que se generan serán los mismos .

- Si ese es nuestro objetivo configuraremos esta instrucción en el void setup con un valor determinado y así en los diferentes reinicios del Arduino la secuencia de números que se generan serán los mismos.

- En caso contrario podemos inicializar su valor con la lectura de una entrada analógica que no se encuentre conectada a nada. Consigiendo así que entre diferentes reinicios el valor inicial sea diferente y por lo tanto la secuencia de números que se generen a partir de él también

- random(); Con el generador de números pseudoaletarios inicializado, esta instrucción devuelve un número que se encuentre entre los dos valores que se le pasen como parámetro.

- En este primer caso ❶, se programa un generador de números pseudoaleatorios con un valor inicial igual para cualquier reinicio del Arduino. Por lo que en los diferentes encendidos del Arduino, visualizaremos los mismos 10 números.

- Por el contrario en este caso ❷ se programa un generador de números pseudoaleatorios con un valor inicial que varía en función de una entrada analógica que no tendremos conectada a nada. Por lo que en los diferentes encendidos del Arduino, visualizaremos una secuencia de 10 números diferentes.

1

```
void setup()
{
  Serial.begin(9600);
  randomSeed(20);
  for (byte i = 0; i <= 10; i++)
  {
    Serial.println(random(-100, 1000));
    delay(3);
  }
}
void loop() {  }
```

```
540
886
655
737
930
352
380
349
749
534
871
```

2

```
void setup()
{
  Serial.begin(9600);
  randomSeed(analogRead(A4));
  for (byte i = 0; i <= 10; i++)
  {
    Serial.println(random(-100, 1000));
    delay(3);
  }
}
void loop() {  }
```

```
295
-96
403
696
24
138
606
-33
592
474
205
```

Reset

Resetar un Arduino permite que se pueda reprogramar, el reseteo pues, se puede producir por el IDE para realizar una programación, cada vez que abrimos el puerto serie o bien mediante la pulsación del botón de reset. También es posible programar un reset debido a la detección de un error, pero al nivel que nos encontramos no ahondaremos en este mecanismo.

Si revisamos el hardware del Arduino UNO ❶, en concreto los pines de potencia, encontraremos un pin correspondiente al RESET, a través de este pin podemos controlar el reset de un Arduino.

Puede resultar un inconveniente que cada vez que abrimos el monitor serie el Arduino se resetee, por lo que en la web de Arduino nos explican como solucionar este inconveniente. (https://playground.arduino.cc/Main/DisablingAutoResetOnSerialConnection).

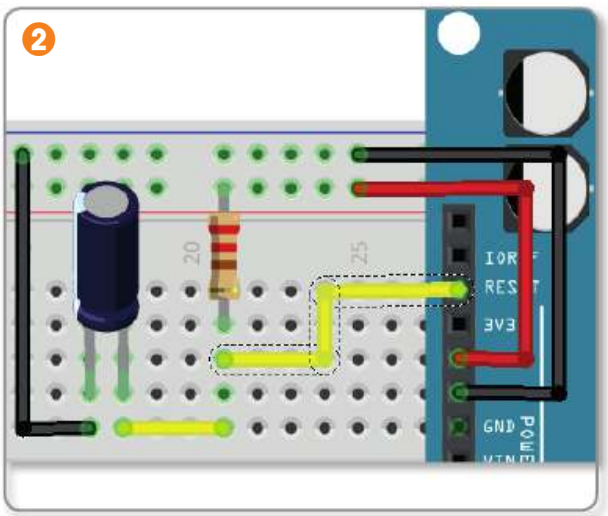
Nos indican que debemos conectar una resistencia de 120 Ω entre los pines de reset y 5V y un condensador de 10 µF entre GND y el pin de Reset ❷.

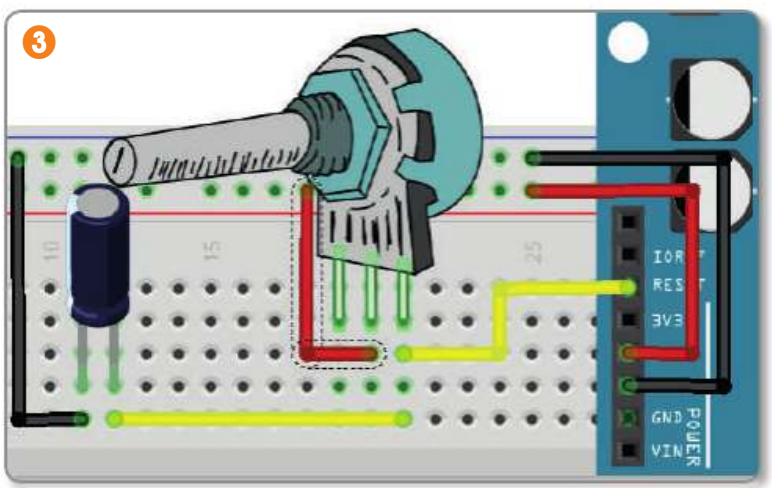
Como una resistencia de 120 Ω no es muy común, esto nos obliga a conectar resistencias de tal forma que la resistencia equivalente sea de 120 Ω, pero esto lo podemos hacer de un forma más sencilla con un potenciómetro.

1. Cogemos un multímetro y un potenciómetro (por ejemplo de 1k Ω nunca inferior a 120 Ω), medimos la resistencia de contacto entre la patilla central y una de las dos de los extremos y movemos el eje del potenciómetro hasta alcanzar el valor de 120 Ω o aproximado entre 110 y 124 Ω.

2. Con esas dos patillas nos vamos a la protoboard y realizamos las conexiones, substituyendo la resistencia por el potenciómetro ❸.

3. Una vez realizada la conexión, podemos probar a abrir la comunicación serie y ver que no se resetea el Arduino ❹.





4

```
void setup()
{
  Serial.begin(9600);
  Serial.println("Arduino reseteado");
}

void loop() {}
```

Pin AREF

- El pin AREF ofrece un voltaje de referencia para poder adaptar el rango de medida de las entradas analógicas. El valor del voltaje por defecto es de 5 voltios, lo que, unido al hecho de que las entradas analógicas tienen una resolución de 10 bits, nos permite cuantificar incrementos de tensión de 5 mV (5/1023).
- En el Arduino UNO no podemos variar la resolución, pero si adaptarla. Si, por ejemplo, tenemos una señal analógica que tenemos que leer y sabemos que, en ningún caso, superará los 3 voltios, sí podemos adaptar nuestra resolución a un rango de entre 0 a 3 voltios, conseguimos detectar incrementos más pequeños (3/1023 = 3 mV, aprox.).

analogReference(parámetro);

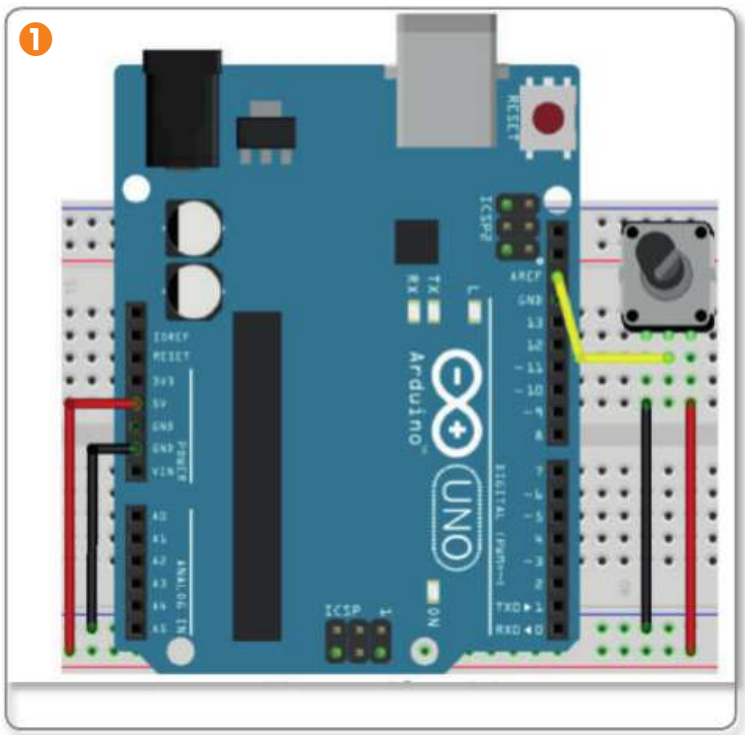
- Con esta función, podemos configurar la tensión de referencia de las entradas analógicas. El parámetro permite tres posibles valores:
- DEFAULT: configura todas las entradas analógicas entre un rango de 0 a 5 voltios. Si no hemos cambiado la referencia de las entradas, no tendría sentido, puesto que este es el valor al cual se encontraría por defecto.
- INTERNAL: voltaje de referencia 1,1 voltios; por lo tanto, las entradas analógicas no pueden superar esta tensión y se podrían cuantificar incrementos de : 1,1/1023 = 1 mV, aprox.
- EXTERNAL: permite referenciar las entradas en función de la tensión del pin AREF(tensión que debe estar entre 0 y 5 voltios). Por lo tanto, si a la entrada del pin AREF conectamos una señal de 2,5 voltios y configuramos el «analogReference(EXTERNAL);», los valores de las entradas analógicas deben encontrarse entre 0 y 2,5 voltios, con una resolución de: 2,5/1023 = 2,5 mV, aprox.

1. Podemos regular un potenciómetro con la tensión de referencia deseada y conectar su salida al pin AREF ❶.
2. Para que el Arduino trabaje con esa referencia se lo indicamos por programación ❷.

- analogReadResolution(parámetro);Esta función sí que permite definir la resolución de las entradas analógicas, pero, en el caso del Arduino UNO, no podemos modificar la resolución.
- analogWriteResolution(parámetro); Modifica la resolución de las señales PWM (por defecto, 8 bits), pero, al igual que el caso anterior, no es posible modificarlo en el Arduino UNO.

081

1



IMPORTANTE

El pin IOREF se utiliza para indicar a placas supletorias a qué tensión opera nuestra placa.

2

```
void setup()
{
  analogReference(EXTERNAL);
}
void loop()
{
}
```

Conector ICSP

IMPORTANTE

No será objeto de este libro el como realizar todo el proceso de instalación del bootloader pero disponemos de información para entender como realizar este proceso

https://www.arduino.cc/en/Hacking/Bootloader

En cuanto al hardware de Arduino quedan una serie de pines de los cuales aún no hemos visto, no son otros más que los pines del conector ICSP.

Para entender su utilidad debemos recordar que el hardware de Arduino es una plataforma que nos aporta una serie de soluciones a la hora de trabajar con un MCU, cuando compramos un Arduino la plataforma se encuentra totalmente integrada con el MCU que incorpora el hardware.

El Arduino UNO se diferencia del resto de Arduinos en que es posible reemplazar el MCU que tiene integrado. Esto nos permite programar diferentes MCU con el mismo hardware sin tener que comprar más Arduinos, con el ahorro económico que eso conlleva.

1. Cuando adquirimos un MCU por separado tenemos que tener presente que no vendrá con lo que se conoce como el gestor de arranque o bootloader cargado.

2. Esto supone que a pesar de poder reemplazar el MCU original que trae el hardware, la plataforma Arduino no funcionará con este nuevo MCU no siendo posible cargar nuestros programas a través del conector USB.

3. ISP hace referencia a una forma en la que podemos programar un MCU sin el bootloader cargado.

4. Entonces a través del conector ICSP o bien podemos cagar un programa a un MCU que no tenga el bootloader instalado o cargarle el gestor de arranque para que, a partir de ese momento pueda ser reprogramado.

5. La forma en la que se instala el gestor de arranque en el Arduino es precisamente a través de este conector ICSP.

Las placas Arduino son idóneas para la elaboración de prototipos, en el caso que nuestra intención sea pasar del prototipo al producto final será nesario diseñar un hardware que implemente todos los sistemas integrados en el prototipo.

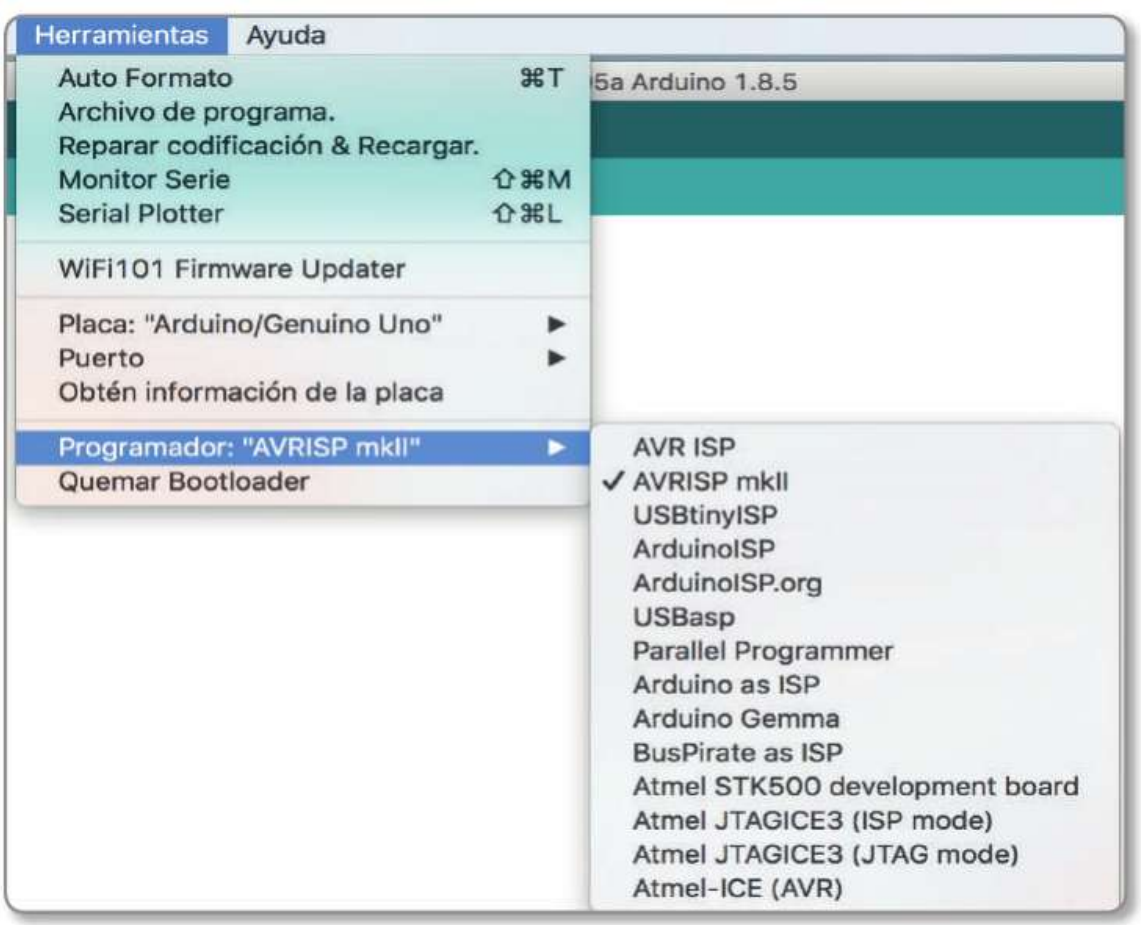
Una vez con todo integrado en una PCB, tenemos que cargar el programa que hemos desarrollado, en este punto necesitaremos dejar una serie de pines accesibles para que através de un programador ICSP poder cargar el programa.

082

Cuando cargamos nuestros programas a través de la comunicación USB estamos utilizando un protocolo de comunicaciones, este protocolo es diferente a si utilizamos el programador ISP.

1. Existen diferentes programadores ISP : AVRISP mkll, USBtinyISP, USBasp, Arduino...
2. Debemos indicar en el IDE que programador estamos utlizando ❶.

❶



Librerías IDE Arduino

Ya hemos utilizado librerías anteriormente, con un librería podemos ampliar las funcionalidades de un Arduino e incluso facilitan la programación a la hora de trabajar con módulos (GSM,ETHERNET,WIFI...).

Existe una gran cantidad de librerías, así que revisaremos las librerías más comunes.

- EEPROM: Permite trabajar con la memoria EEPROM del Arduino, es una memoría que almacena los datos aún después de apagarse el Arduino. Con lo que si los datos a almacenar "cogen" en la capacidad de esta memoria no tendremos que utilizar una memoria SD.
- ETHERNET: Incorpora todas las funciones necesarias para poder conectarnos a internet mediante un módulo ETHERNET, configurando el Arduino como un servidor o un cliente.
- FIRMADATA: Se utiliza cuando tenemos una aplicación instalada en el PC cuya función es el control o monitorización de los pines del Arduino.
- GSM: Desarrollada para utilizar módulos GSM/GPRS con los cuales podemos conectar nuestro Arduino a una red móvil y conectarnos a internet o simplemente el envío y recepción de mensajes y llamadas.
- LiquidCrystal: Dispone de las funciones necesarias para controlar una pantalla LCD, muchos de estos módulos integran un hardware que permite la comunicación con el protocolo I2C con lo que nos ahorramos una gran cantidad de conexiones, pero en este caso la librería sería LiquidCrystal_I2C.
- SD: Esta libería nos permite almacenar y leer datos en una memoria SD. Incluyendo todas las funciones para crear ,buscar eliminar archivos y carpetas con diferentes extensiones.
- SERVO: Con ella se pueden controlar servomotores
- SPI: Controla el protocolo de comunicaciones SPI, por o general no será necesario utilizarla ya que dispositivos que utilizan este protocolo como ETHERNET ,SD.. ya disponen de sus propias librerías con todas las funciones necesarias, pero si es interesante conocer las funciones que utiliza y entender como funciona este protocolo por si a la hora de integrar diferentes dispositivos que lo utilicen nos encontrarnos con algún problema y así ser capaces de hacer pruebas con esta librería y encontrar el error.
- SoftwareSerial: Con esta librería ya hemos trabajado y hemos visto que nos permite crear nuevos puertos serie para ampliar el número de dispositivos con los que poder comunicarnos a través de este protocolo.
- Steeper: Diseñada para trabajar con motores paso a paso.
- TFT: Permite dibujar texto, imágenes y detectar la interacción de usuario con una pantalla TFT.
- WiFi: Incluye toda la programación que incorpora la librería ETHERNET, pero para trabajar con un módulo WIFI.
- Wire: Librería que trabaja con otro de los protocolos de comunicación presentes en el hard-

ware de Arduino como es el I2C.Una gran cantidad de sensores y modulos trabajan con este protocolo : Pantallas LCD ,IMU, RTC... es nu protocolo , maestro esclavo al igual que SPI pero con unas diferencias significativas : como son el número de líneas de comunicación, la limitación del número de esclavos.. Pero al igual que ocurría con el protocolo SPI cualquier módulo que utilice este protocolo dispone de su propia librería con lo que posiblemente no lleguemos a utilizar las funciones directamente de esta librería.

Arduino dispone de más librerías e incluso encontraremos librerías "no oficiales" al fin y al cabo nosotros mismos podemos desarrollar nuestras propias librerías.

Desde la web de Arduino nos muestran como poder crear una librería (https://www.arduino.cc/en/Hacking/LibraryTutorial) lo cual puede ser interesante a la hora de reutilizar partes de programas y así poder desarrollar programas de forma más rápida y siguiendo nuestras propias pautas de programación.

Independientemente del origen de la librería, todas ellas disponen de ejemplos para mostrarnos las funciones de las que dispone. El que un dispositivo disponga de una librería puede ser un factor que nos ayude a decantarnos por un dispositivo u otro ya que tendremos una programación ya desarrollada sin tener que hacer todo el trabajo desde cero.

Shields Arduino

IMPORTANTE

En este capítulo se hace una revisión de los principales shield oficiales de Arduino, pero existen muchos más.

- Adaptador MKR2UNO "transforma" el hardware del MKR1000 en el Arduino UNO 1.
- El Arduino Motor Shield permite conectar motores paso a paso, relés y solenoides a un Arduino 2.
- Arduino USB Host Shield ofrece un interfaz para poder conectar dispositivos USB 3.
- El Arduino 4 Relays Shield incoprota 4 salidas a relay que nos permite aumentar la potencia de salida de ciertos pines del Arduino 4.
- El Yún Shield integra las funcionalidades del Arduino Yun a otros Arduinos como el Arduino UNO 5.
- ARDUINO WIRELESS SD SHIELD : integra una tarjeta SD y la posibilidad de conectar un módulo XBEE 6.
- Arduino Ethernet Shield 2 conecta un Arduino a Internet 7.
- El Arduino GSM Shield V2 permite conectar un Arduino a Internet mediante una red de telefonía o envío y recepción de llamas y mensajes 8.

1

2

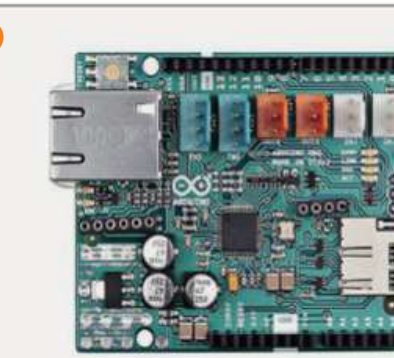
3

4

5

6

7



8

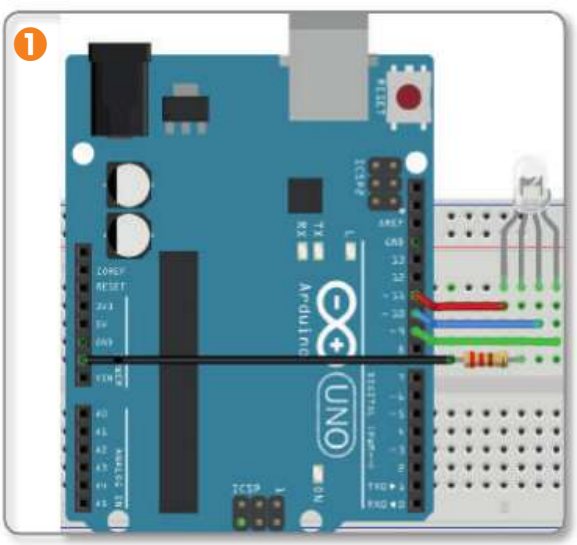
Led RGB

Cualquier color se puede conseguir con la combinación de los 3 colores básicos, pues eso es lo que nos permite hacer un Led RGB ya que no es más que la unión de tres leds de los cuales cada uno representa uno de los tres colores básicos (rojo verde azul).

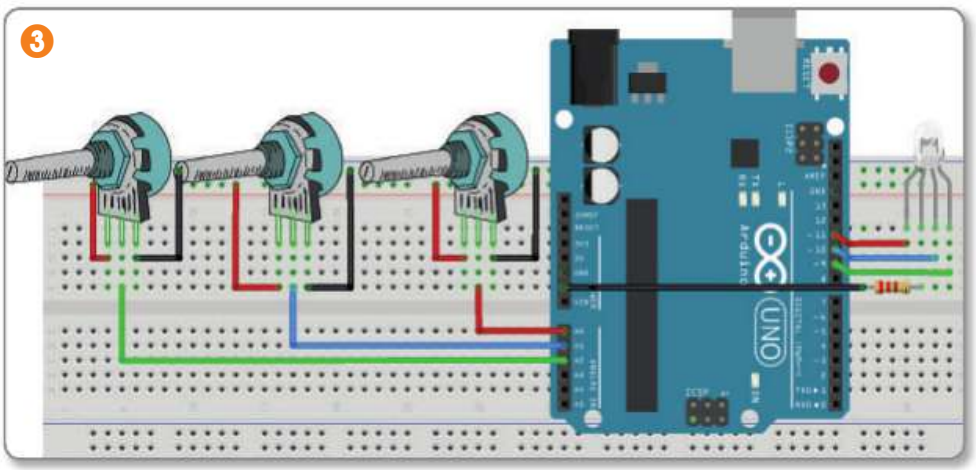
1. Estos leds disponen de 4 patillas, 3 de ellas nos permitirán indicar el grado de intensidad de cada color y el restante para cerrar el circuito de cada diodo y que así se puedan encender
2. Para controlar un led RGB necesitamos señales PWM, por lo que conectaremos cada led a una salida que pueda generar señales PWM.
3. Cada uno de los cables representa el color del led que se va ha controlar ❶.
4. Con el siguiente programa ❷ se consigue visualizar cada segundo uno de los tres colores básicos.
5. El uso más común de un led RGB es variar el color del led así con un único piloto podemos representar diferentes estados de un proceso.
6. Para probar esto conectamos 3 potenciómetros ❸, cada uno de ellos regulará la cantidad de uno de los 3 colores que debe haber en la composición.
7. Con el siguiente programa ❹ podemos regular con cada uno de los 3 potenciómetros la cantidad de cada uno de los tres colores que debe haber en la combinación.
8. Para simplificar el código, se programa una función que recoge la entrada sobre la cual debe calcular el porcentaje de color y devuelve el porcentaje correspondiente.

También encontramos leds RGB SMD, la única diferencia de estos es el tamaño del led, ya que el control y uso es idéntico a los anteriores.

❶



3



2

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

void loop()
{
  analogWrite(11, 255);
  analogWrite(10, 0);
  analogWrite(9, 0);
  delay(1000);
  analogWrite(11, 0);
  analogWrite(10, 255);
  analogWrite(9, 0);
  delay(1000);
  analogWrite(11, 0);
  analogWrite(10, 0);
  analogWrite(9, 255);
  delay(1000);
}
```

4

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

void loop()
{
  analogWrite(11, color(0));
  analogWrite(10, color(1));
  analogWrite(9, color(2));
}

byte color(byte A) {
  return map(analogRead(A), 0, 1023, 0, 255);
}
```

LDR

Un LDR es un fotoresistor, es decir, es un resistencia que varía su valor en función de la luz a la que se encuentre expuesta. Cuanta más incida sobre la resistencia su valor irá decreciendo y a medida que incida menos cantidad de luz su valor irá incrementándose.

El inconveniente de estos sensores es que debemos interpretar su hoja de características, pues su salida no es lineal . Aún así decir que incluso dentro de una misma familia de fabricantes de LDR de unos a otros suelen existir pequeñas variaciones.

1. Conectamos una resistencia 10k a la salida del LDR(ete no tiene polaridad por ser una resistencia por lo que no importaría a que patilla conectamos la resistencia), con esto conseguimos crear un divisor de tensión ❶.

2. En este primer caso simplemente encenderemos un led cuando la lectura del LDR nos indique una cantidad de luz baja y lo se apagará cuando haya un nivel alto de intensidad ❷.

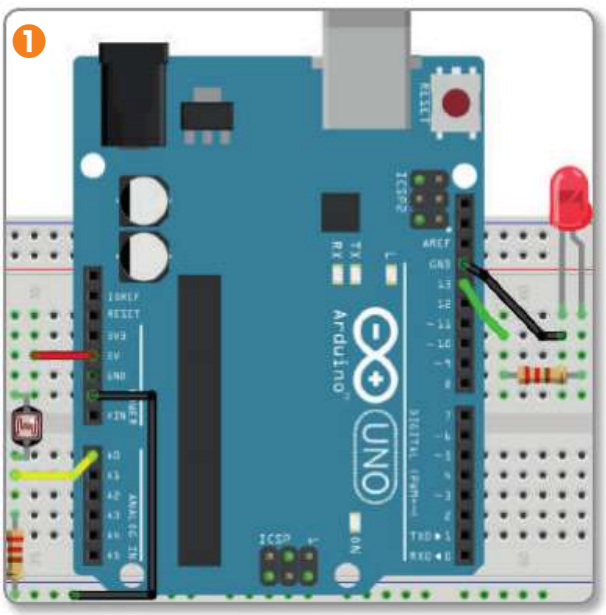
Podemos encontrar LDRs en una gran cantidad de aplicaciones, es por eso que quizás nos interese establecer diferentes actuaciones en función del rango de luz entre el que se encuentre la lectura.

Como será necesario calibrar las medidas que nos devuelva el fotoresistor, quizás el método más rápido y práctico que no nos obligará a realizar cálculos sea el de establecer esos rangos de forma experimental, es decir tomar las lecturas del sensor y comprobar sobre el terreno cuales son los diferentes rangos con los que queremos trabajar.

1. Para ello lo primero será simplemente visualizar los datos que devuelva el sensor por el monitor serie ❸.

2. Con las diferentes pruebas que hagamos podremos definir unos rangos entre los cuales realizaremos las diferentes tareas que nos requiera la aplicación.

1



2

```
void setup()
{
  pinMode(13, OUTPUT);
  pinMode(14, INPUT);
}

void loop()
{
  digitalWrite(13, digitalRead(14));

}
```

3

```
void setup()
{
  Serial.begin(9600);
}

void loop()
{
  Serial.println(analogRead(A0));

}
```

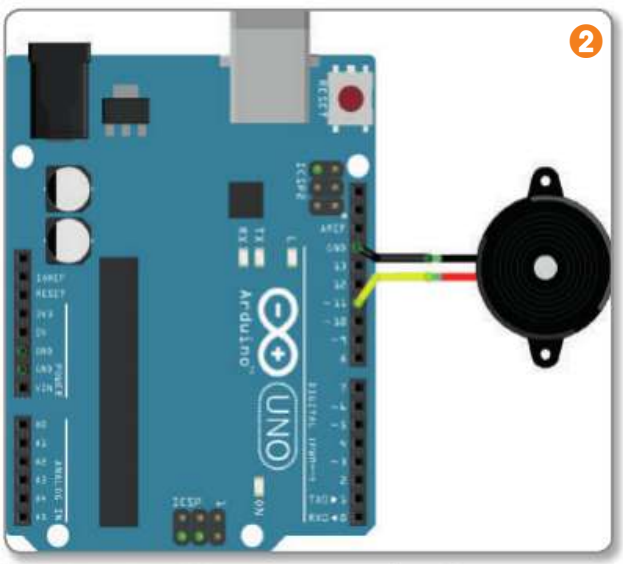
```
545
545
545
```

Buzzers

Un buzzer o zumbador ❶ es un dispositivo piezoeléctrico que como tal al estar sometido a tensión es capaz de vibrar y emitir sonido.

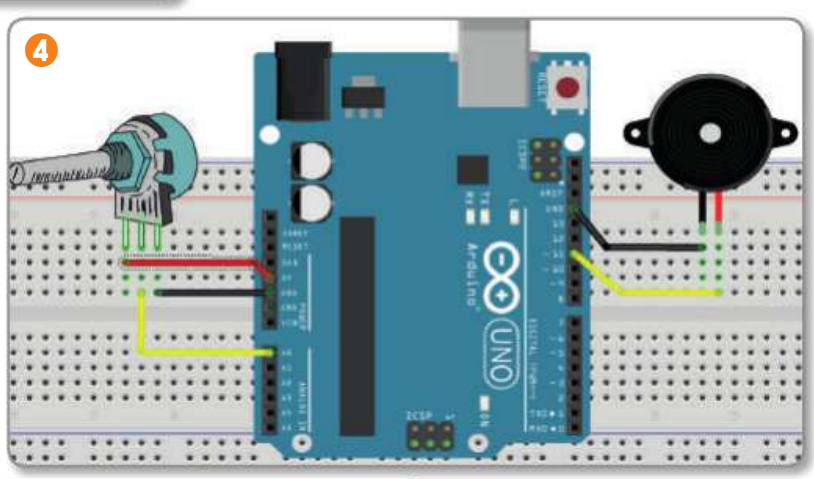
En lo que se refiere a los buzzers suele haber cierta confusión ya que existen buzzers activos y pasivos y físicamente son similares.

- Un buzzer activo es capaz de generar sonido a una determinada frecuencia (un sonido en concreto) al someterse a tensión. Esto es así por que disponen de un oscilador interno que emite señales periódicas al alimentar el buzzer, como esa señal no varía, el sonido que emiten siempre es el mismo.
- En contraposición los pasivos no disponen de un oscilador interno con lo que si variamos la frecuencia de tensión de alimentación conseguiremos variar el sonido que emite el buzzer. Son algo así como una especie de altavoces.
- Entonces utilizaremos un buzzer activo cuando simplemente queremos hacer un sonido constante como puede ser el de una alarma y un pasivo cuando queremos que suene una melodía.
- Realmente podemos conseguir el mismo resultado con un pasivo que con un activo, ya que si alimentamos siempre a la misma frecuencia el buzzer emitirá siempre el mimo sonido por lo que nos centraremos en este para el caso práctico.
- Un buzzer pasivo dispone de 2 patillas y su uso es similar a la de un diodo, por una de ellas es por donde entra la intensidad y la otra por donde cerramos el circuito.
- Con el siguiente conexionado ❷ conseguiremos trabajar con el buzzer pasivo tanto como activo como pasivo.
- Con el siguiente programa ❸ emitiremos el mismo pitido cada 3 segundos.
- Al estar emitiendo continuamente el mismo sonido se estaría comportando de forma similar a un buzzer activo.
- En el siguiente caso trataremos de variar el sonido en función de un potenciómetro ❹.
- Con este simple programa ❺ variamos la frecuencia de la señal PWM en función de la lectura del potenciómetro, quizás no detectemos un cambio significativo del sonido, esto es por que en realidad no estamos cambiando la frecuencia de la señal sino su ciclo de trabajo. Realmente lo que hacemos es cambiar el volumen del sonido



3

```
void setup()
{
  pinMode(11, OUTPUT);
}

void loop()
{
  analogWrite(11, 127);
  delay(3000);
  analogWrite(11, 0);
  delay(3000);
}
```



5

```
void setup()
{
  pinMode(11, OUTPUT);
}

void loop()
{
  analogWrite(11, map(analogRead(0),0,1023,0,255));

}
```

Tone() noTone()

Como hemos comprobado en el capítulo anterior cambiando el ciclo de trabajo de una señal PWM lo único que conseguimos es variar el volumen del sonido emitido, pero no cambia el tono del sonido.

El lenguaje de programación de Arduino dispone de funciones propias para emitir sonido a través de zumbadores o altavoces conectados a nuestro Arduino.

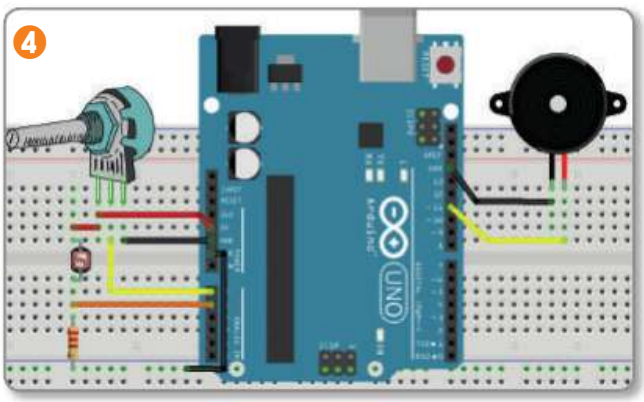
- tone(); Esta función acepta dos o tres parámetros y permite cambiar la frecuencia de una señal pero siempre con el mismo ciclo de trabajo que es del 50% con lo que generemos el sonido que generemos siempre tendrá el mismo volumen.
- tone(pin,frecuencia); En caso de utilizar únicamente dos parámetros podemos configurar sobre que pin estamos generando este señal periódica y como segundo parámetro la frecuencia de dicha señal (sin poder variar el volumen, es decir, un ciclo de trabajo del 50%) .
- noTone(); En caso de estar generándose una determinada señal proveniente de la instrucción tone(), la instrucción noTone() haría que se dejase de generar esa señal .
- Como la función Tone() utiliza el Timer2 del Arduino, en caso de utilizar el Arduino UNO no conviene utilizar las salidas PWM 3 y 11 puesto que podría dar lugar a errores.
- Con la función tone() simplificada podemos comprobar que el sonido no cesaría ❶.
- Pero con la instrucción noTone(), podemos hacer que cese el sonido ❷.
- La función tone() compleja (3 parámetros), nos permite indicar la duración de un tono, el tiempo se indicaría en milisegundos.
- En el siguiente caso volveremos a emitir el mismo tono y ya no necesitaremos la función no-Tone(); para indicar que cese ese sonido a los dos segundos ❸.
- Estas son las dos funciones para poder crear tonos y melodías con un Arduino, navegando por internet encontraremos infinidad de casos de como crear nuestras propias.
- Con un LDR y un buzzer podemos crear nuestro propio instrumente theremin fue uno de los primeros instrumentos musicales que se controlaban sin la ser manipulados físicamente sino a través de gestos.
- Montaremos un circuito ❹ con un buzzer y un LDR que nos permita variar el tono en función de la luz
- Al igual que el dispositivo original modularemos el sonido entre 20Hz Y 20kHz ❺.

1
```
void setup()
{
  pinMode(11, OUTPUT);
  tone(11, 1000);
}

void loop() {}
```

2
```
void setup()
{
  pinMode(11, OUTPUT);
  tone(11, 1000);
  delay(2000);
  noTone(11);
}

void loop() {}
```

3
```
void setup()
{
  pinMode(11, OUTPUT);
  tone(11, 1000,2000);
}

void loop() {}
```

4



5
```
void setup() {
  pinMode(11, OUTPUT);

}
void loop()
{
  int frecuencia = map(analogRead(0), 0, 1023, 20, 20000);
  tone(11, frecuencia);
}
```

Sensor temperatura DS18B20

El sensor DS18B20 es un sensor óptimo para medir temperaturas y muy fiable e incluso puede trabajar en ambientes húmedos s.

Encontraremos este sensor en diferentes formatos incluyendo una sonda impermeabilizada con la que podemos medir la temperatura de líquidos.

- Sea cual sea el formato del sensor dispondrá de 3 patillas, dos de ellas par alimentación y otra para la salida de datos.
- La alimentación del sensor debe estar comprendida entre 3 y 5,5V.
- Puede medir un rango de temperatura de -55ºC a 125ºC pero no en todo el rango de medida tendremos la misma resolución.
- De -10ºC a 85ºC su resolución es de +_ 0,5ºC y fuera de estés rango su precisión es de 0,2ºC.
- Será necesario conectar a la salida de datos una resistencia de un valor de 4,7K(si la longitud del cable comprende entre 0 y 5 m) . La salida de la resistencia se conecta a un pin digital y por a su vez a 5V❶.
- La datos que devuelve este sensor se transmiten mediante el protocolo 1-Wire.
- El protocolo 1-Wire fue diseñado por Dallas Semiconductor y es un bus de datos que utiliza únicamente dos líneas una de ellas para la transmisión de datos y la otra para poner todas las tierras de los dispositivos en común.
- Este protocolo permite comunicar una gran cantidad de dispositivos pero todos ellos deben ir conectados al bus mediante una resistencia pull-up, es por eso el uso de la resistencia en el circuito anterior.
- Para trabajar con este sensor será necesario utilizar varias librerías.
- Instalamos la librería OneWire ❷.
- Instalamos la librería DallasTemperature ❸.
- Con estas librerías instaladas podemos cargar el siguiente programa ❹ que nos devuelve por el monitor serie la temperatura que está midiendo el sensor.

❶


```
#include <OneWire.h>
#include <DallasTemperature.h>
//Definimos el pin al que se conecta el sensor
OneWire pin_Arduino(5);
//Definimos el sensor con el que trabajamos
DallasTemperature DS18B20(&pin_Arduino);
void setup() {
  Serial.begin(9600);
  DS18B20.begin();
}
void loop()
{
  DS18B20.requestTemperatures();
  Serial.print("Temperatura  ");
  Serial.print(DS18B20.getTempCByIndex(0));
  Serial.println(" °C");
  delay(1000);
}
```

4

```
Temperatura  18.44 °C
Temperatura  18.44 °C
Temperatura  18.44 °C
Temperatura  18.44 °C
Temperatura  19.87 °C
Temperatura  20.50 °C
Temperatura  20.12 °C
```

Sensor humedad y temperatura

En el capítulo anterior conseguimos medir temperaturas con un Arduino, pero para una gran cantidad de aplicaciones también será necesario medir la humedad.

Los sensores DHT11 y DHT22 pertenecen a la misma familia de sensores que son capaces de medir temperatura y humedad.

1. Características DHT11:

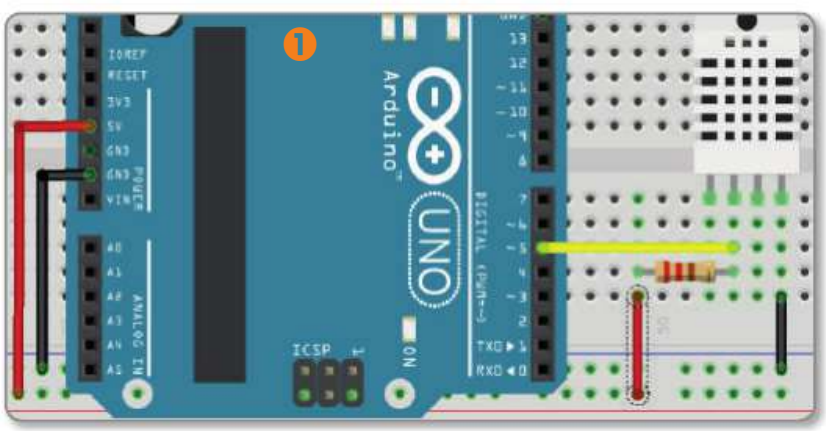
- Rango de temperatura 0:50 ºC precisión de 2ºC
- Rango de Humedad 20:80% precisión de 5%
- Frecuencia de muestreo 1Hz

2. Características DHT22

- Rango de temperatura -40:125 ºC precisión de 0,5ºC
- Rango de Humedad 20:80% precisión de 5%
- Frecuencia de muestreo 2 Hz

3. Características comunes

- Tensión de alimentación de 3,3 a 5V.
- 4 patillas dos para alimentar una sin uso y la última para la salida de datos.
- Al igual que ocurría con el sensor de temperatura del caso anterior es necesario conectar una resistencia de pull-up en la línea de datos del sensor ❶.
- La resistencia tendrá un valor de 10k
- Necesitamos la librería Adafruit_Sensor ❷.
- Y la librería DHT ❸.
- Con los pasos anteriores realizados, con el siguiente programa ❹ visualizaremos la temperatura y humedad del sensor DHT11, en caso de utilizar el sensor DHT22 simplemente se indicaría en la línea <<DHT sensor(5, DHT11); >> por DHT sensor(5, DHT22);



4

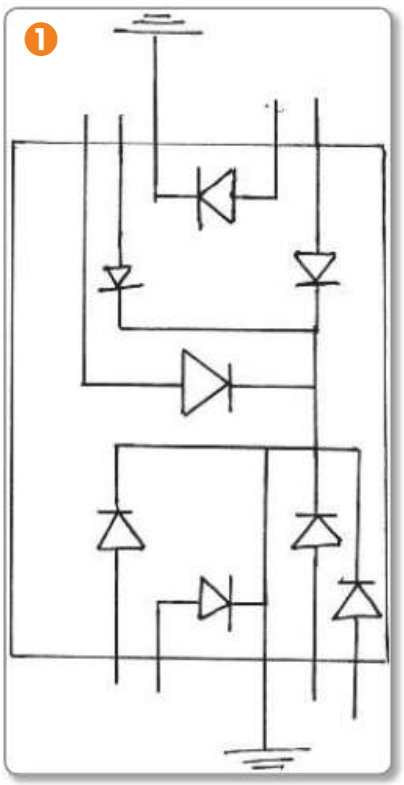
```
#include <Adafruit_Sensor.h>
#include <DHT.h>
DHT sensor(5, DHT11);
void setup() {
  Serial.begin(9600); sensor.begin();
}

void loop() {
  float humedad = sensor.readHumidity();
  float temperatura = sensor.readTemperature();

  if (isnan(humedad) || isnan(temperatura) )
  {
    Serial.println("Error obteniendo los datos ");
  } else {
    Serial.print("Humedad relativa ");
    Serial.print(humedad);
    Serial.print("   Temperatura   ");
    Serial.println(temperatura);
  }
  delay(1000);
}
```

Display 7 segmentos

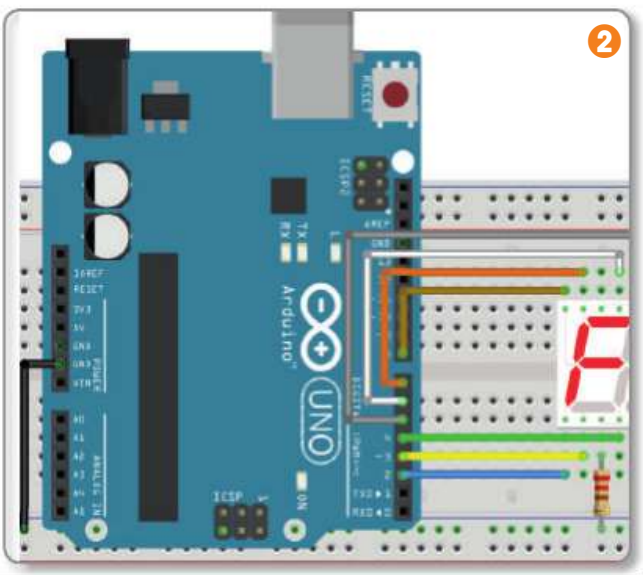
A medida que nuestros proyectos empiecen a crecer necesitaremos mostrar información relativa a un proceso, un display de 7 segmentos permite mostrar dígitos del 0 al 9.

- Se denomina display de 7 segmentos puesto que utilizan 7 leds para representar cualquier dígito.
- Los displays los podemos comprar con ánodo o cátodo común en nuestro caso utilizaremos un display de cátodo común(5161AS) ❶.
- En este caso debemos conectar una de las dos patillas del display de cátodo común a tierra a través de una resistencia cuyo valor será de 330 ohmios para respetar la caída de tensión en cada diodo ❷.
- Por último el display dispone de un led en forma de punto que en nuestro caso no utilizaremos.
- Para mostrar un digito determinado se deben encender la combinación de leds correspondientes, para ello conectaremos cada uno de ellos a una salida del Arduino y uno de los puntos de cátodo común a GND mediante una resistencia.
- Es importante conocer que segmento del display está conectado a cada salida, para saber cuales se deben encender en según que dígito.
- Con el siguiente programa ❸ se irá incrementado el valor del dígito mostrado cada dos segundos hasta llegar a 9, tras lo cual se volvería a repetir el proceso.
- En la matriz <<digito>> se guarda por orden el valor que debe tener cada salida para representar cada dígito.
- En el void loop se recorre cada dos segundos una fila distinta actualizando el valor de las salidas según el valor definido en la columna correspondiente.
- En el caso de haber utilizado un display de ánodo común el proceso sería inverso, es decir, en vez de conectar el ánodo a GND a través de una resistencia se conectaría a 5V y se invertirían todos los valores de la matriz, puesto que en este caso en vez de activar la salida para que se encienda un led debemos ponerla a nivel bajo.

1



2



3

```
boolean digito[10][7] =
{ //2, 3, 4, 5, 6, 7, 8 --> pin
  {1, 1, 1, 1, 1, 1, 0}, //0 -->digito
  {0, 0, 1, 1, 0, 0, 0}, //1 -->digito
  {1, 1, 0, 1, 1, 0, 1}, //2 -->digito
  {0, 1, 1, 1, 1, 0, 1}, //3 -->digito
  {0, 0, 1, 1, 0, 1, 1}, //4 -->digito
  {0, 1, 1, 0, 1, 1, 1}, //5 -->digito
  {1, 1, 1, 0, 0, 1, 1}, //6 -->digito
  {0, 0, 1, 1, 1, 0, 0}, //7 -->digito
  {1, 1, 1, 1, 1, 1, 1}, //8 -->digito
  {0, 1, 1, 1, 1, 1, 1}, //9 -->digito
};
void setup() {
  for (byte i = 2; i < 9; i++)
  {
    pinMode(i, OUTPUT);
  }
}
void loop() {
  for (byte numero = 0; numero < 10; numero++)
  {
    for (byte salida = 2, pin = 0 ; pin < 7; salida++, pin++) {
      digitalWrite(salida, digito[numero][pin]);
    }
    delay(2000);
  }
}
```

Joystick

Los módulos más comunes que nos encontramos en el mercado disponen de cinco pines ❶:

- 5v: alimentación, X: Desplazamiento joystick en el eje X, Y: Desplazamiento joystick en el eje Y ,Button: estado botón, GND: alimentación.

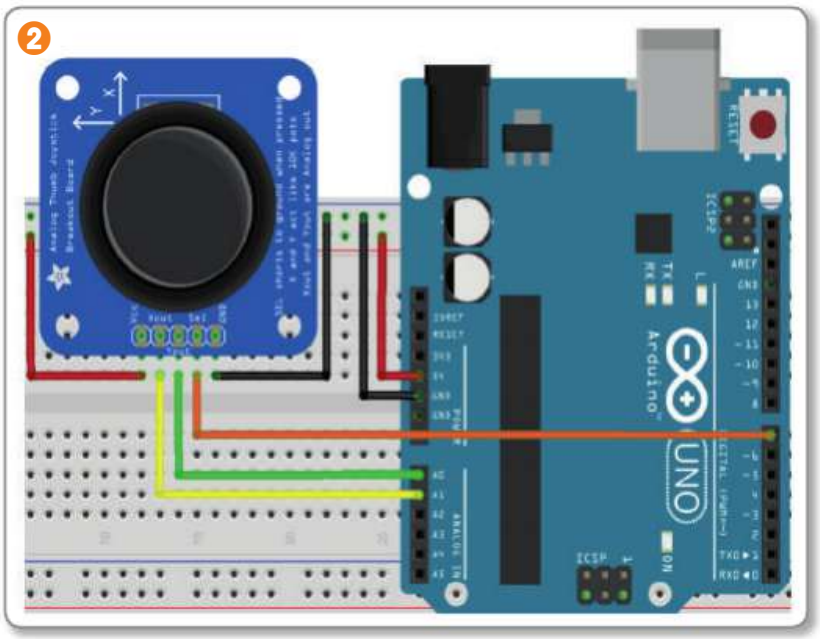
Internamente, un joystick analógico no está formado por más que dos potenciómetros asociados al desplazamiento de sus dos ejes y a mayores dispone de un pulsador, estos tres componentes comparten alimentación.

Al adquirir un joystick debemos tener claro cual es la orientación de sus ejes, con los que he trabajo la coordenada (0,0) se encontraba en la esquina superior izquierda orientando el joystick tal y como se muestra en la imagen.

Por comodidad y de cara a posibles aplicaciones rotaremos el joystick -90º para así simplificar la programación.

- Si montamos el siguiente esquema ❷ con el joystick orientado tal y como se muestra en la imagen , el sentido de los ejes cambia y la salida de eje Y ahora se corresponde con el eje X y viceversa.
- Con el siguiente programa iremos ❸ muestreando el valor de cada uno de los ejes del joystck con ayuda del monitor serie.
- Como se podrá apreciar si movemos el joystick de izquierda a derecha el valor de la entrada analógica correspondiente oscila de 0 a 1023 y si movemos de abajo a arriba ocurriría lo mismo.

2



3

```
void setup()
{
  Serial.begin(9600);
}

void loop()
{
  Serial.print("EJE Y ");
  Serial.print(analogRead(0));
  Serial.print(" EJE X ");
  Serial.println(analogRead(1));
  delay(200);
}
```

```
EJE Y 519 EJE X 504
EJE Y 519 EJE X 505
EJE Y 519 EJE X 505
EJE Y 520 EJE X 505
EJE Y 519 EJE X 505
EJE Y 519 EJE X 506
EJE Y 519 EJE X 505
EJE Y 519 EJE X 505
EJE Y 519 EJE X 505
```

PIR :Sensor de movimiento

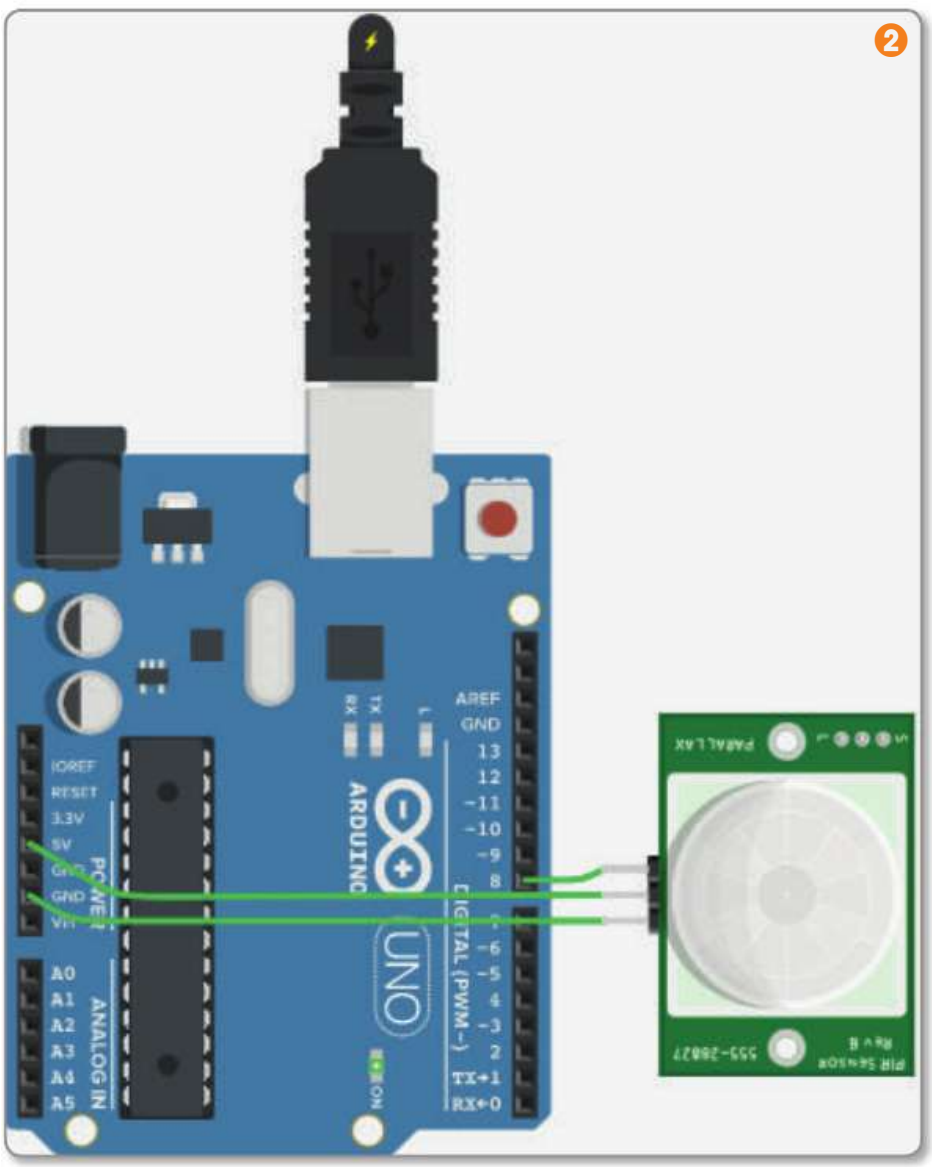
Un PIR es un sensor que es capaz de detectar movimiento gracias a la tecnología de radiación infrarroja.

Estos sensores son ideales para una gran cantidad de aplicaciones, desde sistemas de seguridad hasta el encendido de luces por presencia, además son económicos y fiables.

- Todos los cuerpos emiten radiación infrarroja ,la cantidad de radiación también dependerá de la temperatura del cuerpo . Cuando adquirimos un sensor de movimiento siempre nos indican el rango de medición de este, lo que hace este sensor es detectar un cambio de radiación con lo que emiten una señal eléctrica y gracias a un segundo dispositivo que reciba esa señal por ejemplo un Arduino podemos activar una alarma o encender una luz
- Por lo general estos sensores aparte del hardware necesario incorporan una cúpula que no es más que una lente que a groso modo ayuda a detectar un cambio de radiación dentro del campo de visión.
- Su implementación es sencilla ya que solo disponen de tres pines ❷, dos para alimentar y el restante para indicar cuando se detecta movimiento
- En caso de detectar movimiento emite una señal a nivel alto.
- La aplicación más sencilla que podemos hacer es encender un led si se detecta movimiento, en este caso ya encenderíamos el led que está asociado al pin 13 en el hardware del Arduino UNO ❶.

❶

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

void loop()
{
  digitalWrite(13, digitalRead(12));
}
```


2
AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
DIGITAL (PWM~)
ARDUINO
UNO
ON
TX
RX
L
IOREF
RESET
3.3V
5V
GND
Vin
POWER
A0
A1
A2
A3
A4
A5
ANALOG IN

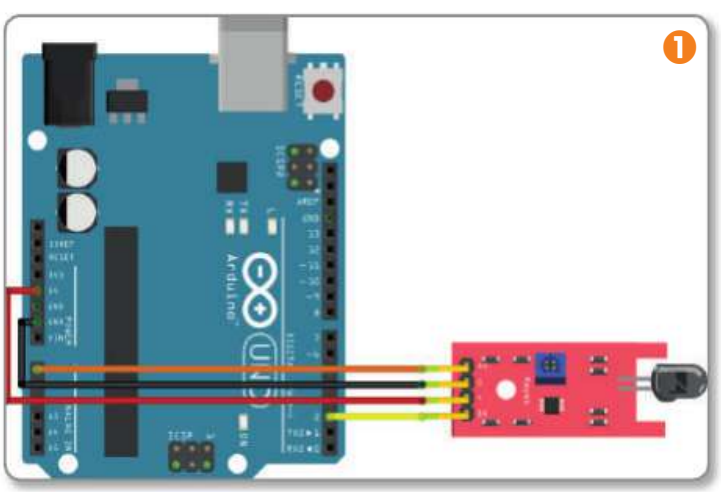
Sensor de llama

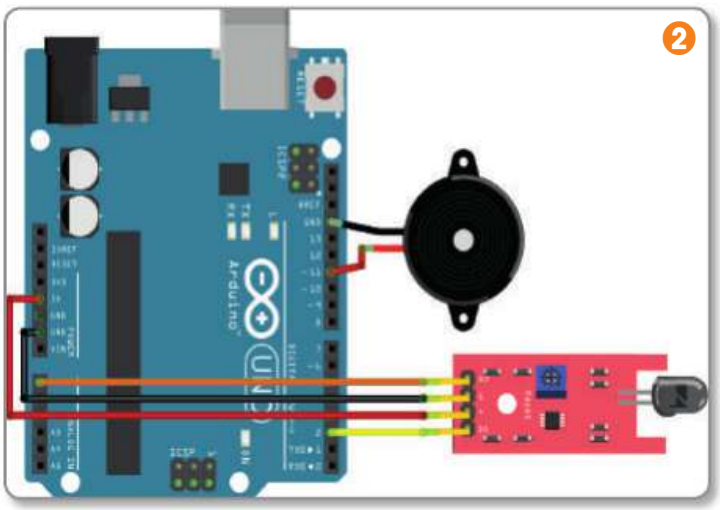
Para entender el funcionamiento de este sensor se creará una alarma contraincendios, que constará de un sensor de llama (KY-026) y un buzzer que sonará al detectar fuego.

Los sensores de llama no solo los encontraremos en instalaciones contra incendio sino en una gran cantidad de dispositivos donde sea necesario detectar la presenciar llama, por ejemplo: hornos, calderas..

El sensor que utilizamos en este caso KY-026 consta de 4 patillas dos para alimentación y otras dos que aportan una señal digital y otra analógica.

1. Calibrar el sensor ❶.
2. Conectamos el sensor al Arduino y ayudándonos del potenciómetro que integra este hardware y del monitor serie calibramos el sensor, con el potenciómetro calibramos la sensibilidad del sensor, es decir calibrar un umbral para que una vez que el sensor detecte una magnitud superior a ese umbral ponga a nivel alto la salida ❷.
3. Con el siguiente programa ❸ muestreamos para que valores el sensor detecta llama es trabajo de nosotros establecer mediante el potenciómetro el umbral con el que el sensor detecta llama.
4. Una vez calibrado el sensor conectamos un buzzer para que suene una alarma cuando el sensor detecte una llama.
5. Con este simple programa ❹ cada vez que el sensor detecte una llama el zumbador empezará a pitar.





3

```
void setup()
{
  pinMode(3, INPUT);
  Serial.begin(9600);
}

void loop()
{
  if (digitalRead(3)) {
    Serial.print("LLAMA ");
  }
  Serial.println(analogRead(0));
  delay(200);
}
```

```
909
910
909
908
LLAMA 56
LLAMA 399
LLAMA 425
LLAMA 374
LLAMA 385
LLAMA 359
LLAMA 369
LLAMA 320
```

4

```
void setup()
{
  pinMode(3, INPUT);
  pinMode(11, OUTPUT);
}

void loop()
{
  if (digitalRead(3))
  {
    analogWrite(11, 127);
    delay(100);
    analogWrite(11, 0);
    delay(100);
  }

}
```

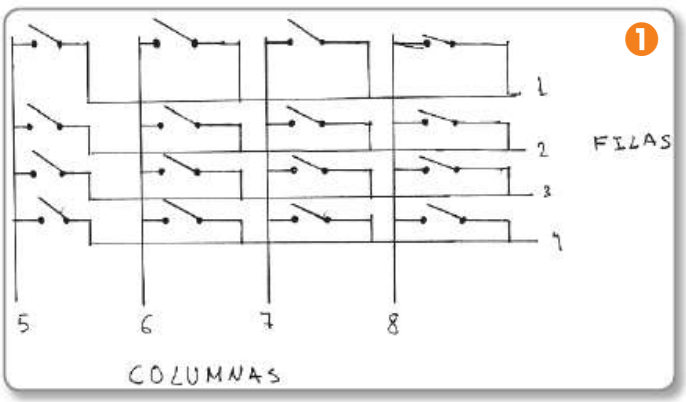
Teclado matricial

Un teclado matricial es un dispositivo que está formado por un conjunto de pulsadores que como iremos descubriendo su nombre tiene relación en como se encuentran conectados los pulsadores.

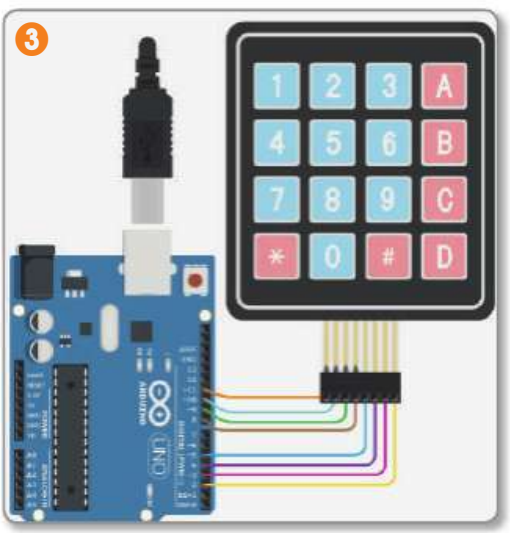
- Los pulsadores de un teclado matricial se disponen en ❶ filas y columnas, este tipo de conexionado permite reducir el número de entradas necesarias para saber que pulsador se encuentra pulsado, si no estuvieran en esta disposición harían falta 16 entradas y como veremos solo utilizaremos 8.
- El funcionamiento de este tipo de teclado es sencillo de entender si pensamos en el como si fuese una matriz de números. Cada pulsador tiene un "índice" único al resto de pulsadores de tal modo que si por ejemplo pulsamos la letra D tendremos un pulso en la columna y fila 4.
- La programación de este teclado puede resultar un tanto compleja pero disponemos de una librería que nos facilita la programación ❷.
- De no disponer de la librería el caso se complicaría teniendo que hacer diferentes operaciones pero en resumidas cuentas sería necesario muestrear cada fila y columna para saber el "índice" del pulsador sobre el que se haya pulsado. Lo cual resultaría de una operación un tanto compleja pues tendríamos que ir cambiando la configuración de los pines conectados al teclado entre salidas y entradas pullup, pero como disponemos de una librería nos centraremos en la solución con ella ❸.
- ❹ Lo primero que tenemos que tener en cuenta es que existen teclados de diferentes dimensiones y podemos conectar sus patillas a cualquier pin del Arduino, por lo que esta librería dispone de una instancia que permite configurar todos estas variantes.
- La instancia Keypad permite configurar la programación para un teclado, lo primero es indicar con una matriz que dígito o símbolo se encuentra en cada pulsador, a continuación se indica por orden donde a que pines están conectadas las filas, después las columnas y para finalizar se indican las dimensiones del teclado.
- Por último la función getKey() devuelve el símbolo o dígito del pulsador que se hubiese pulsado, si no existiera tal pulsación devolvería el valor de 0(que en ASCII significaría nulo).

095

1



3



4

```
#include <Keypad.h>
byte Input_Filas[] = {11, 10, 9, 8};
byte Input_Columnas[] = { 5, 4, 3, 2};
char teclas [ 4 ][ 4 ] =
{
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};
Keypad Teclado_matricial = Keypad(makeKeymap(teclas), Input_Filas, Input_Columnas, 4, 4);

void setup()
{
  Serial.begin(9600) ;
}
void loop()
{ char pulsado = Teclado_matricial.getKey() ;
  if (pulsado != 0)
  {
    Serial.print("Se ha pulsado la tecla: ");
    Serial.println(pulsado);
  }
}
```

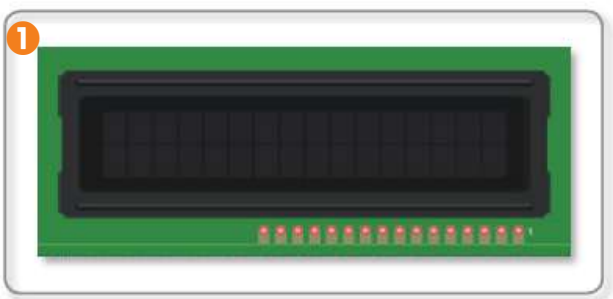
Pantalla LCD (I)

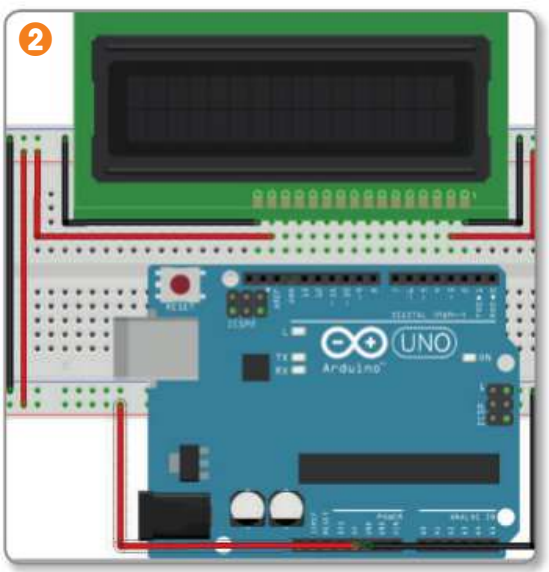
Una pantalla LCD (acrónimo de Liquid Crystal Display lo que en español sería Pantalla de Cristal Líquido) es un dispositivo que nos permite mostrar información en formato texto.

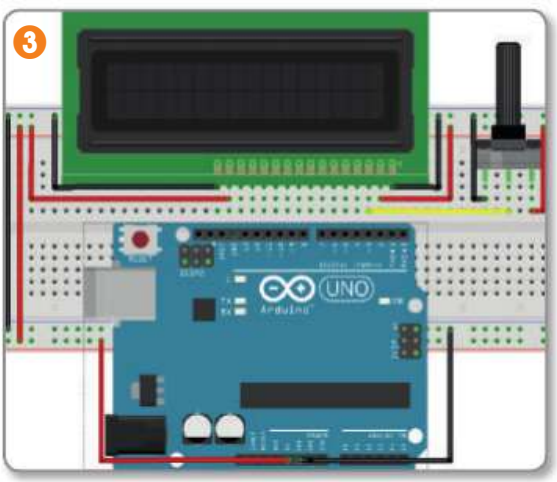
- En el mercado disponemos de pantallas LCD de diferentes dimensiones,la dimensión de una pantalla define la "cantidad" de información que puede mostrar o lo que es lo mismo el número de caracteres que puede mostrar.
- En nuestro caso la pantalla LCD tendrá una dimensión de 16x2 ❶, lo que significa que tiene dos líneas y en cada una puede mostrar 16 caracteres lo que hace un total de 32 caracteres que puede mostrar este modelo.
- Para trabajar con una pantalla LCD es tan importante la programación como el conexionado, con lo que en este capítulo aprenderemos a conectarla a un Arduino y en el siguiente a programarla.
- Lo primero es conectar el apartado de potencia ❷.
- A continuación conectamos un potenciómetro que permitirá regular el brillo de la pantalla ❸.
- Con el potenciómetro conectamos, seguiremos por los pines de control ❹.
- Anotaremos donde conectamos cada pin de control a nuestro Arduino ya que será necesario configurar estos pines por programación. RS(4),E(6),D4(10),D5(11),D6(12) y D7(13).
- Por último se conecta el pin RW a tierra ❺.

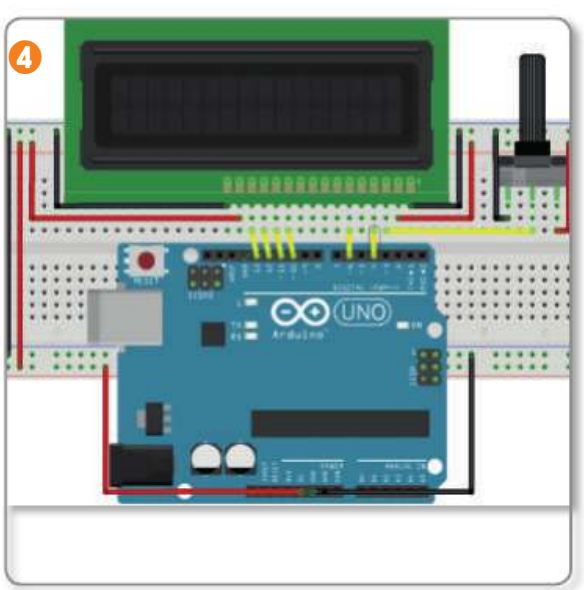
Todo este conexionado puede parecer un tanto engorroso así que si buscamos encontraremos un controlador que se conecta a la pantalla y que con tan solo 4 cables podemos controlar la pantalla LCD.

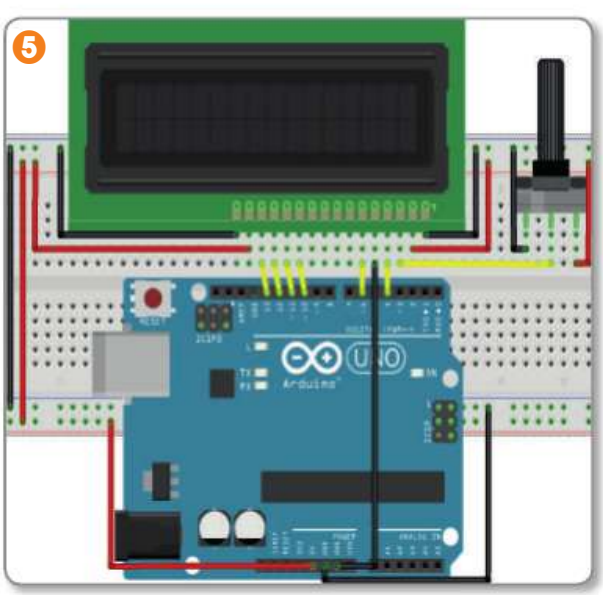
Este controlador opera con el protocolo I2C y ya que en este libro no hemos abordado este protocolo seguiremos con la pantalla LCD conectada tal y como se muestra en las imágenes, pero decir que de cara a la programación no habría grandes cambios ya que para las dos posibilidades tenemos funciones idénticas.




2


3


4


5

Pantalla LCD (II)

Para trabajar con la pantalla LCD utilizaremos la librería <<LiquidCrystal>>.

1. Configuramos la pantalla LCD ; poniéndole un nombre con el cual nos referiremos a ella a partir de este instante y por orden el conexionado de los pines de control ❶.
2. En el void setup por orden realizamos las siguientes operaciones, inicializamos la pantalla indicando sus dimensiones, colocamos el cursor (“elemento“ que indica en que “coordenada “ de la pantalla se escribe) en la posición 0,0 (margen izquierdo superior) y a continuación mostramos un mensaje ❷.
3. La función setCursor como ya se ha mencionado sitúa la posición a partir de la cual se empieza a escribir el mensaje, su primer parámetro indica la columna y el segundo la fila, de tal modo que en este caso ❸ el mensaje aparece en la misma fila pero más centrado .
4. La función clear borra el contenido que hubiese impreso en la pantalla ❹.
5. La función scrollDisplayLeft ( ) y scrollDisplayRight ( ) permiten mover el texto escrito hacia izquierda y derecha respectivamente ❺.
6. El función blink y noBlick hace parpadear el cursor de estilo bloque ❻.
7. El función cursor y noCursor hace parpadear el cursor de estilo guión.
8. Podemos hacer que el texto parpadee con la función display y noDisplay ❼.

❶

```
#include <LiquidCrystal.h>
                          //RS,E, D4 ,D5 ,D6 ,D7
LiquidCrystal pantalla_lcd(4, 6, 10, 11, 12, 13)
```

❷

```
#include <LiquidCrystal.h>
LiquidCrystal pantalla_lcd(4, 6, 10, 11, 12, 13);
void setup()
{
  pantalla_lcd.begin(16, 2);
  pantalla_lcd.setCursor(0, 0);
  pantalla_lcd.print("HOLA MUNDO");
}
void loop(){}
```

3

```
#include <LiquidCrystal.h>
LiquidCrystal pantalla_lcd(4, 6, 10, 11, 12, 13);
void setup()
{
  pantalla_lcd.begin(16, 2);
  pantalla_lcd.setCursor(3, 0);
  pantalla_lcd.print("HOLA MUNDO");
}
void loop(){}
```

4

```
#include <LiquidCrystal.h>
LiquidCrystal pantalla_lcd(4, 6, 10, 11, 12, 13);
void setup()
{
  pantalla_lcd.begin(16, 2);
  pantalla_lcd.setCursor(3, 0);
  pantalla_lcd.print("HOLA MUNDO");
  delay(2000);
  pantalla_lcd.clear();
}
void loop() {}
```

5

```
#include <LiquidCrystal.h>
LiquidCrystal pantalla_lcd(4, 6, 10, 11, 12,
void setup()
{
  pantalla_lcd.begin(16, 2);
  pantalla_lcd.setCursor(0, 0);
  pantalla_lcd.print("HOLA MUNDO");

}
void loop() {
  for (byte i = 0; i < 16; i++)
  {
    pantalla_lcd.scrollDisplayRight ( ) ;
    delay(500);
  }
  for (byte i = 0; i < 13; i++)
  {
    pantalla_lcd.scrollDisplayRight ( ) ;
```

6

```
#include <LiquidCrystal.h>
LiquidCrystal pantalla_lcd(4, 6, 10, 11, 12, 13);
void setup()
{
  pantalla_lcd.begin(16, 2);
  pantalla_lcd.setCursor(0, 0);
  pantalla_lcd.print("HOLA MUNDO");
}
void loop()
{
  pantalla_lcd.noBlink();
  delay(1000);
  pantalla_lcd.blink();
  delay(1000);
}
```

7

```
#include <LiquidCrystal.h>
LiquidCrystal pantalla_lcd(4, 6, 10, 11, 12, 13);
void setup() {
  pantalla_lcd.begin(16, 2);
  pantalla_lcd.print("Hola mundo");
}
void loop() {
  pantalla_lcd.noDisplay();
  delay(1000);
  pantalla_lcd.display();
  delay(1000);
}
```

RTC Arduino(I)

El ser capaz de contabilizar la fecha y hora será esencial en muchos de nuestros proyectos, tenemos infinidad de posibilidades para conseguir que un Arduino trabaje teniendo en cuenta la fecha y hora.

- La solución que vamos a desarrollar será la de programar el propio Arduino para que se compone como un reloj, pero también tenemos otras posibilidades como la de utilizar un reloj externo del cual ir obteniendo la hora o de módulos como GPS, ETHERNET, WIFI, GPRS...que una vez que se encuentren conectados al medio podemos obtener de ellos la fecha y hora.

- Cada solución tendrá sus pros y contras, pero a grandes rasgos quizás la solución menos fiable sea la de programar un Arduino para que se comporte como un reloj ya que el Arduino puede ser sensible a interferencias y con el paso del tiempo se irá retrasando, pero es la solución más económica de entre todas y la más sencilla de desarrollar.

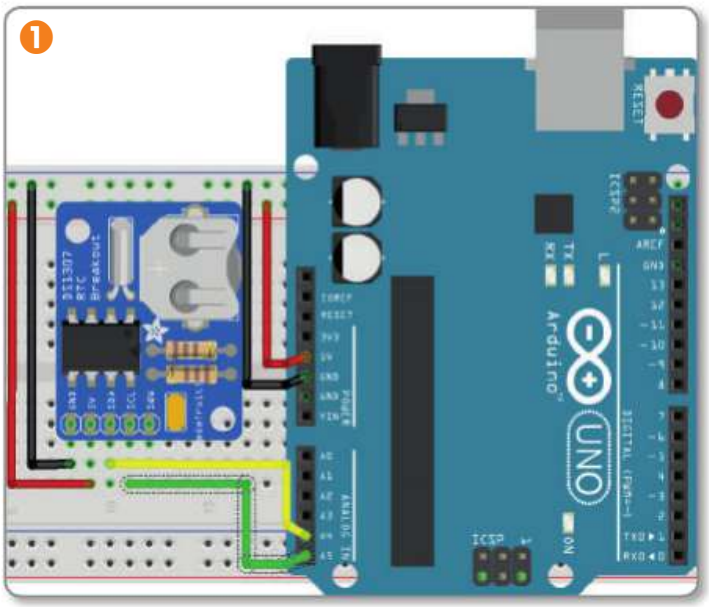
- Si por ejemplo optamos por utilizar un módulo de reloj debemos escoger el que sea más fiable ya que también los hay que no son lo todo robustos que quisiéramos un módulo que funciona bastante bien es el DS3231 ❶.

- En cuanto a obtener la hora de alguno de los módulos mencionados anteriormente tenemos la incertidumbre que pierdan la conexión con Internet o satélites con lo que no conseguiríamos obtener la hora y a parte de eso la programación es un poco más compleja.

Lo importante de lo visto en los puntos anteriores es tener presente todas las posibles soluciones y en función de las características de la aplicación escojer la más adecuada.

Para trabajar con el Arduino como si fuera un reloj necesitaremos instalar la librería TimeLib (https://github.com/PaulStoffregen/Time). En el siguiente capítulo nos adentraremos en las funciones de esta librería ❷.

1



2

Biblioteca de tiempo para Arduino http://playground.arduino.cc/code/time

RTC Arduino(II)

1. setTime configura la fecha y hora con la que se inicia el Arduino, le pasaremos por orden : hora, minuto, segundo, día, mes y año actuales❶.
2. Aquí viene uno de los principales inconvenientes de no utilizar más que el propio Arduino para gestionar la fecha y hora. En el momento que se apague volverá a reprogramarse con la fecha y hora que definimos al inicio si utilizara por ejemplo un módulo de reloj como este dispone de una pila independiente aunque se apague el Arduino el módulo no lo haría y no se reiniciaría la hora.
3. Las funciones day(), month(), year(), hour(), minute() y second() devuelven valor de la fecha y hora actuales.
4. Con el siguiente programa ❷ iremos muestreando la fecha y hora con el monitor serie.
5. Una vez que disponemos de la fecha y hora podemos crear una gran cantidad de aplicaciones entre las más comunes sería crear un datalogger que no es más que un dispositivo que registra datos junto con la fecha y hora en la que fueron tomados o programar alarmas.
6. Para trabajar con alarmas lo más cómodo es utilizar la librería TimeAlarms.Esta librería dispone de las siguientes funciones:
   - Alarm.alarmRepeat(hora, minutos, segundos, función); Función que se repite cada día a la hora indicada
   - Alarm.alarmRepeat(día_semana,hora,minutos,segundos, función); Función que se ejecuta el día de la semana el día y la hora de la semana indicados
   - Alarm.alarmOnce(hora, minutos, segundos, función); Ejecuta una única vez la función a la hora indicada.
   - Alarm.alarmOnce(día_semana,hora,minutos,segundos, función); .Igual que la anterior pero la función se ejecutaría al día siguiente.
   - Alarm.timerRepeat(segundos, función); Se ejecuta la función cada x segundos.

❶

```
#include <Time.h>
#include <TimeLib.h>
void setup()
{
        //H, Min, Se,dia,mes,año
  setTime(12, 00, 00, 1, 1, 2018);
  Serial.begin(9600);
}
```

❷

```
#include <Time.h>
#include <TimeLib.h>
void setup()
{
  setTime(12, 00, 00, 1, 1, 2018);
  Serial.begin(9600);
}
void loop()
{
  Serial.print(day());
  Serial.print("/") ;
  Serial.print(month());
  Serial.print("/") ;
  Serial.print(year());
  Serial.print(" ") ;
  Serial.print(hour());
  Serial.print(":") ;
  Serial.print(minute());
  Serial.print(":") ;
  Serial.println(second());
  delay(1000);
}
```

Fritzing

IMPORTANTE

Fritzing aporta muchas otras funcionalidades como son la generación de esquemáticos y diseño de PCB.

Fritzing es una herramienta que nos permite graficar los circuitos que realizamos mediante un entorno sencillo e intuitivo. Todas las imágenes de circuitos que se muestran en esta serie de ejemplares fueron creadas en esta herramienta.

En muchos casos, necesitamos documentar algún circuito que montemos y, si no conocemos los símbolos que se suelen utilizar en electrónica, usar programas específicos o, simplemente, necesitamos un esquema sencillo para nosotros, Fritzing es la herramienta que precisamos.

1. La podemos descargar de forma gratuita desde su página web (disponible para cualquier sistema operativo) http://fritzing.org/download/. Con el entorno descargado nos disponemos a realizar nuestro primer circuito.

2. En el menú superior accedemos al apartado «Protoboard» ❶.

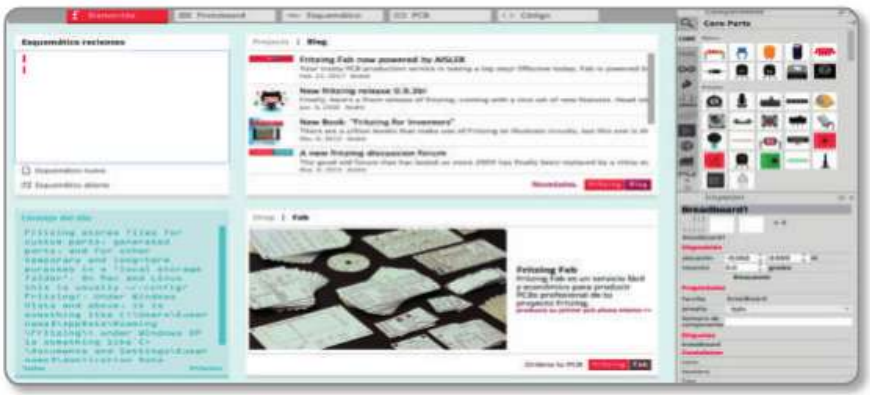
3. En la parte de lienzo ❷ es donde incorporamos los componentes y trazamos las conexiones. Para realizar una conexión con un cable ❸, basta con seleccionar cualquier punto de conexión y desplazar el ratón.

4. Esto lo podemos realizar con cualquier punto de conexión de cualquier componente que incorporemos al lienzo. Existen componentes que permiten algún tipo de configuración gráfica, por ejemplo: rotación, color, etc. Según el componente, tendremos diferentes opciones, pero todos se seleccionan de la misma forma: hacemos clic en el componente y, después, pulsamos el botón derecho de nuestro ratón.

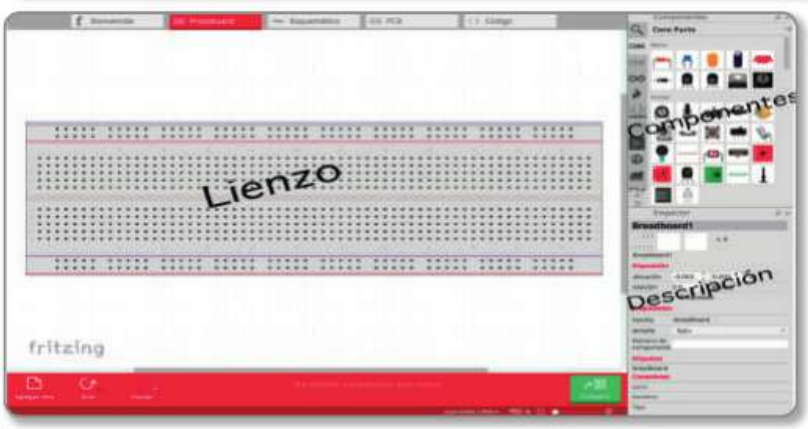
5. Para incorporar componentes, disponemos, en el margen derecho, de un conjunto de opciones ❹.

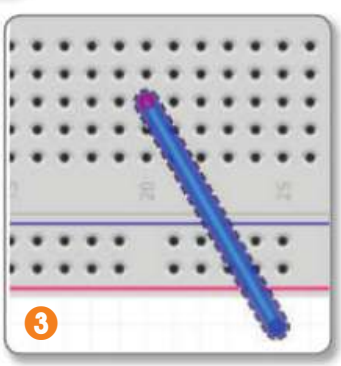
6. Si no encontramos el elemento que necesitamos, podemos buscarlo en nuestro navegador web. Posiblemente alguien lo haya creado y lo haya puesto a nuestra disposición ❺.

1



2





5