Entender el código para encender un LED

Como ya te comenté en la página anterior, lo que vamos a hacer ahora es entender el código que hemos utilizado para programar nuestro Arduino. El lenguaje de programación que empleamos para programar nuestro UNO (o cualquier otro Arduino o placa basada en Arduino que programemos con la IDE que acabamos de descargar) se llama también Arduino y está basado en el lenguaje de programación C++. Diremos que es una versión simplificada de C++.

> Es importante tener ciertas nociones de inglés para poder programar Arduino, ya que al ser un lenguaje basado en C++, C++ tiene como lenguaje natural de referencia el inglés.

Para entender el código que ha iluminado nuestro LED, lo que vamos a hacer es ir viéndolo párrafo a párrafo y así, explicaremos diferentes características que necesitaremos conocer para trabajar en nuestros propios proyectos.

¿Y por qué copiamos código en lugar de escribir el nuestro?

Esta pregunta puede venirnos a la cabeza y es normal. A la hora de comenzar a trabajar con Arduino, suelen seguirse los siguientes pasos:

1. Buscar ejemplos similares a lo que queremos hacer
2. Entender lo que hace el código y copiarlo
3. Realizar las modificaciones necesarias para adaptarlo a nuestro proyecto

¿Y por qué se suele hacer eso en lugar de escribir código desde cero?
Arduino se creó con la finalidad de que gente que no pertenece al mundo de la electrónica y la programación fuese capaz de programar y de realizar proyectos interactivos. Por ello, existe una amplia comunidad de creadores que comparten sus proyectos (tanto el hardware como el software) de manera libre, para que la gente pueda reutilizarlo y adaptarlo a sus necesidades. Muchos de los proyectos que queramos realizar, ya habrán sido realizados antes y además la gente que los comparte suele explicar detalladamente cómo replicarlos y por qué los han realizado de una manera determinada.

Aparte de facilitarnos el trabajo, reutilizar código nos puede enseñar funciones y características de Arduino que desconocíamos. Y por supuesto, siempre podremos decidir hacer las cosas de otra manera cuando tengamos los conocimientos suficientes. En algunas ocasiones es bueno intentar crear algo desde cero, pero en muchas ocasiones **no es necesario tratar de reinventar la rueda**.

¿No estamos plagiando/robando otros proyectos?

- No, los proyectos de Arduino que se comparten son de código abierto (open source) y eso significa que podemos reutilizarlos y adaptarlos a nuestras necesidades sin ningún problema.

Comencemos a entender el idioma de Arduino

Cuando programemos nuestro Arduino, tenemos que tener en mente que el código que escribamos, aparte de ser procesado por nuestro Arduino, es leído por nosotros y por cualquier otra persona que colabore en nuestro proyecto. Como ya se ha comentado, Arduino nació con la finalidad de aproximar la electrónica y la programación a cualquier persona que le interesase sin que sea necesario que tenga amplios conocimientos de programación. Por ello, el trabajo colaborativo y compartir el código es algo muy común. Para que eso se realice correctamente es necesario que nuestro código sea comprensible por nosotros y por cualquiera que pueda trabajar con nosotros en el proyecto.

Una manera de explicar y clarificar la finalidad de nuestro código es el uso de **comentarios**. Los comentarios son aquellas oraciones que no van destinadas a que las procese nuestro Arduino, sino que el destinatario somos nosotros mismos u otras personas. ¿Nosotros mismos? Sí. Imagina que estás trabajando en un proyecto de Arduino, pero que por alguna circunstancia tienes que dejar de trabajar en él y lo retomas un año después. Quizá después de tanto tiempo no recuerdas bien para qué habías escrito ese código... Los comentarios pueden ayudarte en esa tarea.

Échale un vistazo a este fragmento de código:

```
/*
  Blink

  Turns an LED on for one second, then off for one second, repeatedly.

  Most Arduinos have an on-board LED you can control. On the UNO, MEGA and ZERO
  it is attached to digital pin 13, on MKR1000 on pin 6. LED_BUILTIN is set to
```

```
  the correct LED pin independent of which board is used.
  If you want to know what pin the on-board LED is connected to on your Arduino
  model, check the Technical Specs of your board at:
  https://www.arduino.cc/en/Main/Products

  modified 8 May 2014
  by Scott Fitzgerald
  modified 2 Sep 2016
  by Arturo Guadalupi
  modified 8 Sep 2016
  by Colby Newman

  This example code is in the public domain.

  https://www.arduino.cc/en/Tutorial/BuiltInExamples/Blink
*/
```

Como puedes ver, se trata de un código separado en párrafos:

Blink: Lo primero que nos encontramos es el título del proyecto.
Descripción del proyecto: Breve descripción de la finalidad del código y explicación.
Modificaciones y autores: Fechas de modificación y los autores que lo han editado. Arduino se basa en compartir y reutilizar código, por ello es necesario indicar quién ha creado ese código y quién lo ha modificado. No debemos apropiarnos del código de otra persona sin mencionarlo.
Dominio público: cualquiera puede utilizar este código con los fines que prefiera, comerciales o no.
Web: Tutorial en el que se explica el proyecto.

Cuando nosotras creemos un proyecto, será necesario que escribamos algo parecido a esto al principio de nuestro código.

Y si te has fijado bien, al principio y al final encontramos **/*** y ***/** eso nos indica que lo que escribamos entre ambos símbolos es un comentario, sin importar que nos ocupe una línea o varias.
Normalmente se utiliza para comentar párrafos.
Su función es evitar que Arduino lo lea y que cuando el código sea compilado, se obvie. Si Arduino intentase procesar ese texto escrito en inglés (los comentarios también podríamos hacerlos en español) daría error porque no está escrito en el lenguaje que comprende Arduino.

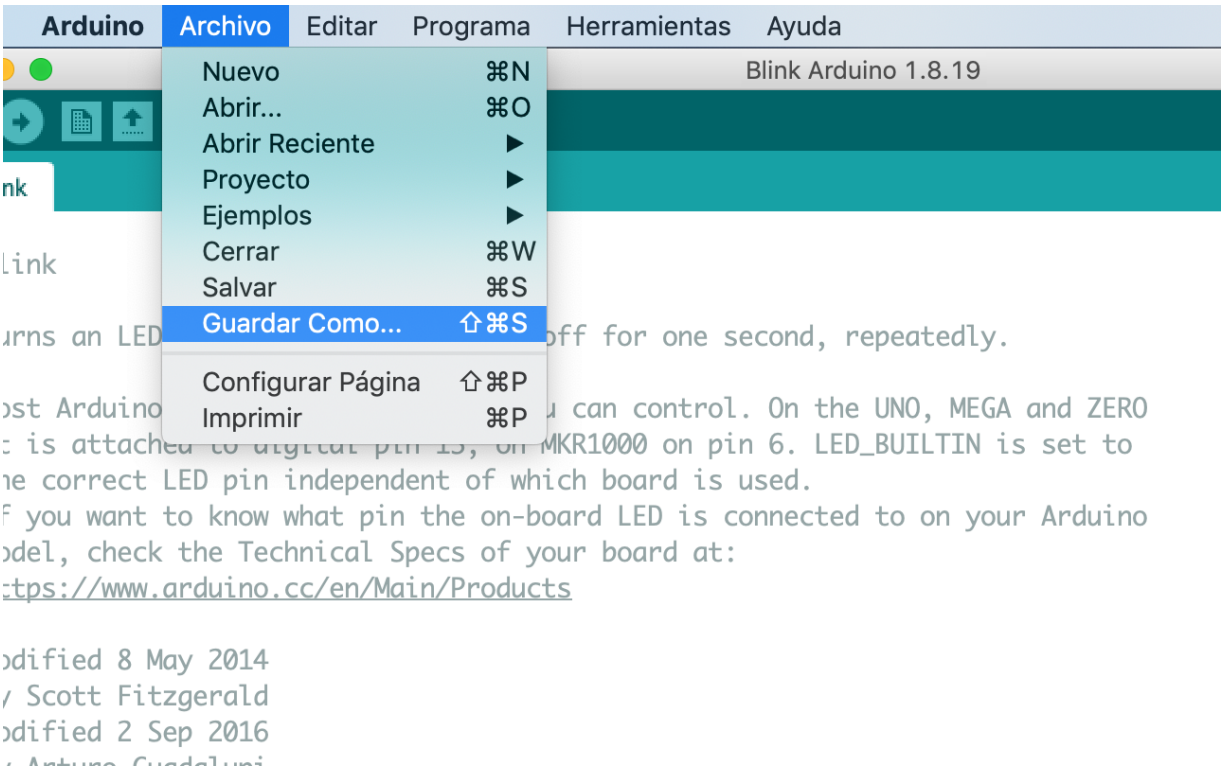
Otra manera de escribir **comentarios cuando solamente ocupan una línea** es empleando **//** únicamente al principio.

Os invito a que abráis Arduino con el **sketch Blink**, lo guardéis con otro nombre y hagáis pruebas con el texto de los comentarios.

Guardar un archivo preexistente de Arduino

Para guardar una copia del sketch Blink, lo que haremos será ir a **Archivo > Guardar como...** y ahí se nos abrirá una ventana en la que podremos escribir un nuevo nombre y elegir la ubicación para guardarlo.

> Yo os recomiendo que para guardar los sketches que escribamos en este taller os creéis una carpeta, por ejemplo en **Documentos** o en el **Escritorio**, y ahí Arduino os creará una subcarpeta para cada archivo nuevo que guardemos.



> Al guardar un archivo por primera vez, se genera el archivo con la extensión **.ino** y una carpeta que lo contiene. Cada **sketch** necesita estar dentro de una carpeta con su mismo nombre. ¿Por qué? Nosotros en este taller no vamos a emplear archivos externos aparte del sketch en nuestro proyecto, pero podría darse el caso que para funcionar, los necesitásemos; estos archivos deben estar dentro de la carpeta, sino, el programa no funcionaría correctamente.

void setup()

Una vez hemos entendido qué son los comentarios y cómo guardar nuestros archivos, vamos a continuar leyendo el código:

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}
```

Lo próximo que nos encontramos (tras el comentario **//** que ya hemos comentado en el párrafo anterior) son las palabras **void setup().** Bueno, la estructura correcta sería esta:

void setup() {
// aquí escribimos los comandos a ejecutar
}

> Es muy importante que **no olvidemos** ningún **paréntesis** ni **corchete**, o el programa no funcionará.

Eso es una función. Y, ¿qué es una función dentro del mundo de Arduino? Una función es un fragmento de código, un subalgoritmo dentro de nuestro algoritmo, que tiene como propósito resolver una tarea determinada.
La palabra **void**, significa que esta función no nos devuelve ningún valor (sea un número, un caracter, un string, etc.) y la palabra reservada **setup** indica que el código que escribamos entre las llaves **{}** se ejecutará **una sola vez** al inicio del programa.

> Una **palabra reservada** es aquella que dentro de un lenguaje de programación tiene un significado determinado. Lo iremos entendiendo a lo largo del taller. Más ejemplos.

> Tienes una lista completa con las funciones que podemos encontrar en Arduino, aquí.

En este caso, el código escrito dentro de la función setup() es:
pinMode(LED_BUILTIN, OUTPUT);

Esta línea está compuesta por **una función** y **dos palabras reservadas**.

pinMode(pin, modo) : es una función que nos ayuda a configurar los pines digitales de nuestro Arduino para que funcionen, o bien como entrada, o como salida de la información.

El **primer parámetro** de esta función recibe el **número** (o en este caso nombre, porque es un pin especial) del pin que vamos a utilizar.
El **segundo parámetro** recibe el modo: INPUT o OUTPUT, si lo vamos a utilizar como pin de entrada o de salida respectivamente. La palabra **LED_BUILTIN** es una palabra reservada y se refiere al LED que viene dentro de nuestra placa, como se ha comentado en la página anterior. Por tanto en esta línea le estamos diciendo a Arduino: "Oye, coge el pin al que llamas LED_BUILTIN y configúramelo como pin de salida, por favor."

Para finalizar, añadimos un **;**. Esto es importante, porque sin él tendríamos un error al verificar el código.

void loop()

```
// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```

Ahora que ya hemos visto qué es una función, nos va a resultar más fácil ver qué es esto de **void loop().** Si la función void setup() ejecutaba el código que contenía solamente una única vez al arrancar el programa, la función loop lo hace en bucle (como su propio nombre indica), ejecutando indefinidamente el código que escribimos **{}** entre las llaves. En este caso encontramos dos funciones, la primera es **digitalWrite()** y la segunda es **delay()**.

digitalWrite(): recibe dos parámetros, el primero es el led que utilizaremos, y que previamente habremos configurado como pin de salida (como hemos hecho con **pinMode()**), y el segundo es lo que haremos con ese pin. Al ser digital solo existen dos acciones, activar o desactivar el pin (**HIGH** o **LOW**). Si la acción es HIGH, el pin tendrá un voltaje de salida de 3.3V si la acción es LOW, el voltaje será 0V.

delay(): esta función hace que el arduino se detenga una determinada cantidad de tiempo, medido en milisegundos. Solamente cuenta con un parámetro.

Por tanto, a nuestro Arduino, dentro de void loop() le estamos diciendo: "Hey, enciéndeme el LED un segundo y apágalo otro segundo, por favor.

Al principio, puede parecer un poco farragoso, pero poco a poco iremos familiarizándonos con el lenguaje de Arduino y nos iremos entendiendo mejor.

> Para practicar, te aconsejo que modifiques los valores dentro de delay (sin que seanmenores a 500ms) y subas esas modificaciones a Arduino, para que veas cómo cambia lafrecuencia con la que parpadea el LED.

FUENTES:
Extraído de: https://libros.catedu.es/books/microcontroladores-vestibles-y-conectados-a-internet/page/entender-el-codigo-para-encender-el-led

---

Revision #1
Created 17 February 2023 11:30:16 by Julia del Río
Updated 17 February 2023 11:30:16 by Julia del Río