MÓDULO 3: ARDUINO

Vamos a aprender qué es Arduino, lo instalaremos y realizaremos nuestras dos primeras prácticas con él

- ¿Qué es Arduino?
- Instalar la IDE de Arduino
- El puerto serie de nuestro UNO
- Práctica 2: Arduino UNO + LED
- Entender el código para encender un LED
- Práctica 3: Hello Arduino World (Hola Mundo Arduino)

Copyright 2026

¿Qué es Arduino?

¿Qué es un microcontrolador?

> Un microcontrolador es un "microordenador" en un chip. Tiene una CPU, RAM (memoria de acceso aleatorio), registros de funciones especiales, memoria ROM de programa, memoria ROM de datos, de uno a varios puertos paralelos de E/S (entrada/salida), y puede tener una gran cantidad de periféricos en el chip que incluyen, pero no se limitan, a un convertidor analógico-digital (ADC), un convertidor digital-analógico (DAC), una UART serie, uno o varios temporizadores, comparadores/referencia de tensión en el chip, módulo de captura/comparación/PWM (Pulse Width Modulation), puerto serie síncrono maestro para comunicaciones SPI (Serial Peripheral Interface)/I2C (Inter Integrated Circuit), puerto USB, puerto ethernet, osciladores en el chip, junto con una serie de otros periféricos.

Todo esto nos puede parecer farragoso en un primer momento, pero si lo vemos con imágenes, aplicado a un ejemplo de Arduino en concreto y os lo voy comentando, lo vamos a acabar entendiendo perfectamente.

Espera un segundo, pero ¿Arduino es un microcontrolador?

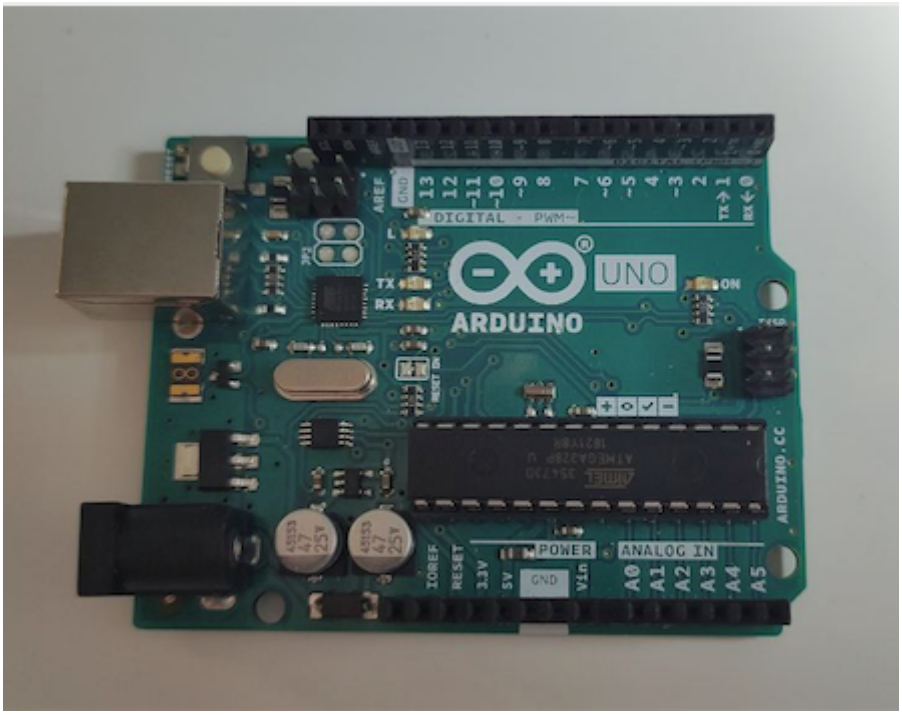
Aunque utilicemos Arduino y microcontrolador como sinónimos, un Arduino en realidad es una plataforma de desarrollo que entre otros componentes, contiene un microcontrolador. Junto a este microcontrolador encontraremos una serie de entradas y salidas que nos permitirán añadir interactividad a nuestros proyectos, empleando los sensores y actuadores que hemos mencionado en la página anterior.

Copyright 2026

El nacimiento y propósito de los Arduinos

Aquí no vamos a hablar de quiénes o cuándo crearon Arduino, ya que eso lo podemos encontrar fácilmente en este enlace (el cual os aconsejo leer); en cambio, sí que vamos a hablar del propósito inicial con el que fue creado: facilitar la comunicación entre un microcontrolador y otros dispositivos. De esta manera, es sencillo incentivar el desarrollo de procesos creativos y el diseño de proyectos tanto a personas de una edad temprana, como a aquellas que sin tener conocimientos amplios de informática o electrónica quieran embarcarse a experimentar con proyectos interactivos.

El formato de Arduino hace posible que podamos adentrarnos en la electrónica sin sentirnos abrumados con información demasiado técnica. Bueno, a continuación, paso a mostraros la imagen de un Arduino:



Como podemos ver, un Arduino cuenta con muchos componentes dentro de su tarjeta, algunos de los cuales ya hemos mencionado en la cita que abre esta página. No voy a entrar a explicar cada uno de ellos, pero sí los más importantes, los que nos ayudarán a entender cómo funciona un microcontrolador y para qué (y para qué no) lo podemos utilizar.

Un buen símil para un Arduino es el de una **caja negra**. ¿Por qué? Porque no vamos a necesitar conocer el funcionamiento de cada parte para poder utilizarlo. Lo que es importante conocer es que cuenta con componentes que son de entrada, otros procesan la información y otros son de salida.

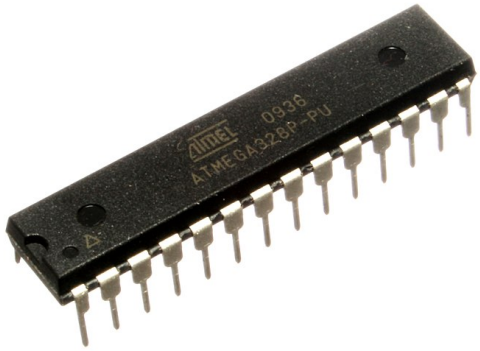
Copyright 2026

```
 ___________    ___________    _________
| ENTRADA |-->| PROCESO |-->| SALIDA |
```

A continuación, veremos cuáles son los componentes de entrada, procesamiento y salida sin perder de vista la definición del comienzo de la página.

Arduino UNO: partes que debemos conocer

Uno de los primeros componentes mencionados es el propio **microcontrolador**. Y es que, el Arduino en sí ya hemos dicho que no es un microcontrolador, sino una placa de desarrollo. El

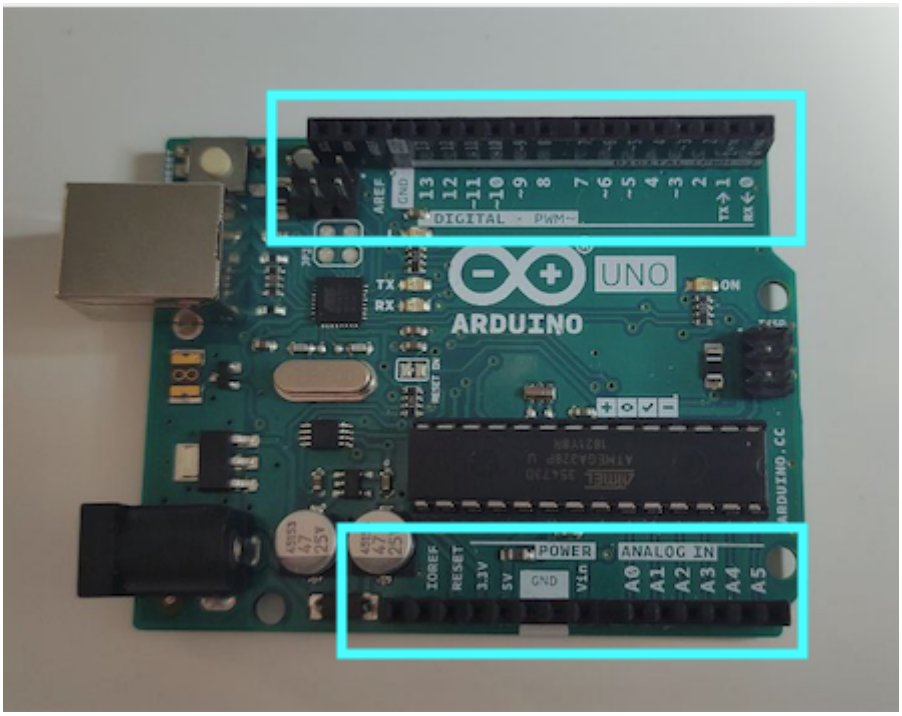


Se llama ATMEGA 328P y es el que tiene el Arduino que vamos emplear en este curso: el **Arduino UNO**. Pero claro, existen otros tipos de Arduinos, los cuales cuentan con otros microcontroladores.

Para profundizar sobre el ATMEGA328P podéis encontrar más información aquí.

Bien, este microcontrolador, como podéis ver, cuenta con una serie de 'patitas' a la cuales vamos a llamar **pines**. (Esta palabra es una de las importantes y la cual vamos a leer en bastantes ocasiones a lo largo de estas páginas). Estos pines son los que nos van a permitir programar, recibir y enviar la información. Para poder acceder a ellos, Arduino nos facilita las cosas con una serie de pines de entrada que encontramos en la imagen siguiente:

Copyright 2026



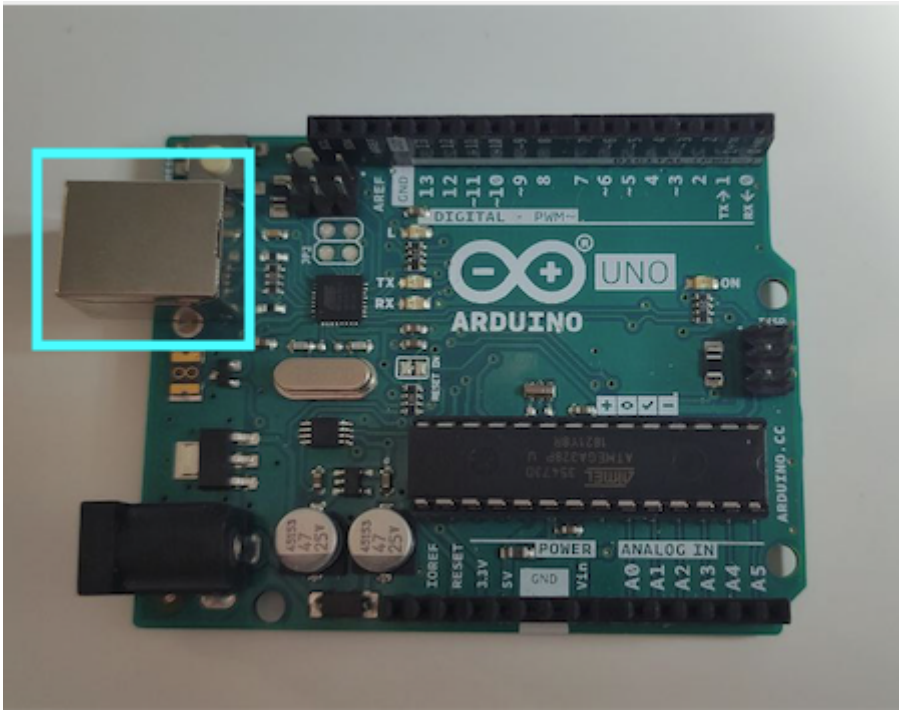
Como podéis comprobar, esos pines se encuentran a ambos lados del ATMEGA 328P, aunque visualmente no parezcan directamente en contacto. Ya veremos, qué cables y cómo los tenemos que conectar más adelante... Por el momento, nos quedaremos con que esos pines serán los que tendremos que utilizar para conectar los sensores y actuadores.

También veremos que algunas de estas entradas son digitales y otras analógicas, pero de eso nos ocuparemos más adelante...

Por tanto, nuestro ATMEGA 328P cuenta con una CPU, RAM (memoria de acceso aleatorio), registros de funciones especiales, memoria ROM de programa, memoria ROM de datos, puertos paralelos de E/S (entrada/salida), etc. Para el tipo de proyecto que vamos a construir, no es necesario preocuparse de sobre la ROM ni sobre la RAM, ya que no vamos a escribir programas excesivamente extensos (quizás en el futuro...). Los componentes que se mencionan en la definición que abre esta página y que sí vamos a necesitar conocer son otros como por ejemplo el puerto USB.

En el Arduino UNO, el puerto USB se encuentra aquí:

Copyright 2026



AB USB:



Con él, conectaremos nuestro Arduino al ordenador y seremos capaces de programarlo y también de proporcionarle el voltaje necesario para alimentarlo a él y a los sensores que conectemos a él.

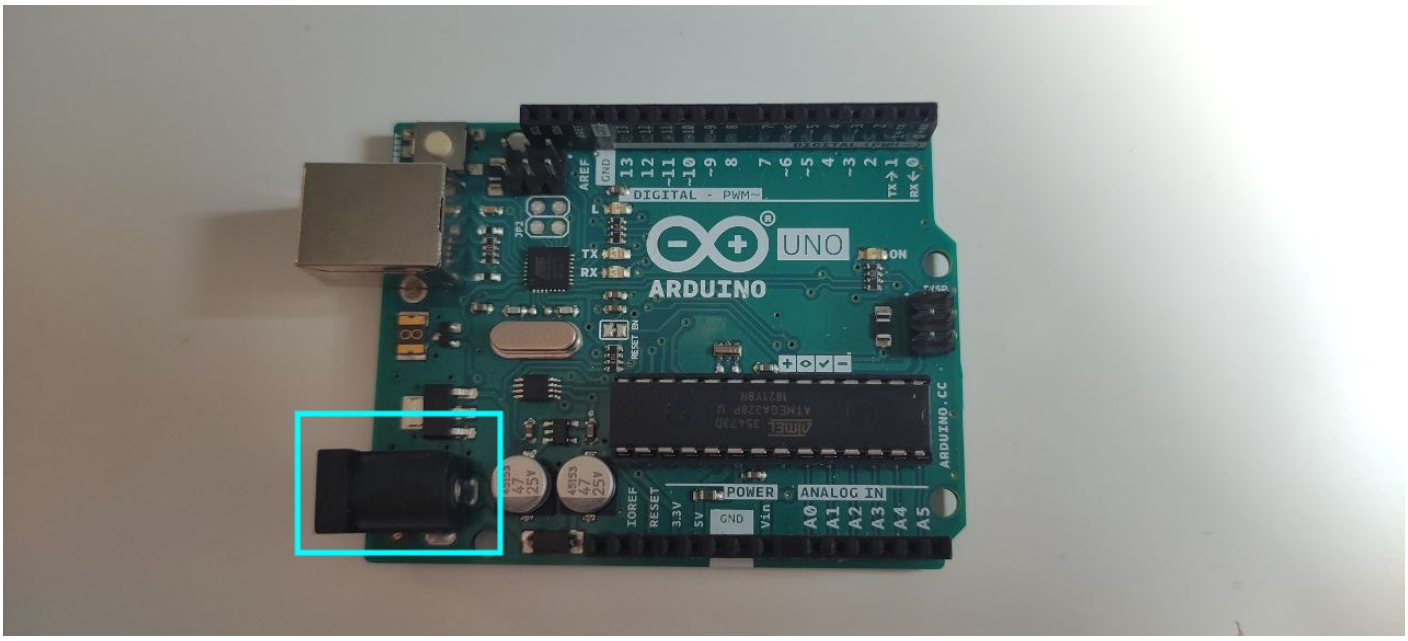
> Tendremos que tener cuidado con qué sensores conectamos, porque algunos de ellos necesitarán más alimentación que la que proporcionamos a nuestro Arduino a través del USB, que son 5V. Algunos sensores y actuadores funcionan a 12V o a voltajes mayores. Para este curso no será una preocupación, porque ninguno de los sensores empleados necesita alimentación extra. No obstante, tendremos que tenerlo en cuenta si en el futuro queremos emplear unos sensores diferentes.

Copyright 2026

Aparte de mediante el USB, es posible alimentar nuestro Arduino UNO utilizando una pila de 9V y un adaptador o un cargador conectado a la corriente con un voltaje de 7 a 12V como máximo.

> Para saber más sobre las maneras de alimentar nuestro Arduino, puede ser útil este artículo.

El lugar al que conectaremos nuestra pila o adaptador es este:



Como puedes comprobar, no se han explicado todas las partes que conforman el Arduino UNO. Esto es, porque para los propósitos de este curso no va a ser necesario conocer esta placa en profundidad, sino entender qué función realizan las partes que vamos a utilizar y hacernos una idea general sobre las partes que comparten la mayoría de los Arduinos.

FUENTES:

Contenido obtenido de: https://libros.catedu.es/books/microcontroladores-vestibles-y-conectados-a-internet/page/comenzamos-microcontroladores

Copyright 2026

Instalar la IDE de Arduino

Lo primero que tenemos que hacer es instalar la IDE (Entorno de Desarrollo Integrado) de Arduino en nuestro ordenador. Dependiendo del sistema operativo que empleemos, tendremos que suguir unas instrucciones u otras.

A continuación te dejo los enlaces para instalarlo tanto en Windows, Linux o Mac.

Arduino IDE Linux

Arduino IDE macOS

Arduino IDE Windows

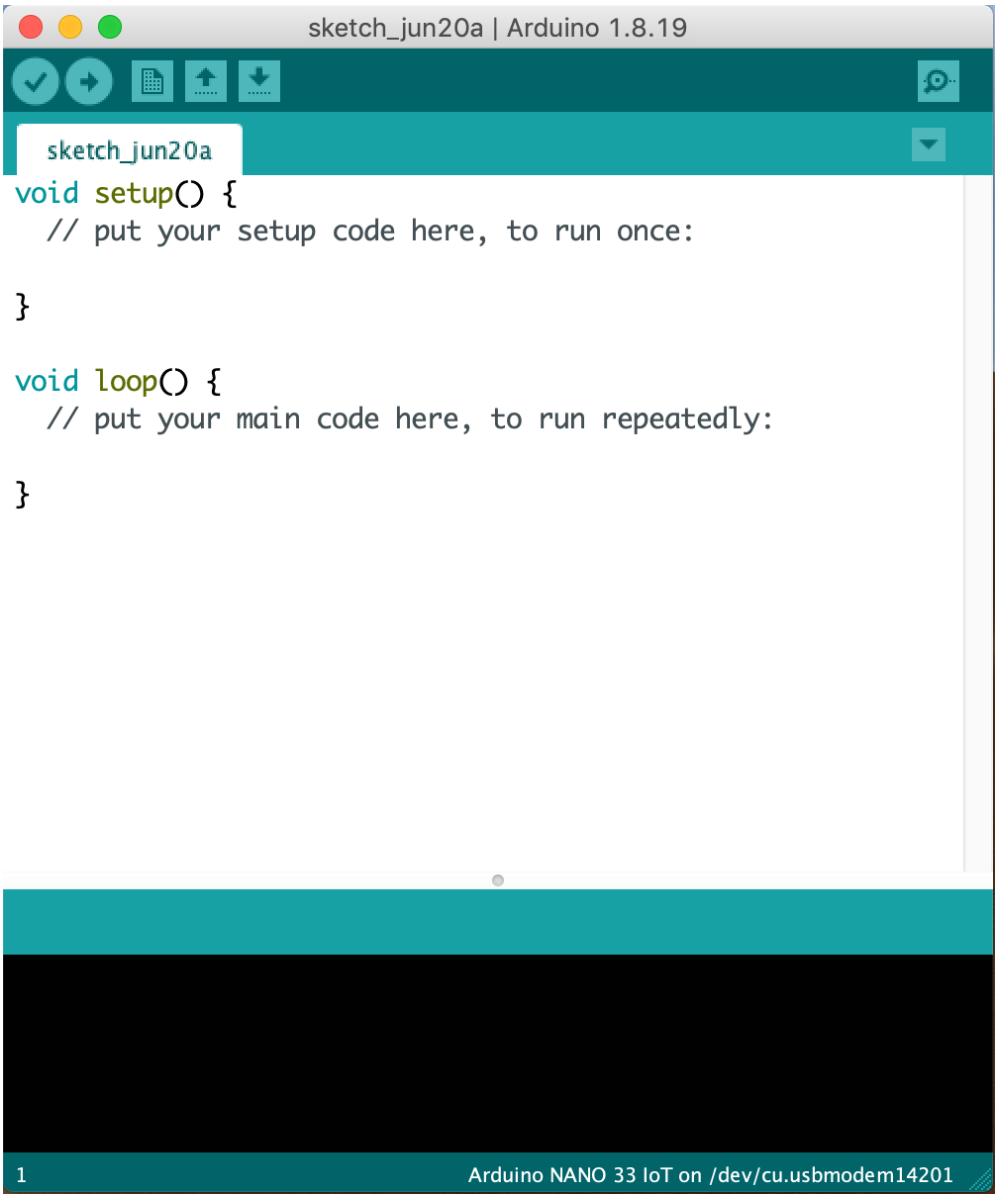
La descarga e instalación nos puede llevar de 5 a 20 minutos, dependiendo de la velocidad de nuestra conexión a internet y del sistema operativo que tengamos, ya que el procedimiento varía notablemente de uno a otro.

> En este taller, instalamos la IDE de Arduino en nuestro ordenador, aunque si durante la instalación tuviéseis muchos problemas (que no suele ser común, aunque puede suceder si, por ejemplo el ordenador que utilizáis tiene un sistema operativo muy antiguo) podrías utilizar la versión online. Para ello es necesario crear una cuenta en la web de Arduino y seguir los siguientes pasos.

Abrimos la aplicación

Una vez hayamos instalado la IDE, estaremos listos para abrirla y nos encontraremos con algo muy similar a esto:

Copyright 2026



A través de esta aplicación programaremos nuestro Arduino. Lo primero que vamos a hacer es ver qué podemos hacer con algunos de los botones que utilizaremos con más frecuencia. Comenzaremos de arriba a abajo y de izquierda a derecha.

> Se recomienda abrir la aplicación para poder ir viendo cada botón conforme se explica su función.

- **Símbolo tick (Verificar):** con él, comprobaremos que el código que queremos subir a nuestro Arduino no contiene fallos. ¿Dónde nos comunicará la aplicación si nuestro código tiene fallos o no? Lo veremos en el rectángulo negro de la parte inferior. Esa parte se denomina consola. ¿Siempre que nuestro código pase satisfactoriamente el paso de verifiación nos aseguramos de que funcione correctamente? Pues, desafortunadamente

no. Esta verificación nos asegura que no existen fallos sintácticos: palabras mal escritas, expresiones incorrectas... pero no nos asegura que la lógica de nuestro programa sea la correcta.

- **Flecha a la derecha (Cargar):** una vez nos hayamos asegurado de que nuestro código es correcto sintácticamente, al darle a la flecha, comenzaremos a subirlo a nuestro Arduino.
- **Hoja de papel (Nuevo):** este botón abre una pestaña nueva para que podamos escribir código. No la utilizaremos en este taller.
- **Flecha arriba (Abrir):** nos permite abrir un archivo (sketch) existente.
- **Flecha abajo (Guardar):** permite guardar el archivo en el que estamos trabajando actualmente.
- **Lupa (monitor serial): a**bre una ventana que nos permite ver la información serie que la placa transmite. Es muy útil para encontrar fallos en el código y ver si el programa y el Arduino funcionan correctamente. Lo veremos con más detalle en las partes prácticas del taller.
- **Nombre del sketch:** en esta pestaña encontraremos el nombre de nuestro sketch. Por defecto será algo similar a *sketch_fecha*
- **Área de escritura:** en esta zona escribiremos nuestro programa.
- **Área de mensajes:** en esta parte, el IDE nos indicará si existen errores en el código.
- **Consola de texto:** la consola nos mostrará los mensajes de error completos, es muy útil a la hora de depurar nuestro código.
- **Placa y puerto serie**: se muestra el nombre de la placa (Arduino) que tenemos conectada (o la última que hemos conectado, si no hay ninguna conectada) y el puerto al que está conectada.

Aparte de estas herramientas, contamos con una barra de herramientas en la parte superior (Arduino, Archivo, Editar...). El aspecto de esta barra de herramientas puede variar ligeramente de un sistema operativo a otro, pero las funciones serán las mismas. Algunas de ellas las veremos a continuación.

REFERENCIAS:

Instalar la IDE de Arduino: https://libros.catedu.es/books/microcontroladores-vestibles-y-conectados-a-internet/page/practica-11-descargar-la-ide-de-arduino-y-encender-un-led

Copyright 2026

El puerto serie de nuestro UNO

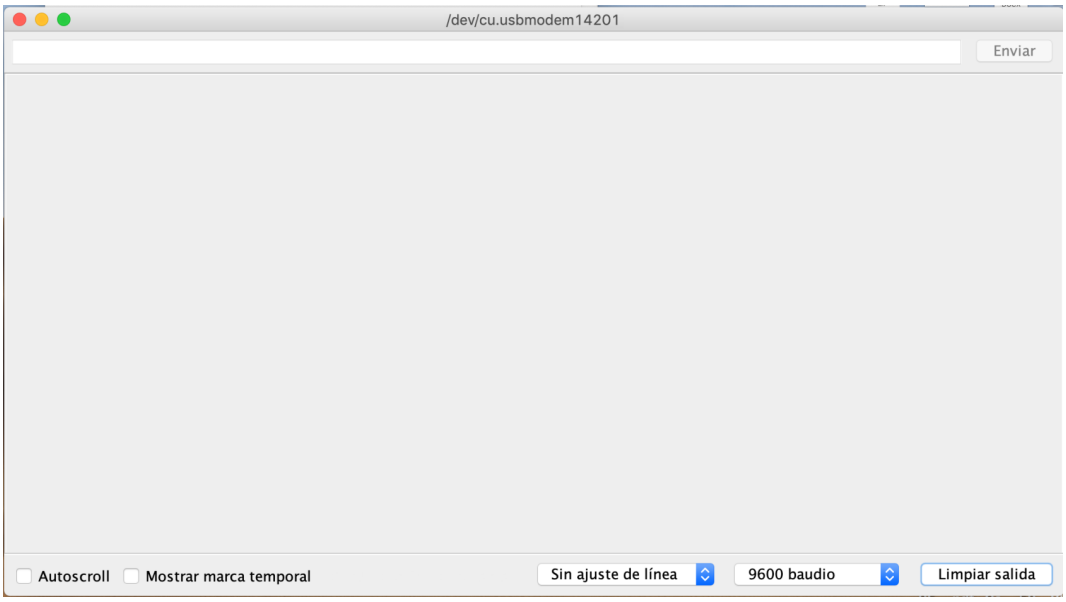
Una parte fundamental de nuestro Arduino es el puerto serie. Gracias a él se comunicará con nuestro ordenador y seremos capaces de enviar y recibir información con nuestro Arduino. ¿Existen otras maneras? Sí, pero esta es la que emplearemos a lo largo de este curso para ver qué valores reciben nuestros sensores a través de la pantalla de nuestro ordenador.

Para abrir nuestro puerto serie en Arduino necesitaremos hacer click en **la lupa** que se encuentra en la parte superior derecha de nuestro sketch:



Una vez hayas instalado Arduino, puedes venir a esta sección a probarlo, si no, lo probaremos en las prácticas, no te preocupes.

Si tenemos nuestra placa de Arduino conectada, al hacer clic sobre la lupa se nos abrirá esta ventana:



Pero si no tenemos nuestro Arduino conectado, nos dará un error similar a este:



Existen ciertos parámetros que podemos modificar en la ventana de nuestro puerto serie, dependiendo de a qué velocidad o baudios nos tengamos que comunicar con nuestro Arduino. En la mayoría de los proyectos de este curso no será necesario modificar nada, porque emplearemos los 9600 baudios que aparecen por defecto, solamente en una de las prácticas, necesitaremos cambiar ese valor a 115200 baudios.

Práctica 2: Arduino UNO + LED

Llegadas a este punto, estamos listas para conectar nuestro Arduino al ordenador. Para ello, necesitaremos nuestro cable USB-AB.

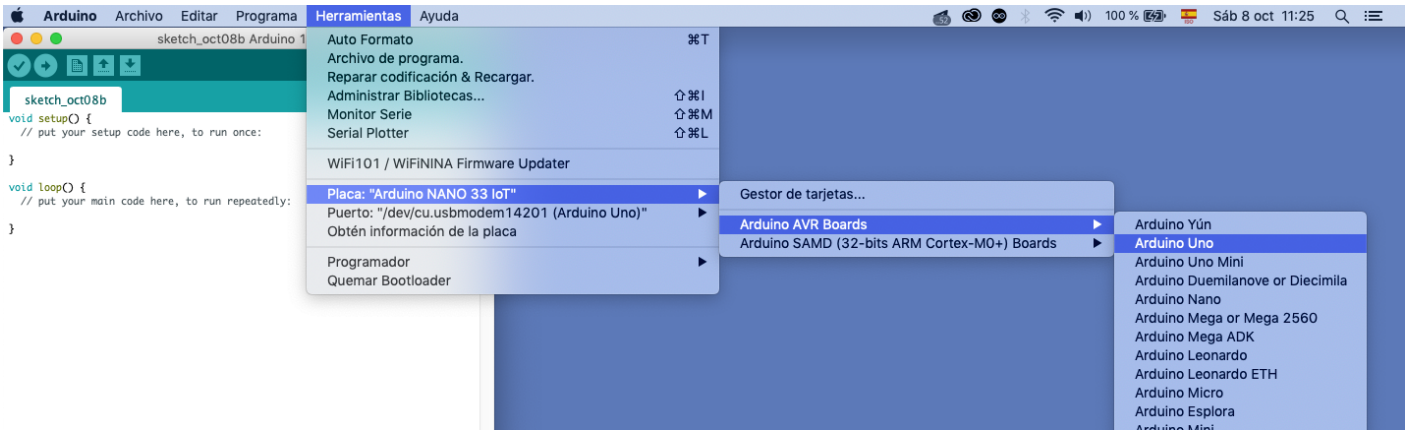
Una vez hayamos conectado un extremo a nuestro Arduino y el otro al puerto USB de nuestro ordenador, tendremos que ayudar a la IDE a que reconozca a nuestro Arduino. Lo primero que veremos será un **LED VERDE** que se enciende y junto al que pone **ON**. Eso nos indica que la placa está recibiendo corriente desde nuestro ordenador.

> Los usuarios de Windows (seguramente) tendrán que instalar una serie de drivers una vez se conecte el Arduino al ordenador.

Lo siguiente que tendremos que hacer es ir a la barra de herramientas superior y precisamente en **Herramientas** tendremos que configurar dos cosas, **la placa y el puerto**.

Configuramos la placa

En **Herramientas > Placa** nos aparecerá nuestro Arduino UNO, tendremos que seleccionarlo

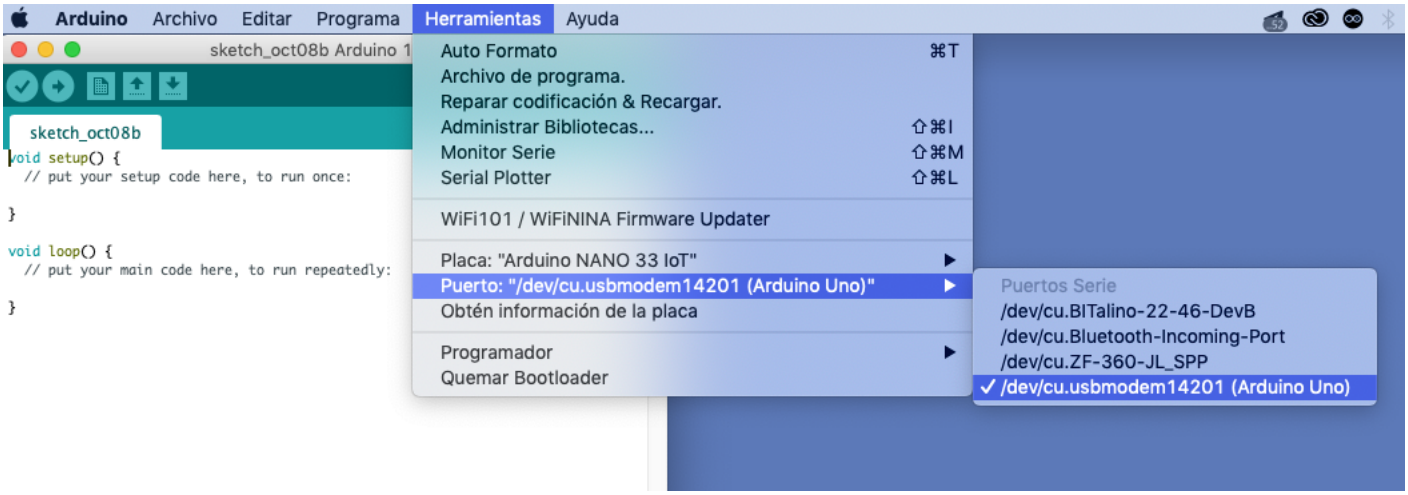


Al seleccionarla, le estamos diciendo a nuestro Arduino "¡Hey! Prepárate, voy a conectarte un Arduino UNO", por lo que si al final conectásemos otro tipo de Arduino, la IDE no la reconocería y no podríamos programarlo.

> Si utilizamos otro tipo de Arduino, lo primero que tenemos que hacer es decírselo a Arduino seleccionando la placa correcta en **Placa**, como acabamos de hacer.

Configuramos el puerto

Una vez le hemos dicho la placa que vamos a conectar, nuestro IDE necesita saber en qué puerto USB la hemos conectado. Para ello, tendremos que ir a **Herramientas > Puerto**



Para los usuarios de **Windows** podría tener esta apariencia:

```
<COM29> (Arduino UNO)
```

Para los de **MAC** o **Linux**, la apariencia puede ser parecida a esta:

```
/dev/cu.usbmodem14112 (Arduino UNO)
```

Una vez hayamos seleccionado nuestra placa y puerto, pasaremos a encender el LED.

Y se hizo la luz

Para encender nuestro LED, vamos a utilizar un ejemplo de los que se nos proporcionan desde la propia IDE de Arduino.

> Estos ejemplos nos pueden ser útiles para descubrir lo que podemos hacer con estas placas, por lo que os recomiendo que les echéis un vistazo.

El ejemplo que vamos a utilizar se llama **Blink** y nos va a venir bien para asegurarnos de que la instalación y toda la configuración se han realizado correctamente. Para abrir y subir este ejemplo a nuestro Arduino tenemos que navegar a: **Archivo > Ejemplos > 01.Basics > Blink**.



Una vez hagamos click, se abrirá una nueva ventana de Arduino con el código que permitirá que nuestro LED comience a parpadear. El código que veremos será el siguiente:

```
/*
  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.

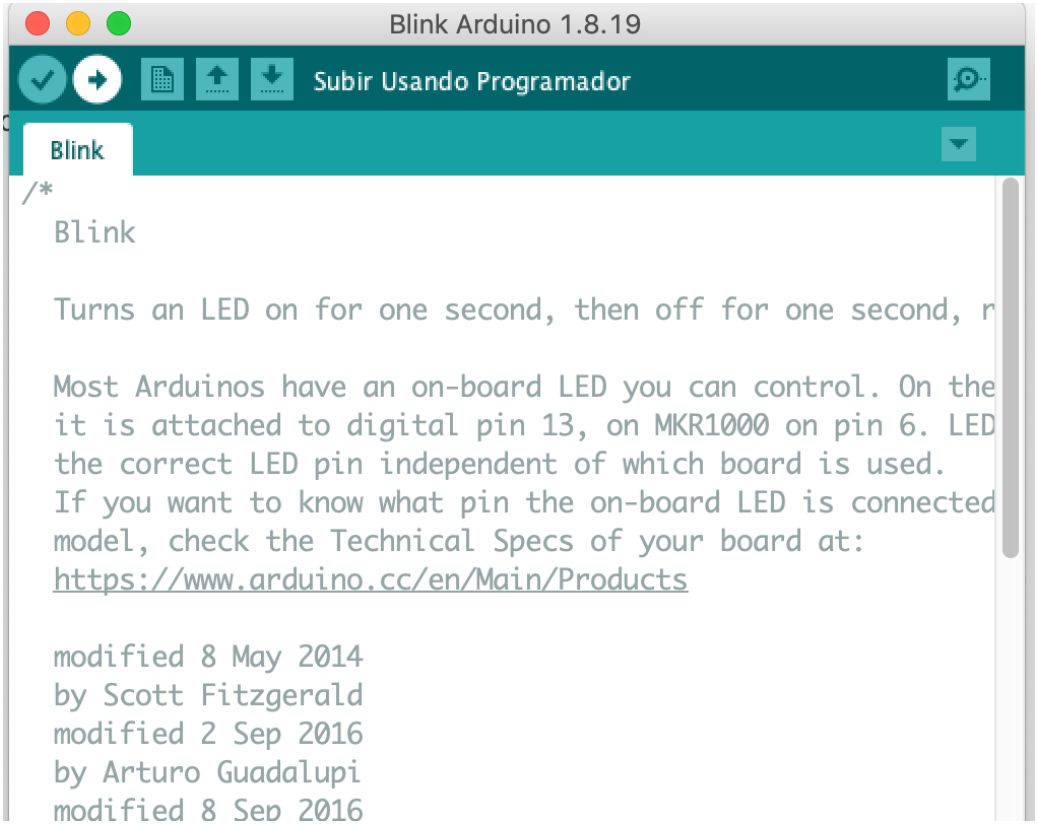
  https://www.arduino.cc/en/Tutorial/BuiltInExamples/Blink
*/

// 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);
}

// 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, no vamos a entrar a explicar qué significa este código, eso lo haremos en la página siguiente. Lo que sí vamos a hacer es darle a la **flecha a la derecha** para subir ese código a nuestro UNO:

Copyright 2026



Y si todo ha ido bien, lo que veremos será un LED naranja, cerca del conector USB, que permanecerá un segundo encendido, un segundo apagado.

¡Y ya lo tenemos!

Copyright 2026



Es posible que nos de algún error y que no funcione. Eso no significa que algo esté roto. Como ya avisé en la primera página del taller, no siempre funciona todo a la primera. Si en la consola vemos algún error, tendremos que asegurarnos de que hemos seleccionado el

> puerto y la placa correctas y que hemos seguido todos los pasos correctamente, sin olvidar ninguno.

¿Qué tenemos que entregar?

Tendrás que subir un video o .gif en el que se vea tu Arduino conectado al ordenador con el LED parpadeando y luego la pantalla de tu ordenador con el sketch abierto.

FUENTES:
Basado en: https://libros.catedu.es/books/microcontroladores-vestibles-y-conectados-a-internet/page/practica-11-descargar-la-ide-de-arduino-y-encender-un-led

Copyright 2026

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.

Copyright 2026

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.

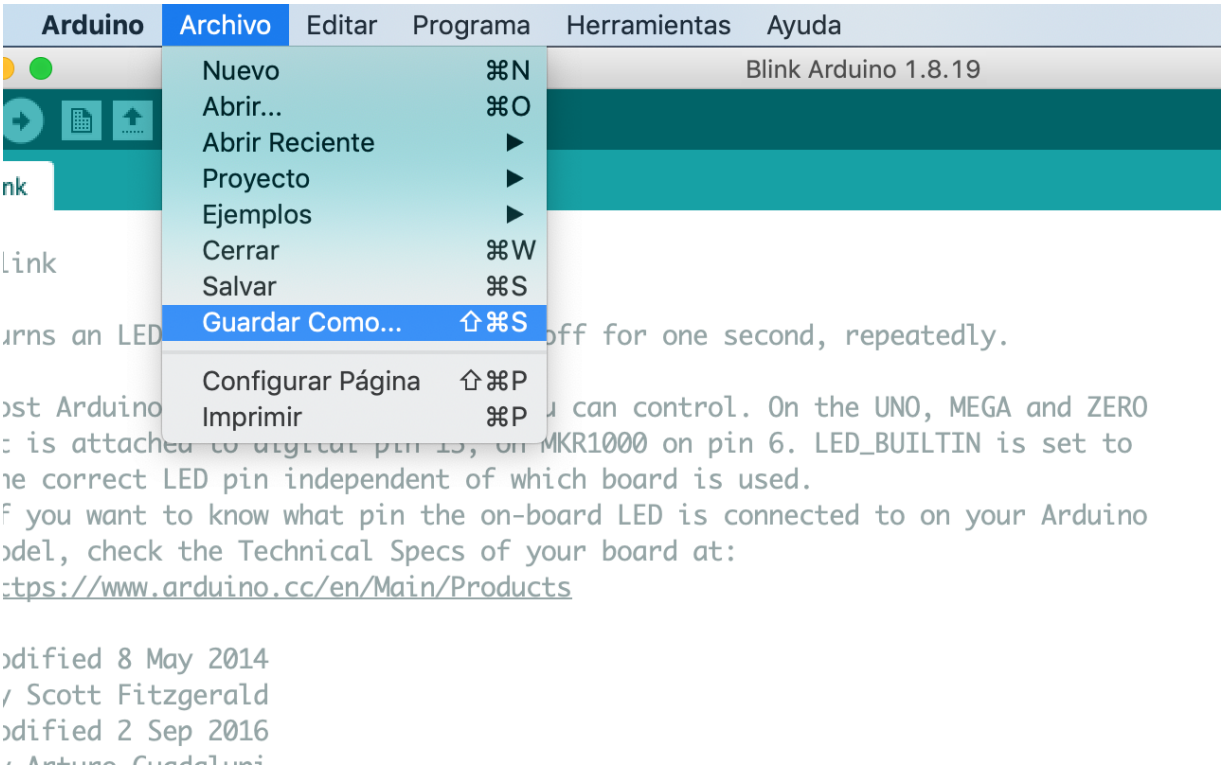
Otra manera de escribir **comentarios cuando solamente ocupan una línea** es empleando **//** únicamente al principio.

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 sean menores a 500ms) y subas esas modificaciones a Arduino, para que veas cómo cambia la frecuencia 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

Copyright 2026

Práctica 3: Hello Arduino World (Hola Mundo Arduino)

En Arduino, al igual que en Pure Data, tenemos la opción de escribir un programa del tipo 'Hello, World!'. Para ello, solamente necesitaremos nuestro Arduino y el cable USB AB.

Lo primero que haremos será **conectar nuestro UNO al ordenador**. Una vez lo hayamos conectado, **abriremos la IDE de Arduino**. Si justo has realizado la práctica anterior, no será necesario que selecciones la placa, ya que seguirá figurando tu UNO. Si lo has conectado a otro puerto USB, tendrás que seleccionarlo en **Herramientas > Puerto,** como hemos hecho en la práctica anterior.

Escribimos nuestro código

Para esta práctica no vamos a tener que escribir mucho. Lo primero que haremos, en la función **setup()**, para poder ver escritas nuestras palabras de saludo al mundo, será **iniciar la comunicación con el puerto serie** en 9600 baudios. Después imprimiremos por el puerto serie las palabras "**Hello, World!**". Para ello, es necesario que usemos **comillas**, así la IDE sabe que lo que queremos imprimir es una cadena de caracteres (String).

En la función **loop() no será necesario que escribamos nada.**

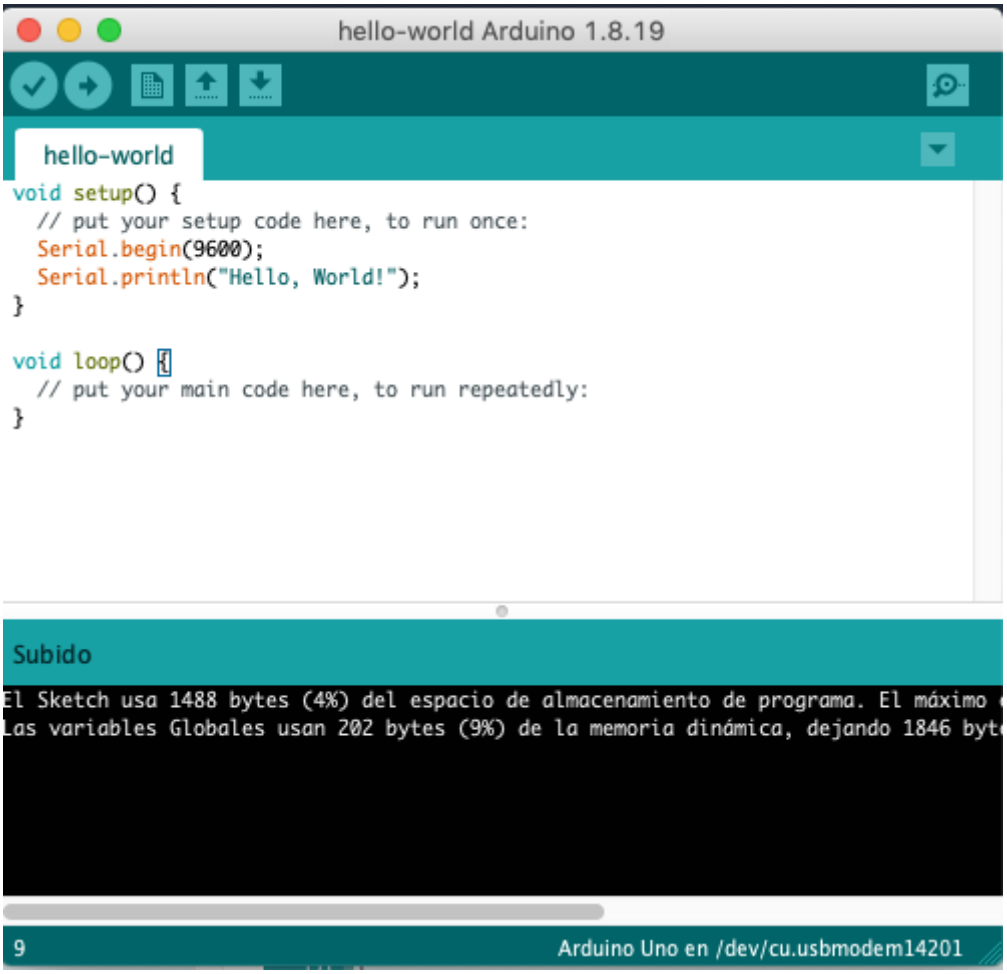
El código tendrá el siguiente aspecto:

```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  Serial.println("Hello, World!");
}

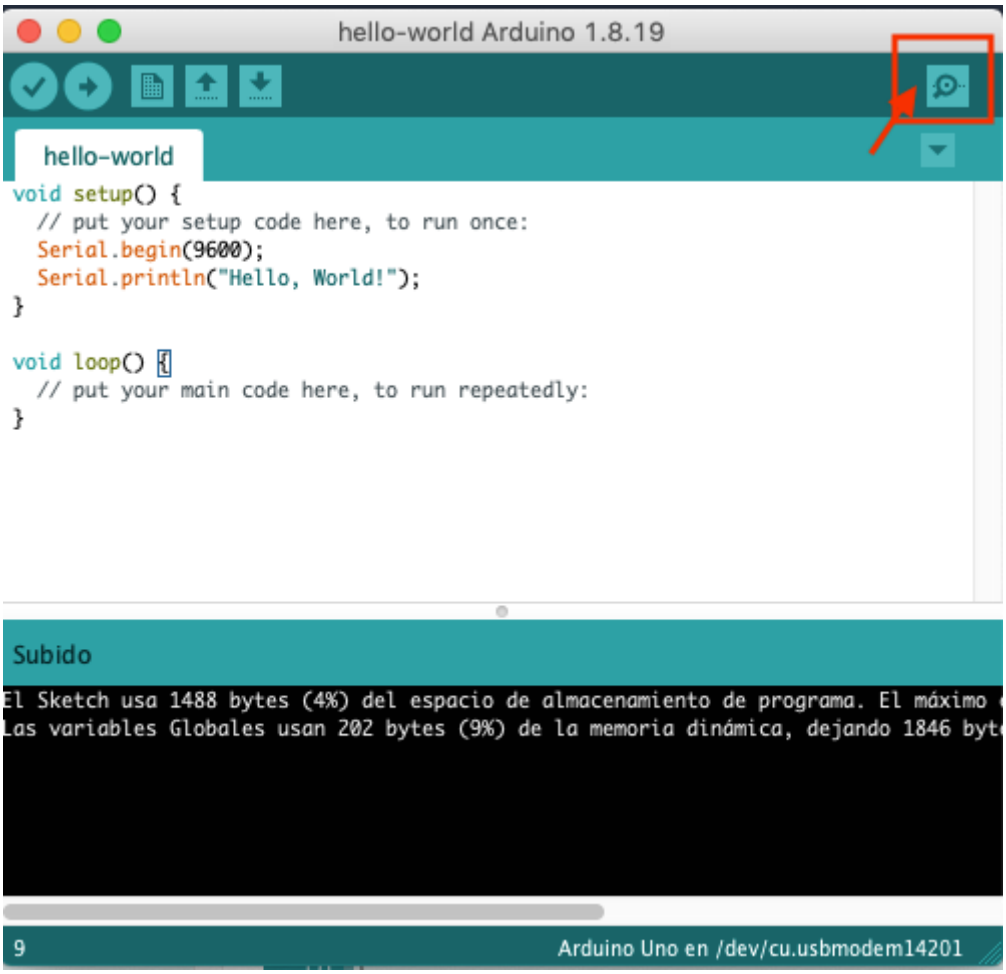
void loop() {
```

```
  // put your main code here, to run repeatedly:
```

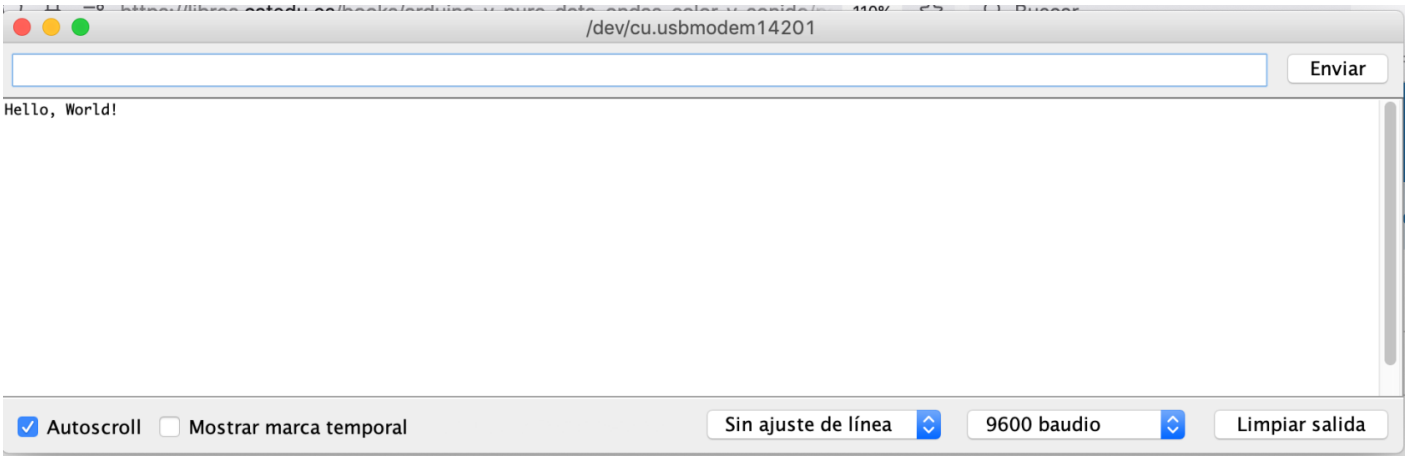
Este será el aspecto de nuestro programa en la IDE de Arduino. Cuando lo tengamos, le daremos al **botón circular con el símbolo del tick**, para verificar y a continuación al **botón circular con la flecha hacia la derecha**, para subirlo a nuestro UNO:



Una vez lo hayamos subido, tendremos que hacer click sobre **la lupa** de la esquina superior derecha, para abrir el puerto serie:



Una vez hayamos hecho click, se abrirá una ventana emergente en la que aparecerán escritas las palabras **"Hello, World!":**



¿Qué tenemos que entregar?

Tendrás que subir un video o .gif en el que se vea tu Arduino conectado al ordenador y luego el puerto serie abierto con las palabras Hello, World! impresas como se ve en la imagen superior.

FUENTES:

Hello World con Arduino: https://arduinogetstarted.com/tutorials/arduino-hello-world

Copyright 2026