

INTRODUCCIÓN a la **PROGRAMACIÓN CON ARDUINO**

Guía básica con ejemplos prácticos



**Maxwell Arbey
Salazar Guilcamaigua**

**Paola Alexandra
Portero Donoso**

Diseño de carátula y edición: D.I. Yunisley Bruno Díaz
Dirección editorial: PhD. Jorge Luis León González

Sobre la presente edición:
© Editorial EXCED, 2025

ISBN: 978-9942-560-00-1

Podrá reproducirse, de forma parcial o total el contenido de esta obra, siempre que se haga de forma literal y se mencione la fuente.

El contenido del texto y sus datos en su forma, corrección y confiabilidad son de exclusiva responsabilidad de los autores, y no representan necesariamente la posición oficial de la editorial EXCED.

Se permite descargar la obra y compartirla siempre que se den los créditos a los autores, pero sin posibilidad de alterarla de ninguna forma ni utilizarla con fines comerciales. El manuscrito fue previamente sometido a evaluación abierta por pares y aprobado por el Consejo Editorial, con base en criterios de neutralidad e imparcialidad académica.

EXCED se compromete a garantizar la integridad editorial en todas las etapas del proceso de publicación, evitando plagios, datos o resultados fraudulentos y evitando que los intereses económicos comprometan los estándares éticos de la publicación.



Editorial EXCED
Dr. Kennedy Nueva. 2do Callejón 11
A. Manzana 42, Número 26.
Guayaquil, Ecuador.
E-mail: editorial@excedinter.com

INTRODUCCIÓN
a la
PROGRAMACIÓN
CON ARDUINO

Guía básica con ejemplos prácticos



Maxwell Arbey Salazar Guilcamaigua
Paola Alexandra Portero Donoso

COMITÉ EDITORIAL

Maritza Librada Cáceres-Mesa,

Universidad Autónoma del Estado de Hidalgo, México

Yamilka Pino-Sera,

Universidad de Holguín, Cuba

Samuel Sánchez-Gálvez,

Universidad de Guayaquil, Ecuador

María Hernández-Hernández,

Universidad de Alicante, España

Héctor Tecumshé-Mojica-Zárate,

Universidad de La Sierra, México

Yadir Torres-Hernández,

Universidad de Sevilla, España

Rodolfo Máximo Fernández-Romo,

Universidad Autónoma de Chile, Chile

Kenia Noguera-Nuñez,

Universidad Católica Santo Domingo, República Dominicana

Oscar Alberto Pérez-Peña,

Universidad Internacional de La Rioja, España

Marily Rafaela Fuentes-Aguila,

Universidad Metropolitana, Ecuador

Nancy Malavé-Quintana,

Universidad Rey Juan Carlos, España

Lázaro Salomón Dibut-Toledo,

Universidad del Golfo de California, México

Luisa Morales-Maure,

Universidad de Panamá, Panamá

Farshid Hadi,

Islamic Azad University, Irán

Mikhail Benet-Rodríguez,

Fundación Universitaria Cafam, Colombia

| | |
|---|-----------|
| PRÓLOGO | 9 |
| Introducción | 11 |
| CAPÍTULO I. | |
| Introducción a Arduino | |
| ¿Qué es Arduino? | 13 |
| Tipos de Arduino | 14 |
| Componentes principales de Arduino UNO | 20 |
| Instalación del IDE de Arduino | 21 |
| Introducción a las plataformas de simulación | 25 |
| Guía para usar Tinkercad con Arduino Uno | 25 |
| Guía para usar Wokwi con Arduino Uno | 32 |
| CAPÍTULO II. | |
| Fundamentos de Programación en Arduino | |
| Interfaz del Arduino | 41 |
| Carga del Código | 43 |
| Variables Reservadas del Arduino | 45 |
| Constantes Predefinidas | 45 |
| Variables de Tiempo | 46 |
| Nombres de Funciones Predefinidos | 47 |
| Variables de Estado del Sistema | 48 |
| Modificadores y Tipos de Datos | 48 |
| Variables de Interrupción y Funciones de Estado | 49 |
| Estructuras de Control Avanzadas en Arduino | 50 |
| Ejemplo 1 – Hola Mundo, Puerto Serie | 52 |
| Ejemplo 2 – Delay | 55 |
| Ejemplo 3 – Variables en Arduino | 57 |
| Ejemplo 4 – Lectura de datos número | 61 |
| Ejemplo 5 – Lectura de datos texto | 62 |
| Ejemplo 6 – Condicionales Booleanos | 62 |

| | |
|---|----|
| Ejemplo 7 – Operaciones Lógicas | 63 |
| Ejemplo 8 – True y False | 65 |
| Ejemplo 9 – Manipulación de variables | 65 |
| Ejemplo 10 – Uso de Variables Lógicas (Booleanas) | 67 |
| Ejemplo 11 – Operaciones con Flotantes | 68 |
| Ejemplo 12 - Uso de delay() para Temporizadores | 69 |

CAPÍTULO III.

Entradas y salidas digitales

| | |
|---|----|
| Introducción a las Entradas y Salidas Digitales en Arduino | 70 |
| Ejemplo 13 – Encender y apagar Led | 72 |
| Ejemplo 14 – Leer estado de un pulsador | 74 |
| Ejemplo 15 – Encender led con pulsador | 75 |
| Ejemplo 16 – Efecto Rebote | 77 |
| Ejemplo 17 – Controlar 4 leds con 4 pulsadores | 80 |
| Ejemplo 18 - Encender LEDs en secuencia con un pulsador...83 | |
| Ejemplo 19 - Controlar un LED con dos pulsadores (Encender y Apagar) | 85 |
| Ejemplo 20 - Secuencia de LEDs con un Pulsador | 87 |
| Ejemplo 21 - Pulsador para cambiar el estado de un LED (Interruptor ON/OFF) | 90 |
| Ejemplo 22 - LED Parpadeante Controlado por un Pulsador....92 | |

CAPÍTULO IV.

Entradas y Salidas Analógicas (PWM)

| | |
|--|-----|
| Concepto básico | 96 |
| Ejemplo 22 – Leer un potenciómetro | 98 |
| PWM (Modulación por Ancho de Pulso) en Arduino | 100 |
| Ejemplo 23 - Cambiar el brillo de un LED según la posición del potenciómetro | 102 |
| Ejemplo 24 - Control de múltiples LEDs con un solo potenciómetro | 105 |

| | |
|--|-----|
| Ejemplo 25 - Control de un LED RGB con un potenciómetro | 107 |
| Ejemplo 26 - Control de la velocidad de un motor (simulado con el brillo de un LED) | 111 |
| Ejemplo 27 - Medición y visualización del valor del potenciómetro en el Monitor Serial | 113 |
| Ejemplo 28 – Medidor de voltaje | 116 |
| Ejemplo 29 – Semáforo controlado por potenciómetro | 118 |

CAPÍTULO V.

Sensores y actuadores

| | |
|--|-----|
| Ejemplo 30 – Sensor de temperatura LM35 | 123 |
| Ejemplo 31 – Sensor de luz (Fotoresistor) LDR | 125 |
| Ejemplo 32 - Sensor de Movimiento PIR (HC-SR501) | 127 |
| Librerías en Arduino | 130 |
| Ejemplo 33 – Sensor de humedad y temperatura DHT22 | 134 |
| Ejemplo 34 – Sensor ultrasónico HC-SR04 sin librería | 137 |
| Ejemplo 35 – Sensor ultrasónico HC-SR04 con librería | 141 |
| Ejemplo 36 – Teclado numérico Keypad | 143 |

CAPÍTULO VI.

Control de motores DC

| | |
|---|-----|
| Ejemplo 37 – Control de motor de 5V | 148 |
| Arduino Motor Shield | 150 |
| Ejemplo 38 – Control de motor DC con Arduino Shield | 153 |
| Ejemplo 39 – Control de motor DC con L298N | 155 |
| Ejemplo 40 – Servomotor | 160 |
| Ejemplo 41 – Servomotor con potenciómetro | 163 |
| Ejemplo 42 – Motor a pasos | 167 |
| Referencias bibliográficas | 174 |
| AUTORES | 176 |



PRÓLOGO

En la actualidad, la programación y la electrónica se han consolidado como habilidades esenciales en un mundo impulsado por la tecnología. La plataforma Arduino, con su enfoque de código abierto y su accesibilidad, ha transformado la manera en que estudiantes, educadores y profesionales abordan el aprendizaje y la aplicación de estas disciplinas. A través de sus diversas aplicaciones, Arduino ha facilitado el prototipado rápido y ha promovido la innovación en múltiples campos, desde la robótica hasta la automatización del hogar.

Este libro tiene como objetivo servir como una guía integral para quienes deseen explorar el entorno de Arduino, proporcionando una base sólida en los conceptos fundamentales de la programación. A lo largo de sus capítulos, se abordarán desde los principios básicos hasta proyectos más avanzados que involucren sensores, actuadores y control de motores.

La estructura del contenido ha sido cuidadosamente diseñada para facilitar la comprensión y el aprendizaje progresivo. Cada capítulo se enfoca en un aspecto específico de la plataforma Arduino, ofreciendo ejemplos prácticos que ilustran los conceptos teóricos y permiten al lector aplicar el conocimiento adquirido de forma inmediata. Este enfoque práctico no solo refuerza el aprendizaje, sino que también fomenta la creatividad y la capacidad de resolución de problemas.

En un contexto donde la tecnología avanza a pasos agigantados, este libro busca dotar a los lectores de las herramientas necesarias para enfrentar los desafíos contemporáneos en el ámbito de la programación y la electrónica. La comprensión profunda de estas disciplinas no solo es un activo valioso en el mundo laboral, sino que también capacita a los individuos para contribuir de manera significativa a la innovación y al desarrollo tecnológico.

INTRODUCCIÓN

En la actualidad, el acceso a la tecnología y la electrónica ha transformado la forma en que interactuamos con el mundo que nos rodea. Arduino, una plataforma de hardware y software de código abierto, ha democratizado la creación de dispositivos electrónicos, permitiendo que tanto principiantes como expertos desarrollen proyectos innovadores de manera accesible y sencilla. Este libro se ha concebido como una guía integral para aquellos que desean aprender sobre programación en Arduino, desde sus fundamentos hasta aplicaciones más complejas, con un enfoque práctico que fomenta la experimentación y la creatividad.

Arduino no es solo una herramienta; es un ecosistema que ha inspirado a millones de entusiastas, educadores y profesionales a explorar el fascinante mundo de la electrónica. Desde su lanzamiento en 2005, ha crecido exponencialmente en popularidad y versatilidad, convirtiéndose en la elección preferida para la educación, el prototipado rápido y el desarrollo de proyectos personalizados. Con una comunidad global activa y una vasta cantidad de recursos disponibles, Arduino permite a los usuarios aprender de forma colaborativa y compartir sus experiencias.

Este libro se estructura en varios capítulos que cubren diferentes aspectos de la programación en Arduino. Comenzamos con una introducción a la plataforma, explorando qué es Arduino, sus tipos y componentes principales, así como los entornos de desarrollo y simulación. Los siguientes capítulos abordan los fundamentos de la programación en Arduino, donde los lectores aprenderán a interactuar con entradas y salidas digitales y analógicas, utilizando una variedad de sensores y actuadores.

Además, se incluirán ejemplos prácticos que permitirán a los lectores aplicar sus conocimientos en proyectos reales.

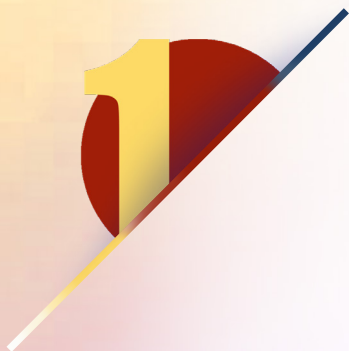
A medida que avanzamos, profundizaremos en el control de motores DC y servomotores, cruciales para la creación de sistemas automatizados. Este enfoque no solo busca desarrollar habilidades técnicas, sino también inspirar la innovación y la resolución de problemas a través de la tecnología.

La combinación de teoría y práctica es esencial para un aprendizaje efectivo. Por ello, cada capítulo incluirá ejemplos detallados, junto con explicaciones claras y concisas que faciliten la comprensión de los conceptos. Los lectores no solo dominarán la programación en Arduino, sino que también estarán equipados con las habilidades necesarias para crear sus propios proyectos y contribuir a la creciente comunidad de makers y desarrolladores.

Este libro es una invitación a explorar, experimentar y descubrir el apasionante mundo de la programación en Arduino. Ya sea un estudiante, un educador o un entusiasta de la tecnología, encontrará en estas páginas las herramientas y conocimientos necesarios para dar vida a sus ideas y transformar su entorno a través de la electrónica.

CAPÍTULO I.

Introducción a Arduino



¿Qué es Arduino?

En la última década, Arduino ha emergido como una de las plataformas más populares para proyectos de electrónica y programación. Su diseño de hardware abierto y su gran comunidad han permitido a usuarios de todos los niveles innovar con facilidad. Arduino combina una placa electrónica, un microcontrolador y un entorno de desarrollo integrado (IDE) que facilita la programación de dispositivos electrónicos.

Existen varios modelos de Arduino, como el Uno, Mega, Nano y Leonardo, cada uno con características específicas para distintas aplicaciones. Por ejemplo, el popular Arduino Uno incluye un microcontrolador ATmega328P, pines de entrada/salida digital y analógicos, un puerto USB, un regulador de voltaje y un botón de reinicio.

La instalación del IDE de Arduino es esencial para comenzar a programar, asegurando la comunicación entre la placa y el ordenador. También existen simuladores

virtuales como Tinkercad y Wokwi que permiten aprender y experimentar sin hardware físico. Estas plataformas son ideales para la educación y el desarrollo de habilidades técnicas de manera accesible.

Como señala Lemos (2021), *“Arduino ha democratizado la programación y la electrónica, brindando a millones de personas la oportunidad de aprender y crear”* (p. 25), una democratización vital en un mundo digital y conectado.

Arduino es una plataforma de hardware libre basada en una sencilla placa con un microcontrolador, junto con un entorno de desarrollo que facilita la creación de software para la misma. Fue creada en 2005 por un grupo de desarrolladores liderado por Massimo Banzi y David Cuartielles en el Instituto de Diseño Interactivo de Ivrea, Italia, con el objetivo de proporcionar una herramienta accesible para estudiantes, ingenieros y aficionados para la creación de proyectos de electrónica (Banzi & Shiloh, 2022).

El proyecto Arduino se caracteriza por su código abierto, lo que permite a cualquier persona descargar los esquemas y modificar el software para personalizar sus proyectos. Su popularidad se debe a la sencillez de uso del lenguaje de programación basado en Wiring y a la comunidad global que ofrece soporte a los usuarios. Los proyectos con Arduino abarcan desde la robótica hasta sistemas de automatización, control de dispositivos y proyectos artísticos interactivos (Monk, 2022).

Tipos de Arduino

Arduino ofrece una amplia gama de placas diseñadas para abordar distintos tipos de proyectos, desde simples prototipos de aprendizaje hasta aplicaciones industriales y profesionales. Cada placa tiene características que permiten su uso en una variedad de escenarios, lo que convierte a Arduino en una plataforma flexible y versátil para la enseñanza y el desarrollo de proyectos electrónicos.

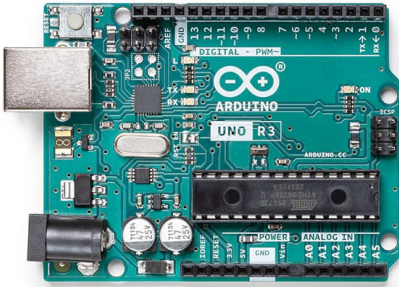


Figura 1. Arduino Uno.

Fuente: Arduino (2024).

La **Arduino Uno** es la placa más popular y utilizada, especialmente entre principiantes. Está basada en el microcontrolador **ATmega328P** y tiene características que la hacen ideal para proyectos básicos de electrónica y prototipos.

- **Microcontrolador:** ATmega328P.
- **Voltaje de operación:** 5V.
- **Entradas/salidas digitales:** 14 (6 de ellas PWM).
- **Entradas analógicas:** 6.
- **Memoria Flash:** 32 KB.
- **Frecuencia del reloj:** 16 MHz.
- **Conectividad:** USB Tipo B para cargar programas.

La Uno es ampliamente utilizada en proyectos educativos, debido a su sencillez y amplia documentación (Banzi & Shiloh, 2022).

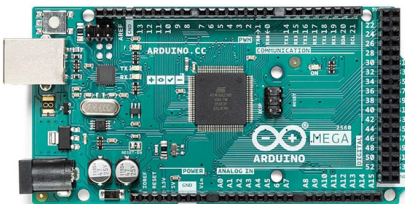


Figura 2. Arduino Mega 2560.

Fuente: Arduino (2024).

La **Arduino Mega 2560** es una placa diseñada para proyectos más complejos que requieren un mayor número de pines de entrada y salida o más memoria para almacenar programas.

- **Microcontrolador:** ATmega2560.
- **Voltaje de operación:** 5V.
- **Entradas/salidas digitales:** 54 (15 de ellas PWM).
- **Entradas analógicas:** 16.
- **Memoria Flash:** 256 KB.
- **Frecuencia del reloj:** 16 MHz.
- **Conectividad:** USB Tipo B y puerto ICSP para programación.

Es muy utilizada en proyectos que requieren controlar múltiples dispositivos, como sistemas de automatización doméstica o robótica avanzada (Arduino, 2023a).

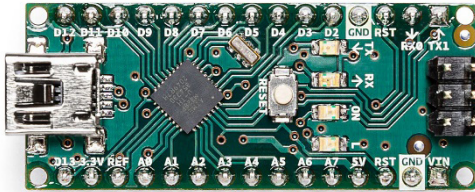


Figura 3. Arduino Nano.

Fuente: Arduino (2024).

La **Arduino Nano** es una versión compacta de la Uno, diseñada para proyectos que requieren ahorrar espacio, como dispositivos portátiles o proyectos de Internet de las Cosas (IoT).

- **Microcontrolador:** ATmega328 (o ATmega168 en versiones más antiguas).
- **Voltaje de operación:** 5V.
- **Entradas/salidas digitales:** 14 (6 de ellas PWM).

- **Entradas analógicas:** 8.
- **Memoria Flash:** 32 KB.
- **Frecuencia del reloj:** 16 MHz.
- **Conectividad:** Mini USB para cargar programas.

Por su tamaño y funcionalidad, es ideal para proyectos de circuitos embebidos y dispositivos pequeños (Monk, 2022).

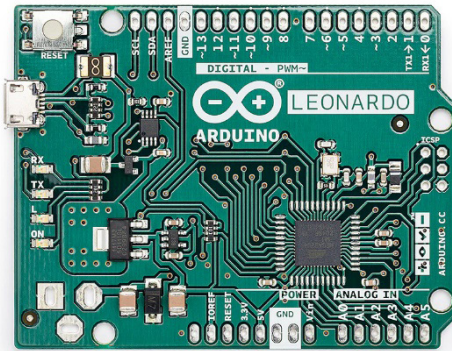


Figura 4. Arduino Leonardo.

Fuente: Arduino (2024).

La **Arduino Leonardo** se distingue de otras placas por la capacidad de actuar como un dispositivo USB, como un teclado o un ratón, sin necesidad de hardware adicional. Utiliza el microcontrolador ATmega32U4, que incluye la capacidad de comunicación USB nativa.

- **Microcontrolador:** ATmega32U4.
- **Voltaje de operación:** 5V.
- **Entradas/salidas digitales:** 20 (7 de ellas PWM).
- **Entradas analógicas:** 12.
- **Memoria Flash:** 32 KB.
- **Frecuencia del reloj:** 16 MHz.

- **Conectividad:** Micro USB para comunicación y alimentación. Es especialmente útil para proyectos que requieren interacción con computadoras u otros dispositivos USB (Arduino, 2023b).

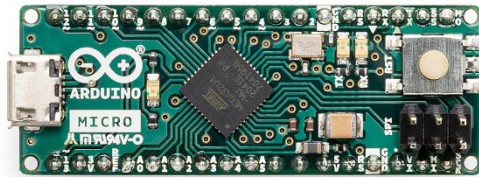


Figura 5. Arduino Micro.

Fuente: Arduino (2024).

La **Arduino Micro** es una placa compacta basada en el ATmega32U4, similar a la Arduino Leonardo, con la capacidad de actuar como un dispositivo USB nativo (como un teclado o ratón). Esta placa es ideal para proyectos embebidos y dispositivos portátiles que requieren la funcionalidad de un teclado o un mouse.

- **Microcontrolador:** ATmega32U4.
- **Voltaje de operación:** 5V.
- **Entradas/salidas digitales:** 20 (7 de ellas PWM).
- **Entradas analógicas:** 12.
- **Memoria Flash:** 32 KB.
- **Frecuencia del reloj:** 16 MHz.
- **Conectividad:** Micro USB.

Es adecuada para proyectos que requieran portabilidad y dispositivos USB personalizados (Arduino, 2023g)

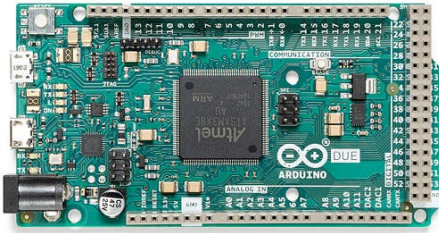


Figura 6. Arduino Due.

Fuente: Arduino (2024).

La **Arduino Due** es la primera placa Arduino que utiliza un microcontrolador de 32 bits, el **ARM Cortex-M3**, lo que la hace significativamente más potente en comparación con las placas de 8 bits como la Arduino Uno. Está diseñada para proyectos que requieren mayor capacidad de procesamiento y más memoria, como el control avanzado de robots, simulaciones en tiempo real y sistemas embebidos más complejos.

Especificaciones:

- **Microcontrolador:** AT91SAM3X8E (ARM Cortex-M3 a 32 bits).
- **Voltaje de operación:** 3.3V.
- **Entradas/salidas digitales:** 54 (12 de ellas PWM).
- **Entradas analógicas:** 12 (resolución de 12 bits).
- **Salidas analógicas (DAC):** 2 (resolución de 12 bits).
- **Memoria Flash:** 512 KB.
- **SRAM:** 96 KB.
- **Frecuencia del reloj:** 84 MHz.
- **Conectividad:** Micro USB, puerto JTAG, ICSP.
- **Corriente máxima por pin de E/S:** 130 mA.

El Arduino Due es una placa poderosa, adecuada para proyectos avanzados que requieren mayor capacidad de procesamiento y

entradas/salidas que las ofrecidas por las placas más comunes de Arduino. Sin embargo, es importante tener en cuenta las diferencias en el voltaje y compatibilidad al trabajar con shields y componentes externos.

Componentes principales de Arduino UNO

Este libro utiliza el Arduino Uno para sus ejemplos, al ser el más fácil y el más común ya que fue el primero en aparecer. La plataforma Arduino se compone de varios elementos clave que permiten el desarrollo de proyectos electrónicos:

- **Microcontrolador:** El corazón de cada placa Arduino es un microcontrolador (por ejemplo, ATmega328 en el caso de la Arduino Uno). Este microcontrolador es el encargado de ejecutar el código que se programa en la placa.
- **Pines digitales y analógicos:** Los pines digitales se utilizan para leer o enviar señales binarias (alto/bajo, encendido/apagado), mientras que los pines analógicos permiten leer variaciones de voltaje en sensores. Por ejemplo, la placa Arduino Uno tiene 14 pines digitales y 6 pines analógicos (Arduino, 2023).
- **Conexiones y alimentación:** Las placas Arduino pueden ser alimentadas a través de un puerto USB o una fuente de alimentación externa. Dependiendo del proyecto, es posible usar baterías o adaptadores de corriente para alimentar la placa.

Cada modelo de placa Arduino tiene diferentes configuraciones en cuanto a número de pines, capacidad de procesamiento y memoria, lo que permite adaptar el uso de cada placa a las necesidades del proyecto (Arduino, 2023).

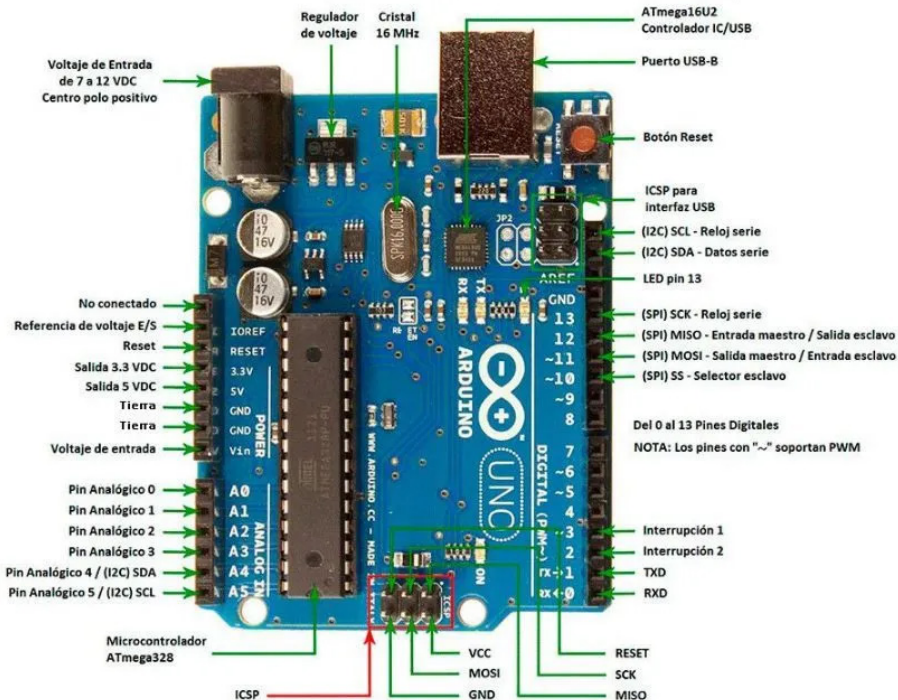


Figura 7. Pines y componentes del Arduino Uno.

Fuente: Arduino (2024).

La figura 7 muestra todos los pines disponibles en el Arduino con su respectiva descripción. Es importante recalcar que el Arduino Uno es el más común, sin embargo, los ejemplos de este libro pueden ser usados en cualquier otra versión del Arduino.

Instalación del IDE de Arduino

Para programar cualquier placa Arduino, es necesario instalar el IDE (Entorno de Desarrollo Integrado, por sus siglas en inglés). Este software permite escribir el código, compilarlo y cargarlo en la placa mediante un cable USB. A continuación, se presentan los pasos detallados para la instalación:

- 1. Descarga del IDE:** El IDE se puede descargar gratuitamente desde el sitio oficial de Arduino (<https://www.arduino.cc/en/software>) para diferentes sistemas operativos (Windows, macOS, y Linux).

Downloads



 **Arduino IDE 2.3.2**

The new major release of the Arduino IDE is faster and even more powerful! In addition to a more modern editor and a more responsive interface it features autocompletion, code navigation, and even a live debugger.

For more details, please refer to the [Arduino IDE 2.0 documentation](#).

Nightly builds with the latest bugfixes are available through the section below.

SOURCE CODE
The Arduino IDE 2.0 is open source and its source code is hosted on [GitHub](#).

DOWNLOAD OPTIONS

- Windows** Win 10 and newer, 64 bits
- Windows** MSI installer
- Windows** ZIP file
- Linux** AppImage 64 bits (X86-64)
- Linux** ZIP file 64 bits (X86-64)
- macOS** Intel, 10.15: "Catalina" or newer, 64 bits
- macOS** Apple Silicon, 11: "Big Sur" or newer, 64 bits

[Release Notes](#)

Figura 8. Versiones de descarga para el Arduino IDE.

Fuente: Arduino (2024).

Arduino está disponible en varias plataformas, es importante asegurarse de descargar la versión correcta para cada computadora.

- 2. Instalación del software:** Tras la descarga, el instalador guiará al usuario a través de un proceso sencillo, que incluye la instalación de controladores para la comunicación entre la computadora y la placa Arduino.

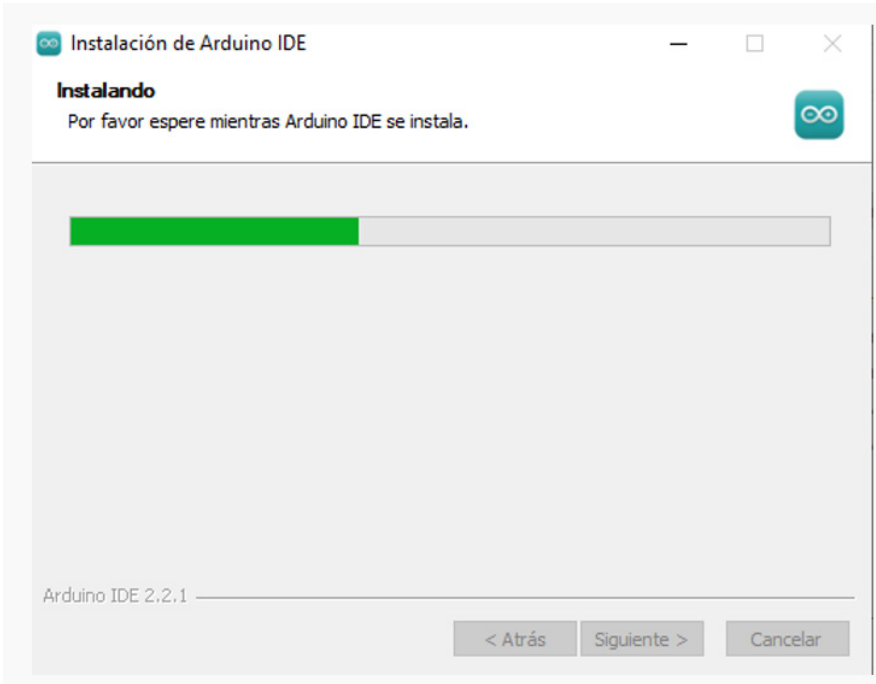


Figura 9. Proceso de instalación del software Arduino IDE.

El proceso de instalación es rápido y sencillo, solo hay que hacer clic en siguiente hasta finalizar el proceso.

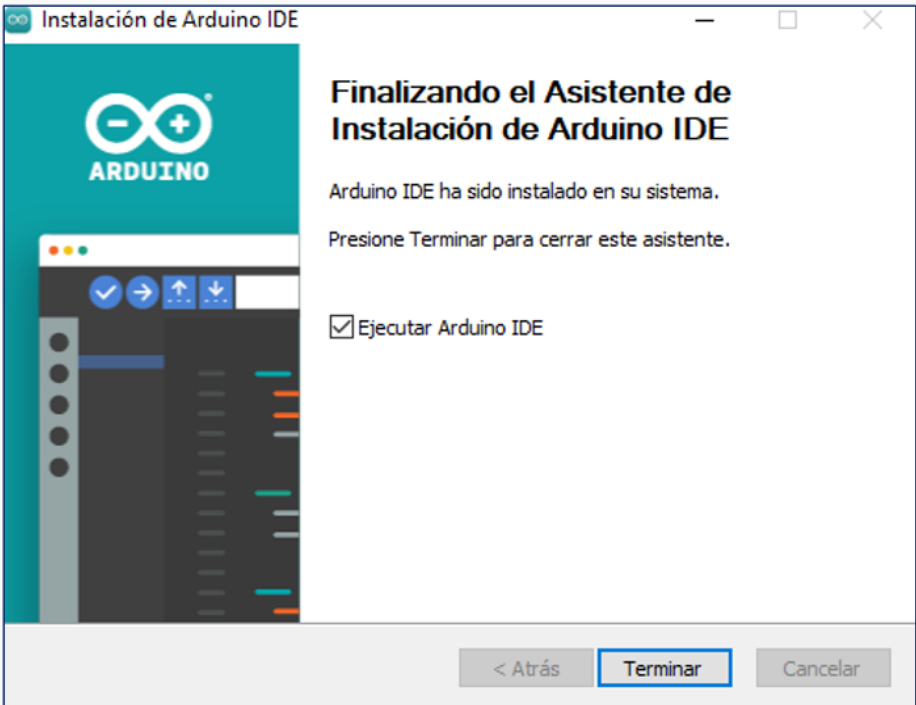


Figura 10. Ventana de finalización de instalación del Arduino.

Al finalizar la instalación se hace clic en terminar y el software se ejecutará.

3. Configuración inicial: Una vez instalado, se debe seleccionar el modelo de placa que se utilizará (por ejemplo, Arduino Uno) y el puerto de conexión al que está conectada la placa. Esto se hace en la pestaña “Herramientas” del IDE.

El IDE de Arduino se basa en Processing, un entorno popular para artistas y diseñadores. Utiliza un lenguaje derivado de Wiring que facilita la programación para usuarios no familiarizados con la programación tradicional de microcontroladores (Purdum, 2020).

Introducción a las plataformas de simulación

Para los principiantes en Arduino, las plataformas de simulación son herramientas útiles que permiten desarrollar y probar proyectos sin necesidad de hardware físico. Dos de las plataformas más populares para simular proyectos de Arduino son Tinkercad y Wokwi.

Guía para usar Tinkercad con Arduino Uno

Desarrollada por Autodesk, Tinkercad es una plataforma en línea que permite diseñar circuitos electrónicos y simular proyectos con Arduino. Es ideal para principiantes, ya que ofrece una interfaz gráfica intuitiva y una amplia base de componentes preconfigurados, como LEDs, sensores, resistencias y motores. Tinkercad no solo permite diseñar el circuito, sino también escribir y cargar el código en la simulación (Autodesk, 2023).

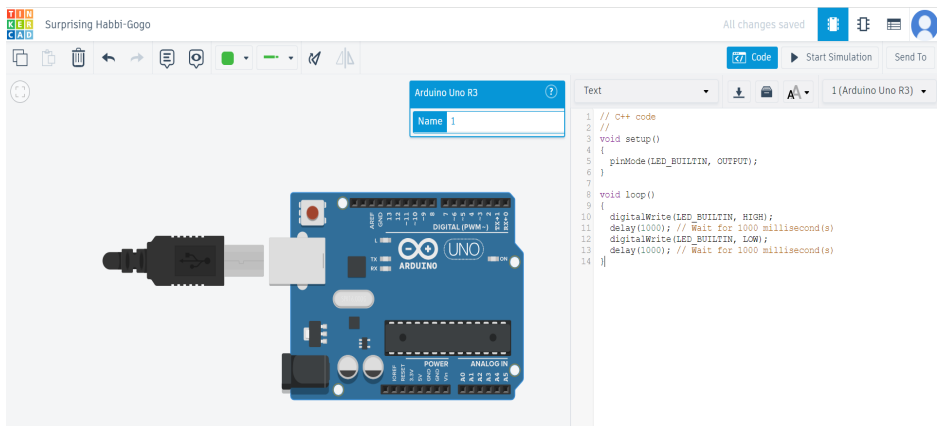


Figura 11. Entorno de simulación de Tinkercad.

La figura 11 muestra el entorno de desarrollo para realizar simulaciones en Tinkercad. Para iniciar el proceso a continuación se describen los pasos a seguir.

1. Crear una cuenta en Tinkercad

- **Visita la página web oficial:** Abra el navegador y vea la página oficial de Tinkercad: <https://www.tinkercad.com>.
- **Registrarse/Iniciar sesión:** Si no tiene una cuenta, haga clic en “**Join Now**” para registrarse. Puede usar su correo electrónico, cuenta de Google o Facebook para hacerlo. Si ya tiene cuenta, simplemente inicie sesión.

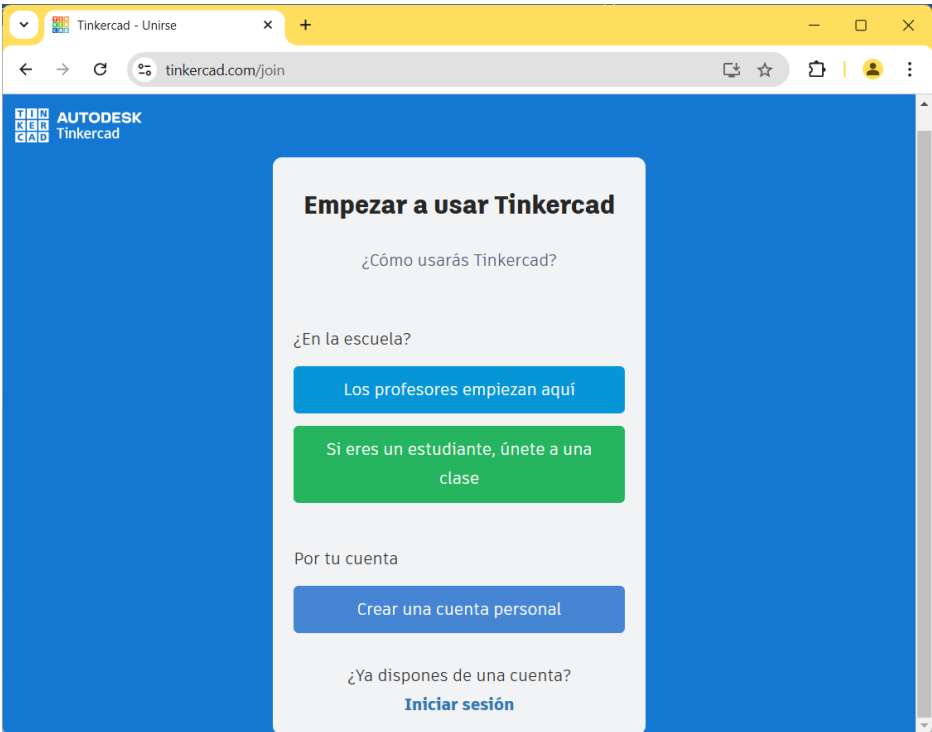


Figura 12. Sitio web para registro de cuenta en Autodesk.

2. Crear un Nuevo Proyecto de Circuitos

- **Ir a “Circuits”:** Una vez dentro de Tinkercad, en la página principal, busque la opción “Circuits” en el menú superior.

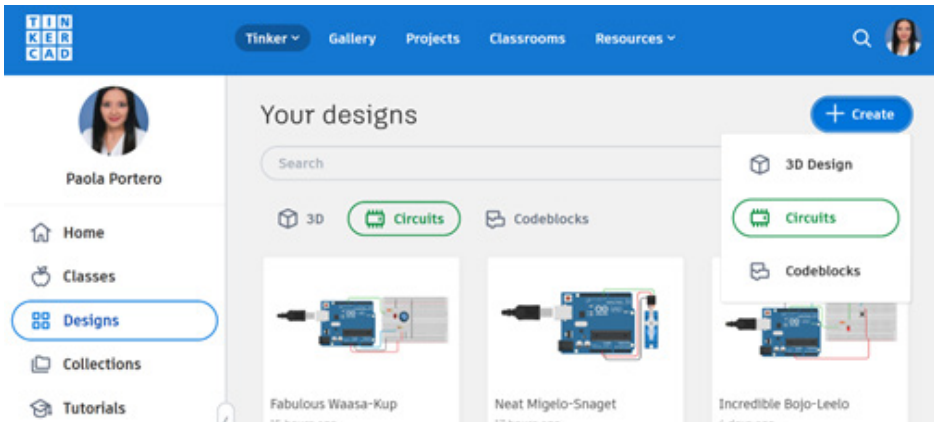


Figura 13. Pantalla de inicio de Tinkercad.

- Esto le llevará a la sección donde puede diseñar y simular circuitos electrónicos.
- **Crear un nuevo circuito:** Haga clic en el botón “**Create new Circuit**” para empezar un nuevo proyecto de simulación de circuitos.

3. Añadir el Arduino Uno

- **Seleccionar el Arduino Uno:** En la barra lateral derecha, verá una lista de componentes electrónicos. Use el campo de búsqueda y escriba “Arduino Uno”. Cuando aparezca, haga clic sobre él para agregarlo a su espacio de trabajo.
- **Colocar el Arduino Uno:** Una vez añadido, aparecerá el Arduino Uno en el área de trabajo. Puede arrastrarlo para colocarlo en la posición que desee.

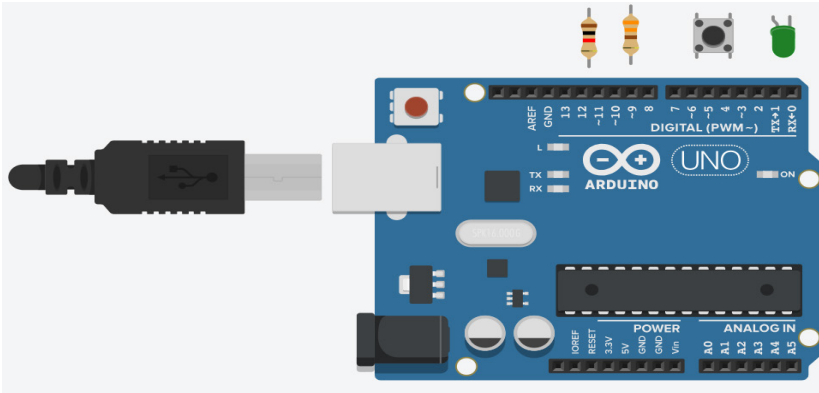


Figura 14. Arduino Uno en Tinkercad.

4. Añadir Otros Componentes

- **Agregar componentes electrónicos:** Dependiendo del proyecto que quiera simular, puede añadir diferentes componentes como LEDs, resistencias, botones, servomotores, etc.

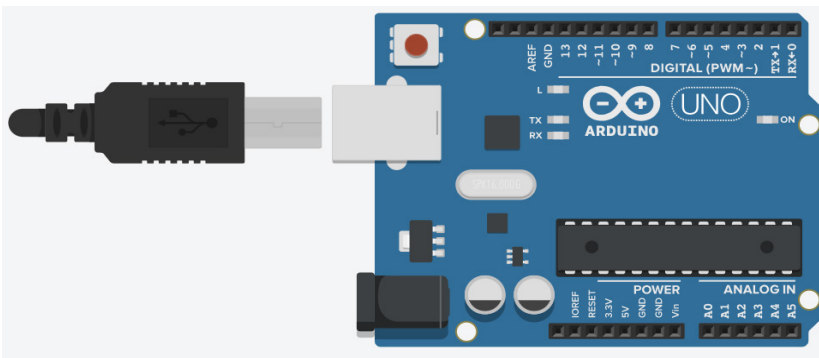


Figura 15. Elementos en pantalla.

- Al igual que el Arduino Uno, busque los componentes en la barra lateral. Puede arrastrar y soltar los componentes en el área de trabajo.

5. Conectar los Componentes

- **Conexión de los cables:** Para conectar los componentes en el circuito, haga clic en uno de los pines (como el pin digital 13 del Arduino) y arrastre hasta el terminal del componente que desee conectar (como el ánodo del LED).

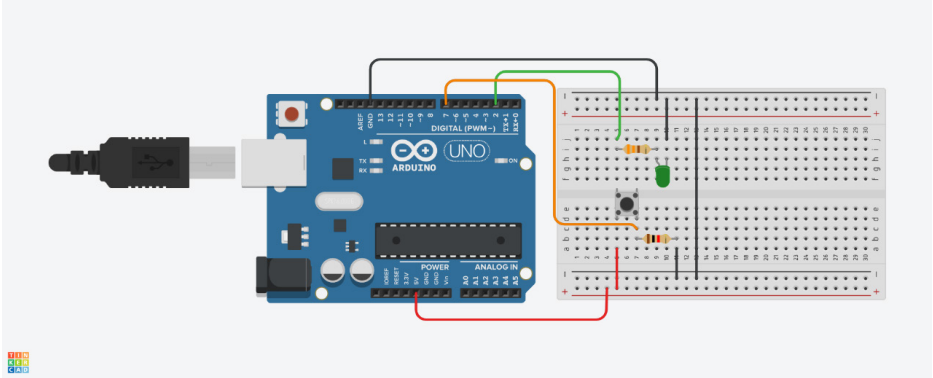


Figura 16. Tinkercad - Implementación de un circuito.

- Se puede cambiar el color del cable haciendo clic sobre él para facilitar la identificación de las conexiones.

6. Programar el Arduino Uno

- **Abrir el editor de código:** Tinkercad te permite programar directamente el Arduino Uno. Para ello, haga clic en la pestaña “Code” en la parte superior derecha del área de trabajo.
- Por defecto, Tinkercad ofrece un editor de bloques de programación (tipo Scratch), pero puede cambiarlo a código textual (basado en el lenguaje de Arduino) haciendo clic en “Blocks” y seleccionando “Text”.

All changes saved



Code

Start Simulation

Send To

Text



1 (Arduino Uno R3)

```
1 int pul = 7;
2 int led = 2;
3 int estado_pul = 0;
4
5 void setup()
6 {
7   pinMode(pul, INPUT);
8   pinMode(led, OUTPUT);
9 }
10 void loop()
11 {
12   estado_pul = digitalRead(pul);
13
14   if(estado_pul == HIGH)
15   {
16     digitalWrite(led, HIGH);
17   }
18   else
19   {
20     digitalWrite(led, LOW);
21   }
22 }
```

Serial Monitor

Figura 17. Tinkercad - Código de programación de Arduino.

- **Compilar el código:** Una vez que haya escrito el código, puede hacer clic en **“Start Simulation”** para comenzar la simulación.

7. Iniciar la Simulación

- **Correr la simulación:** Haga clic en **“Start Simulation”** para ver el comportamiento del circuito que ha diseñado.

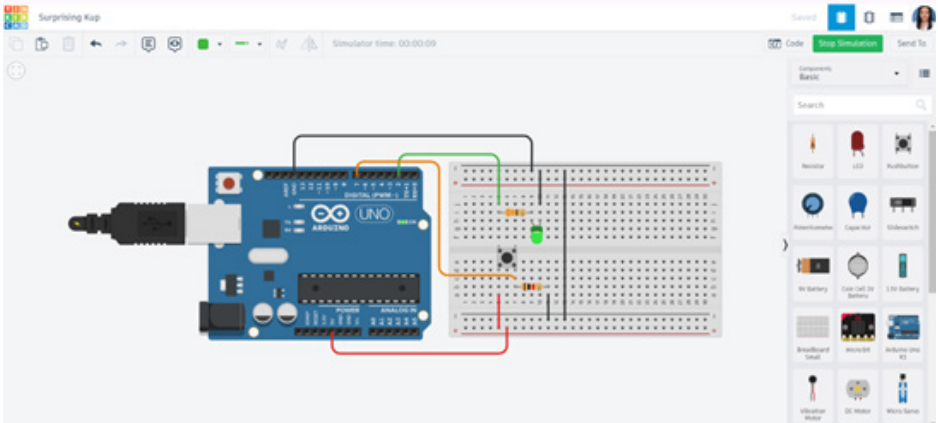


Figura 18. Tinkercad - Funcionamiento del circuito.

- **Verificación y corrección:** Si la simulación no funciona como espera, puede detenerla, revisar las conexiones o el código y hacer los ajustes necesarios.

8. Guardar el Proyecto

- **Guardar su circuito:** Para guardar el proyecto, haga clic en el botón **“Save”** en la parte superior de la pantalla. Tinkercad guarda sus proyectos en la nube, y puede acceder a ellos en cualquier momento desde su cuenta.
- **Cambiar el nombre del proyecto:** Puede cambiar el nombre del proyecto haciendo clic en el título predeterminado (por ejemplo, **“Untitled Circuit”**) en la parte superior de la pantalla.

9. Compartir el Proyecto

- **Compartir su simulación:** Tinkercad permite compartir sus proyectos a través de un enlace. Para hacerlo, diríjase a la parte superior derecha y haga clic en “**Share**”. Podrá generar un enlace para compartir con otros.

Características Adicionales de Tinkercad

- **Simulación de Sensores y Actuadores:** Tinkercad permite simular diversos sensores (como sensores de temperatura, ultrasonido, etc.) y actuadores (motores, pantallas LCD, etc.), lo que lo convierte en una herramienta muy versátil para proyectos de Arduino.
- **Biblioteca de Ejemplos:** Tinkercad también tiene una colección de proyectos y ejemplos predefinidos que puede explorar para aprender más sobre Arduino y otras tecnologías.

Guía para usar Wokwi con Arduino Uno

Wokwi es otra plataforma de simulación para Arduino, enfocada en proyectos más avanzados. A diferencia de Tinkercad, Wokwi permite simular componentes más complejos como pantallas OLED, matrices de LEDs y múltiples tipos de sensores. Además, ofrece soporte para simular diferentes placas Arduino (como la Mega y la Nano) y permite la integración de otros microcontroladores como el ESP32 (Wokwi, 2023).

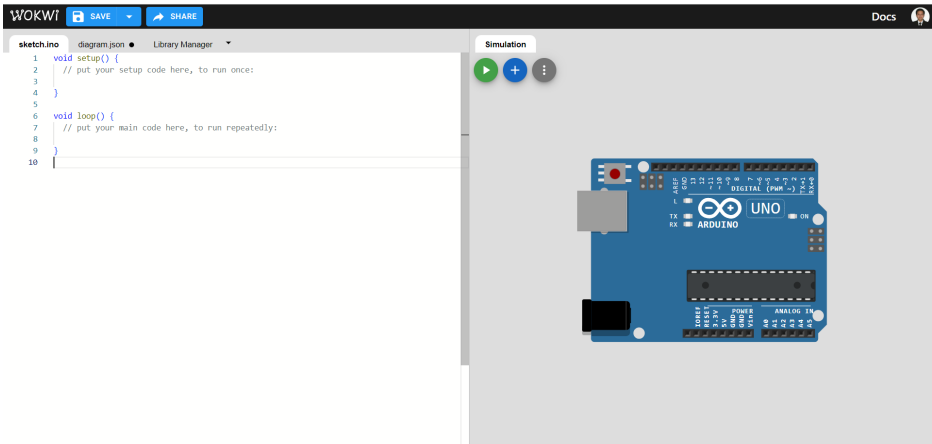


Figura 19. Entorno de simulación de Wokwi.

La figura 19 muestra el entorno de desarrollo para realizar simulaciones en Wokwi. Para iniciar el proceso a continuación se describen los pasos a seguir.

1. Acceder a Wokwi

- **Visite la página web oficial:** Abra el navegador web y diríjase a la página oficial de Wokwi: <https://wokwi.com>.
- **Registrarse/Iniciar sesión (opcional):** No es necesario crear una cuenta para empezar a usar la plataforma, pero si desea guardar sus proyectos, es recomendable registrarse o iniciar sesión.

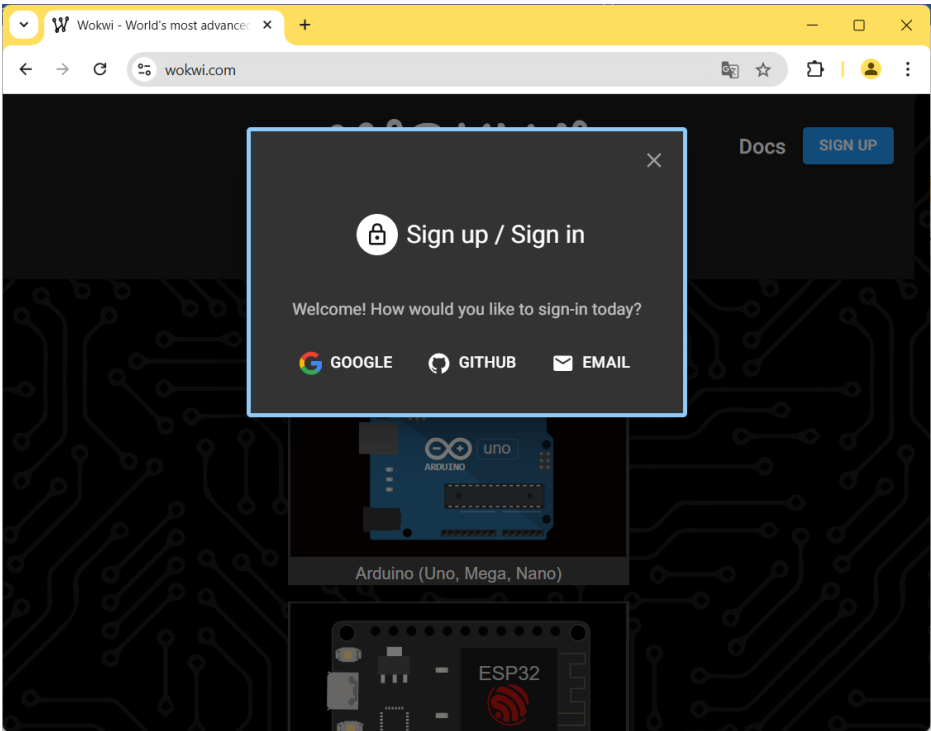


Figura 20. Inicio de sesión en Wokwi.

2. Crear un Nuevo Proyecto

- **Hacer clic en “Start a New Project”**: Una vez en la página principal, encontrará un botón que dice **“Start a New Project”** o **“Create New Project”**. Haga clic para comenzar.

3. Seleccionar Arduino Uno

- **Elegir la placa Arduino Uno**: Al iniciar el nuevo proyecto, le pedirá seleccionar una placa. En este caso, seleccione Arduino Uno.

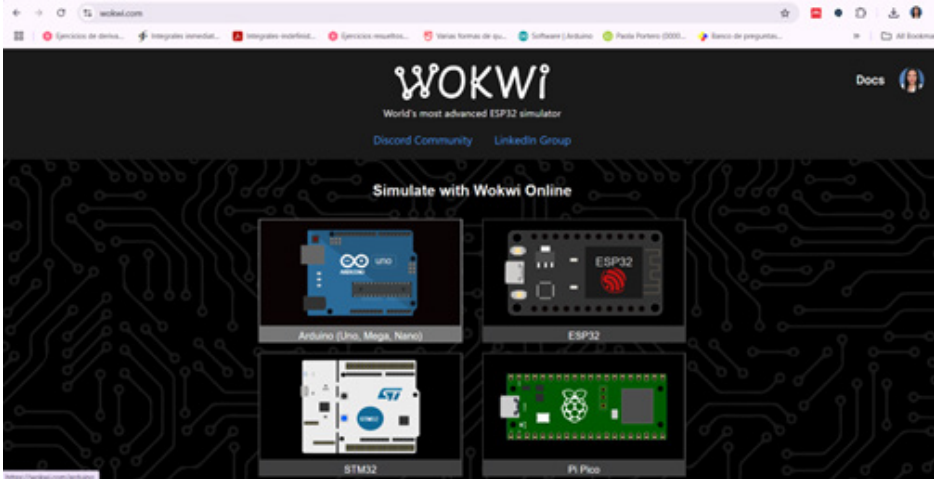


Figura 21. Pantalla de inicio Wokwi – Arduino UNO.

- Puede hacerlo desde la lista de microcontroladores disponibles o usando el buscador en la parte superior de la pantalla.

4. Configuración Inicial del Proyecto

- **Añadir componentes:** La plataforma permite agregar varios componentes como LEDs, resistencias, servos, pantallas, y más.

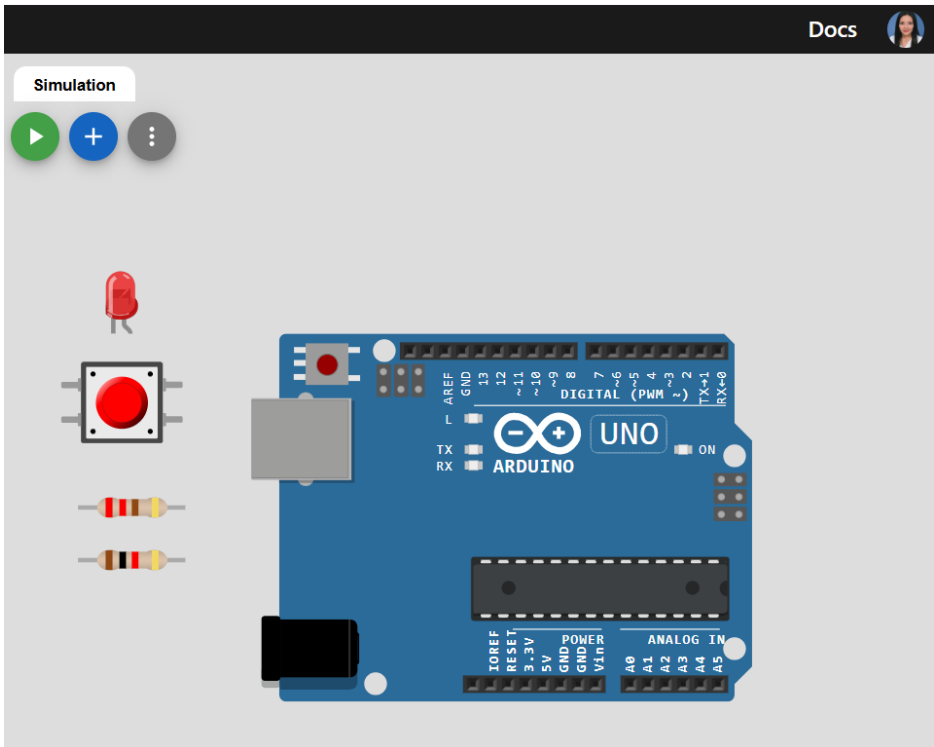


Figura 22. Wokwi – Elementos en pantalla.

- En el menú del lado izquierdo, puede buscar los componentes que necesita, como **LEDs, resistencias, motores, etc.**
- Para añadir un componente, simplemente haga clic en él y arrástrelo al espacio de trabajo.

5. Conectar los Componentes

- **Interconectar componentes:** Wokwi permite conectar los componentes de forma visual.

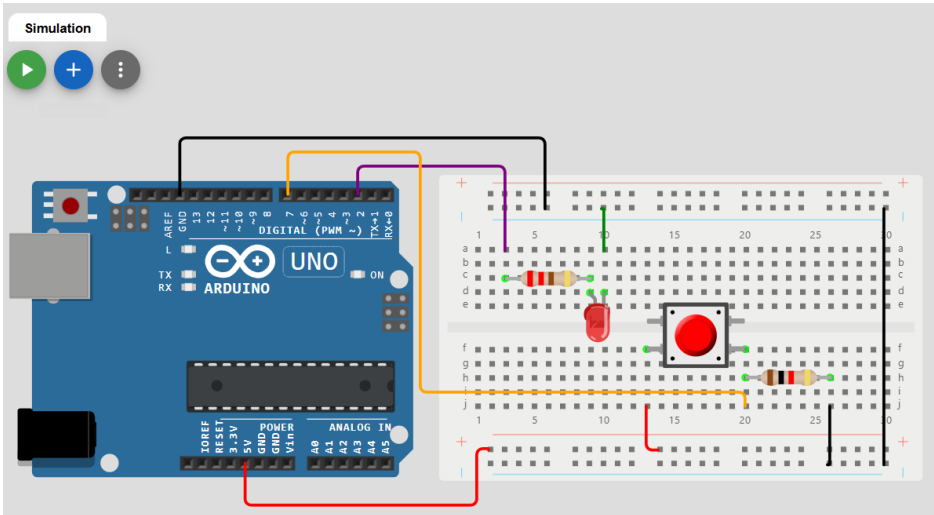
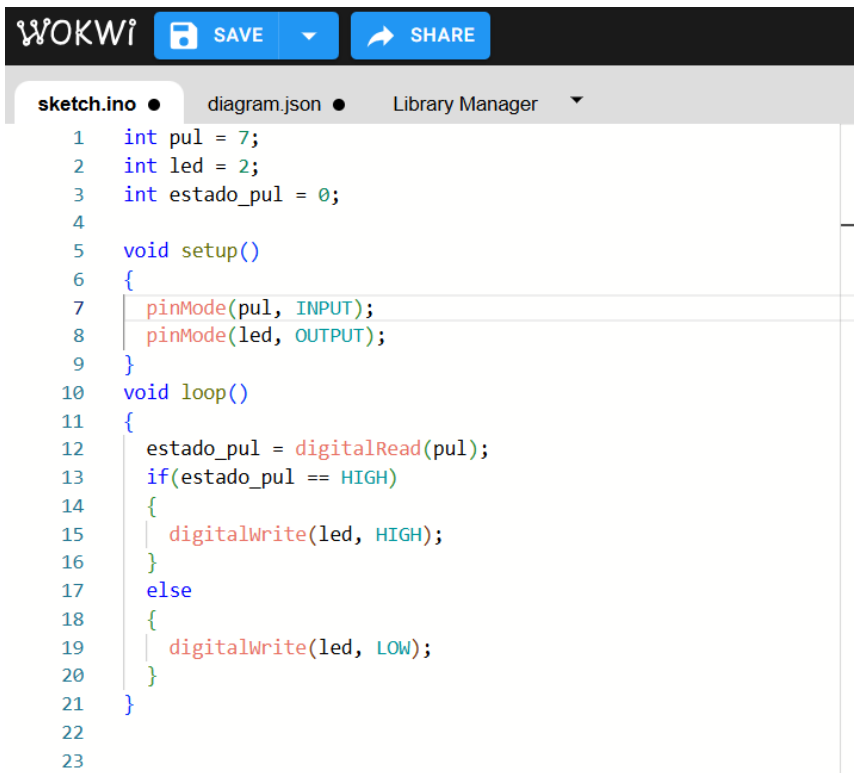


Figura 23. Wokwi – Implementación de un circuito.

- Para conectar un componente al **Arduino Uno**, haga clic en uno de los terminales del componente y arrastre el cable hasta el pin correspondiente del Arduino.
- Puede cambiar el color del cable haciendo clic sobre él para una mejor organización del circuito.

6. Escribir el Código

- **Abrir el editor de código:** A la derecha del área de trabajo, verá una pestaña llamada “**Code**” donde puede escribir el código para su Arduino Uno.



```
1  int pul = 7;
2  int led = 2;
3  int estado_pul = 0;
4
5  void setup()
6  {
7    pinMode(pul, INPUT);
8    pinMode(led, OUTPUT);
9  }
10 void loop()
11 {
12   estado_pul = digitalRead(pul);
13   if(estado_pul == HIGH)
14   {
15     digitalWrite(led, HIGH);
16   }
17   else
18   {
19     digitalWrite(led, LOW);
20   }
21 }
22
23
```

Figura 24. Wokwi – Código de programación.

- La plataforma usa el mismo lenguaje de programación que el IDE de Arduino.
- **Compilar y Simular:** Una vez que escriba el código, puede hacer clic en el botón “**Start Simulation**” o “**Play**” en la parte superior de la pantalla para comenzar a simular el comportamiento del circuito.

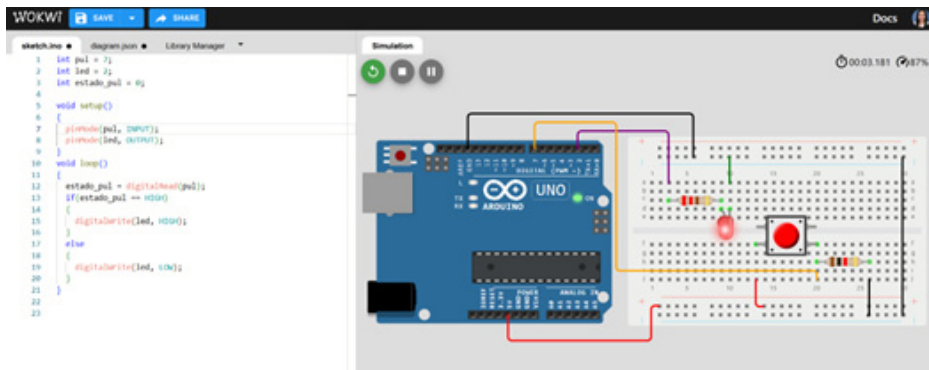


Figura 25. Wokwi – Funcionamiento de la simulación.

7. Depuración y Visualización de Resultados

- **Verificar el funcionamiento del circuito:** Observa cómo se comportan los componentes en la simulación.
- **Depurar el código:** Si hay algún problema, Wokwi te notificará los errores en el código o en las conexiones. Puede detener la simulación, corregir el código y volver a iniciar la simulación.

8. Guardar el Proyecto

- **Guardar su proyecto:** Si tiene una cuenta en Wokwi, puede guardar su proyecto haciendo clic en el botón **“Save”** en la parte superior.
 - Si no tiene cuenta, también puede exportar el proyecto como un archivo JSON o código en formato Arduino.

9. Compartir su Proyecto

- **Compartir enlaces:** Wokwi permite compartir sus proyectos fácilmente a través de un enlace. Simplemente haz clic en **“Share”** y genera un enlace que puede compartir con otros.

Características Adicionales de Wokwi

- **Bibliotecas Integradas:** Wokwi permite el uso de bibliotecas de Arduino como en el IDE oficial. Puede incluirla en tu código usando `#include`.
- **Simulación de Sensores y Actuadores:** Puede agregar componentes como sensores de temperatura, pantallas LCD, motores, etc., y simular cómo interactúan con tu código en tiempo real.
- **Compatibilidad con Otras Placas:** Wokwi también soporta otras placas como Arduino Mega, ESP32, y ESP8266 para proyectos más avanzados.

Consejos para Usar Wokwi

- **Documentación:** Wokwi tiene una sección de documentación y tutoriales que pueden ayudar a entender mejor cómo funciona cada componente y cómo conectarlos.
- **Práctica:** La plataforma es ideal para practicar sin tener acceso a hardware físico, lo que la convierte en una excelente herramienta para estudiantes.

Ambas plataformas ofrecen un entorno accesible para experimentar con Arduino sin la necesidad de hardware, lo que resulta ideal para entornos educativos y laboratorios virtuales.

CAPÍTULO II.

Fundamentos de Programación en Arduino



Interfaz del Arduino

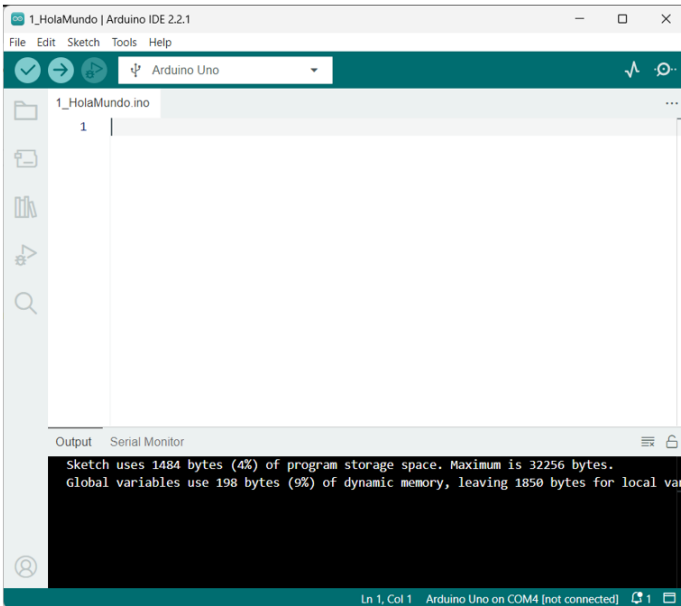
A lo largo de este capítulo, se explorarán varios ejemplos prácticos que ilustran los conceptos fundamentales de programación en Arduino. Comenzaremos con el clásico “Hola Mundo” utilizando el puerto serie, donde los usuarios aprenderán a comunicar información básica desde su placa. Posteriormente, se abordarán conceptos como el uso de la función `delay()` para temporización, la declaración y manipulación de variables, así como la lectura de datos tanto numéricos como de texto. También se presentarán ejemplos que incorporan condicionales booleanos, operaciones lógicas y el uso de variables lógicas (booleanas). Finalmente, se abordarán las operaciones con flotantes y la importancia de utilizar la función `delay()` en aplicaciones de temporización.

Estos ejemplos proporcionarán a los usuarios las habilidades

necesarias para comenzar a desarrollar proyectos más complejos en el futuro. Como señala Pahlavan (2022), *“la práctica y la experimentación son fundamentales para el aprendizaje efectivo en el mundo de la programación”* (p. 13). Al practicar estos conceptos y ejemplos, los lectores estarán mejor equipados para afrontar los desafíos que se presenten en su viaje por el mundo de Arduino.

El dominio de la programación en Arduino se basa en la comprensión de los fundamentos del entorno de desarrollo integrado (IDE) de Arduino y su interfaz de usuario, así como en la práctica de ejemplos básicos que ilustran los conceptos clave. El Arduino IDE se presenta como una herramienta intuitiva y amigable que permite a los usuarios escribir, cargar y depurar su código de manera efectiva. La interfaz, que incluye un editor de texto para el código, una ventana de salida para mensajes del sistema y un menú de opciones, está diseñada para simplificar el proceso de programación, lo que permite a los usuarios concentrarse en la creación de sus proyectos sin distracciones innecesarias.

Cargar y compilar el código es un proceso esencial para cualquier desarrollador que trabaje con Arduino. El IDE facilita esta tarea mediante un solo clic en el botón de carga, lo que permite transferir el código desde el ordenador a la placa de Arduino. Durante este proceso, el IDE compila el código, transformándolo en un formato que el microcontrolador puede interpretar y ejecutar. Este flujo de trabajo simplificado es uno de los aspectos que hace de Arduino una plataforma tan accesible para principiantes y expertos por igual, permitiendo a los usuarios experimentar y aprender a través de la práctica.



Barra de Menú
Accesos Directos
Área de pestañas

Área de Edición

Consola de salida

Barra de información

Figura 26. Interfaz de usuario del Arduino IDE.

La figura 26 muestra el entorno de desarrollo del Arduino IDE.

Carga del Código

1. Conectar la Placa Arduino al Computador

- Conecte su placa Arduino (por ejemplo, Arduino Uno) al puerto USB de su computadora utilizando un cable USB.
- Asegúrese que la placa esté bien conectada y el LED de encendido de la placa esté encendido.

2. Abrir el Arduino IDE

- Inicie el Arduino IDE en su computadora.
- Si aún no tiene el código abierto, diríjase a Archivo > Nuevo para crear un nuevo “sketch” o abra un archivo existente desde Archivo > Abrir.

3. Seleccionar la Placa Arduino Correcta

- Vaya a Herramientas > Placa y seleccione el modelo de su placa Arduino. Si está usando un Arduino Uno, seleccione Arduino Uno. Si es otro modelo, elija el adecuado (por ejemplo, Arduino Nano, Arduino Mega, etc.).

4. Seleccionar el Puerto COM

- Vaya a Herramientas > Puerto y seleccione el puerto COM al que está conectada la placa Arduino. El puerto suele tener un nombre como COM3, COM4 en Windows o /dev/ttyUSB0 en Linux/macOS.
- Si no sabe cuál es, pruebe desconectando el Arduino, verificando qué puertos aparecen y luego conectando el Arduino nuevamente para ver cuál es el nuevo puerto disponible.

5. Verificar el Código

- Antes de cargar el código, puede verificar si hay errores de sintaxis. Haga clic en el botón de “Verificar” (el ícono con una marca de verificación en la barra de herramientas).
- Esto compilará el código sin cargarlo en la placa, y mostrará si hay errores que deba corregir.

6. Cargar el Código en la Placa

- Haga clic en el botón “Cargar” (ícono con una flecha hacia la derecha) en la barra de herramientas o use el atajo de teclado Ctrl + U (o Cmd + U en macOS).
- El Arduino IDE compilará su código y lo cargará en la placa.
- Si todo está bien, verá en la consola de salida mensajes indicando que la carga ha sido exitosa.

7. Verificación en el Monitor Serial (Opcional)

- Si su programa utiliza el Monitor Serial (como en el ejemplo anterior), abra el Monitor Serial desde el menú Herramientas > Monitor Serie o presionando Ctrl + Shift + M (o Cmd + Shift + M en macOS).
- Asegúrese de que la velocidad de baudios en el Monitor Serial coincida con la configuración del código (por ejemplo, 9600 baudios).

Consejos adicionales:

- Si tiene problemas para cargar el código, asegúrese de que el puerto y la placa seleccionada sean correctos.
- Verifique que ningún otro programa esté usando el puerto COM que utiliza su Arduino.
- Si hay errores de compilación, revise cuidadosamente el código para corregir los posibles problemas antes de intentar cargarlo nuevamente.

Variables Reservadas del Arduino

En Arduino, las **variables reservadas** son aquellas que tienen un significado especial dentro del lenguaje de programación o en el entorno de desarrollo de Arduino. Estas variables no pueden ser utilizadas como identificadores (nombres de variables, funciones o clases) porque ya están asignadas para otros propósitos. A continuación, se explica las variables reservadas más importantes en Arduino, organizadas por categorías:

Constantes Predefinidas

Estas constantes se utilizan comúnmente para simplificar el código, brindando nombres fáciles de recordar en lugar de usar valores numéricos.

HIGH: Representa un valor lógico “alto” o 1 (5V en la mayoría de las placas Arduino). Se usa principalmente para encender una salida digital o leer un valor alto en una entrada digital.

```
digitalWrite(13, HIGH); // Enciende el LED en el pin 13
```

LOW: Representa un valor lógico “bajo” o 0 (0V). Se utiliza para apagar una salida digital o leer un valor bajo en una entrada digital.

```
digitalWrite(13, LOW); // Apaga el LED en el pin 13
```

INPUT: Se usa para configurar un pin como entrada digital.

```
pinMode(2, INPUT); // Configura el pin 2 como entrada
```

OUTPUT: Se usa para configurar un pin como salida digital.

```
pinMode(13, OUTPUT); // Configura el pin 13 como salida
```

INPUT_PULLUP: Configura el pin como entrada digital con una resistencia pull-up interna.

```
pinMode(2, INPUT_PULLUP); // Configura el pin 2 como entrada con resistencia pull-up interna
```

LED_BUILTIN: Esta constante hace referencia al número del pin al que está conectado el LED integrado en la placa (generalmente el pin 13).

```
digitalWrite(LED_BUILTIN, HIGH); // Enciende el LED integrado
```

Variables de Tiempo

millis(): Devuelve el número de milisegundos desde que la placa Arduino comenzó a ejecutar el programa. Se utiliza para medir intervalos de tiempo sin detener la ejecución del programa.

```
unsigned long tiempo = millis(); // Guarda el tiempo actual en milisegundos
```

micros(): Devuelve el número de microsegundos desde que comenzó a ejecutarse el programa. Es útil para medir tiempos más precisos que los proporcionados por millis().

```
unsigned long tiempoMicrosegundos = micros();
```

delay(): Pausa la ejecución del programa por un número específico de milisegundos.

```
delay(1000); // Pausa el programa por 1 segundo
```

delayMicroseconds(): Pausa la ejecución del programa por un número específico de microsegundos.

```
delayMicroseconds(10); // Pausa el programa por 10 microsegundos
```

Nombres de Funciones Predefinidos

Estas funciones son parte del ciclo de vida básico de un programa de Arduino. Deben estar presentes en cualquier sketch.

setup(): Se ejecuta una vez al inicio del programa. Aquí se colocan las configuraciones iniciales, como la configuración de pines.

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

loop(): Se ejecuta continuamente después de que setup() haya terminado. Aquí se colocan las acciones que se deben repetir indefinidamente.

```
void loop() {
```

```
digitalWrite(13, HIGH);  
delay(1000);  
digitalWrite(13, LOW);  
delay(1000);  
}
```

Variables de Estado del Sistema

true: Representa el valor booleano verdadero (equivalente a 1).

```
if (true) {  
    // Siempre se ejecuta  
}
```

false: Representa el valor booleano falso (equivalente a 0).

```
if (false) {  
    // Nunca se ejecuta  
}
```

Modificadores y Tipos de Datos

Estos modificadores están reservados para definir tipos de datos con especificaciones precisas.

int: Representa un número entero (16 bits, -32,768 a 32,767 en la mayoría de las placas Arduino).

```
int x = 100; // Declara una variable entera
```


long: Es un entero de mayor tamaño (32 bits, -2,147,483,648 a 2,147,483,647).

```
long y = 100000L;
```

float: Representa un número en coma flotante (32 bits, para números decimales).

```
float z = 3.14;
```

boolean: Representa un valor booleano (true o false).

```
boolean flag = true;
```

const: Declara una variable como constante (no puede cambiar su valor).

```
const int pin = 13;
```

byte: Representa un número de 8 bits sin signo (de 0 a 255).

```
byte counter = 255;
```

Variables de Interrupción y Funciones de Estado

attachInterrupt(): Se usa para asociar una función de interrupción a un pin específico.

```
attachInterrupt(digitalPinToInterrupt(pin), ISR, mode);
```

detachInterrupt(): Desactiva una interrupción previamente habilitada.

```
detachInterrupt(digitalPinToInterrupt(pin));
```

interrupts(): Habilita las interrupciones en el programa.

```
interrupts();
```

noInterrupts(): Desactiva las interrupciones.

```
noInterrupts();
```

Estructuras de Control Avanzadas en Arduino

Las estructuras de control en Arduino permiten que el programa tome decisiones y repita acciones de acuerdo con condiciones específicas. A continuación, se explica en detalle las estructuras de control más avanzadas: if, if/else, for, while, y do-while. También se incluyen ejemplos prácticos aplicados al control de dispositivos electrónicos, como LEDs y sensores.

Estructura de Control if

La estructura if permite ejecutar un bloque de código solo si una condición es verdadera.

Sintaxis:

```
if (condición) {  
    // Código a ejecutar si la condición es verdadera  
}
```

Estructura de Control if/else

El if/else permite ejecutar un bloque de código si una condición es verdadera, y otro bloque si es falsa.

Sintaxis:

```
if (condición) {  
    // Código a ejecutar si la condición es verdadera  
} else {  
    // Código a ejecutar si la condición es falsa  
}
```

Estructura de Control for

La estructura for se usa para ejecutar un bloque de código un número específico de veces. Es útil cuando se sabe cuántas veces se debe repetir una acción.

```
for (inicialización; condición; incremento) {  
    // Código a ejecutar  
}
```

Estructura de Control while

La estructura while repite un bloque de código mientras una condición sea verdadera. Se usa cuando no se sabe cuántas veces se ejecutará el ciclo, pero se tiene una condición de salida.

```
while (condición) {  
    // Código a ejecutar mientras la condición sea  
    verdadera  
}
```

Estructura de Control do-while

El do-while es similar a while, pero garantiza que el bloque de código se ejecute al menos una vez, incluso si la condición es falsa desde el principio.

Sintaxis:

```
do {  
    // Código a ejecutar  
} while (condición);
```

Switch, case:

Permiten la ejecución de diferentes bloques de código en función de una variable.

```
switch (variable) {  
  
  case 1:  
  
    // Haz algo  
  
    break;  
  
  case 2:  
  
    // Haz otra cosa  
  
    break;  
  
}
```

Estas estructuras de control proporcionan un gran poder para controlar dispositivos electrónicos de manera eficiente en proyectos con Arduino. Con ellas, es posible tomar decisiones basadas en condiciones, repetir acciones de forma controlada y realizar tareas más complejas, como el control de múltiples dispositivos, lectura de sensores, y más.

Ejemplo 1 – Hola Mundo, Puerto Serie

El **puerto serial** de Arduino es una interfaz de comunicación que permite el intercambio de datos entre la placa Arduino y otros dispositivos, como computadoras, módulos de comunicación (como Bluetooth o Wi-Fi), y otros microcontroladores. La comunicación serial se basa en el envío y recepción de datos en forma de bits, uno tras otro, a través de un solo canal.

Funcionamiento del Puerto Serial

- **Transmisión y Recepción:** Arduino tiene pines específicos designados para la transmisión (TX) y recepción (RX) de datos. Por ejemplo, en la placa Arduino Uno, el pin TX es el pin 1 y el pin RX es el pin 0.
- **Comunicación Asincrónica:** El puerto serial en Arduino utiliza un protocolo de comunicación asincrónico, lo que significa que no necesita un reloj compartido entre los dispositivos que se comunican. En cambio, cada dispositivo debe saber a qué velocidad está transmitiendo el otro.

Inicialización del Puerto Serial a 9600 Baudios

En el código de Arduino, la comunicación serial se inicializa con `Serial.begin(9600);`, donde el número 9600 representa la **tasa de baudios** (baud rate). Esta tasa indica cuántos bits se transmiten por segundo.

- **9600 Baudios:** Es una de las tasas de baudios más comunes y se utiliza a menudo porque es lo suficientemente rápida para la mayoría de las aplicaciones de comunicación básica, al mismo tiempo que proporciona una buena compatibilidad con diversos dispositivos.

¿Por Qué Usar 9600?

1. **Compatibilidad:** Muchos dispositivos y programas (como el Monitor Serial de Arduino) utilizan 9600 baudios por defecto, lo que facilita la comunicación.
2. **Estabilidad:** Esta velocidad es estable y permite la transmisión de datos sin errores en la mayoría de las situaciones.

¿Qué Sucede si se Cambia la Tasa de Baudios?

Si decides utilizar una tasa de baudios diferente, es importante que la velocidad de transmisión sea la misma en ambos extremos de la comunicación (es decir, tanto en el Arduino como en el dispositivo receptor, como el Monitor Serial). Algunas tasas comunes son 4800, 9600, 14400, 19200, 38400, 57600, y 115200.

1. Si se Aumenta la Velocidad (por ejemplo, 115200):

- **Ventajas:** Permite transmitir datos más rápidamente, lo que puede ser útil para aplicaciones que requieren un alto volumen de datos.
- **Desventajas:** Puede ser más susceptible a errores de transmisión si el cableado o las conexiones no son de buena calidad. No todos los dispositivos son compatibles con tasas de baudios más altas.

2. Si se Disminuye la Velocidad (por ejemplo, 4800):

- **Ventajas:** Menos propenso a errores de transmisión, especialmente en distancias largas o con cables de menor calidad.
- **Desventajas:** La transmisión de datos es más lenta, lo que puede no ser adecuado para aplicaciones que requieren alta velocidad.

```
// Configuración inicial
void setup() {
  // Iniciar la comunicación serial a 9600 baudios
  Serial.begin(9600);
  // Enviar el mensaje "Hola Mundo" al Monitor
  Serial.println("Hola Mundo");
}

// Bucle principal
void loop() {
  // No se necesita hacer nada en el bucle principal
}
```

Ejemplo 2 – Delay

El comando `delay()` en Arduino es una función utilizada para pausar la ejecución del programa durante un tiempo específico. Aquí tienes una explicación detallada de cómo funciona:

¿Qué es `delay()`?

- La función `delay(ms)` detiene la ejecución del código durante el número de milisegundos especificado por el argumento `ms`.
- Por ejemplo, `delay(1000)`; hará que el programa se detenga durante 1 segundo (1000 milisegundos).

Funcionamiento

1. Interrupción del Flujo de Programa:

- Cuando se llama a `delay()`, el microcontrolador deja de ejecutar cualquier otra instrucción durante el período especificado. Durante este tiempo, no se pueden realizar otras tareas, incluyendo la lectura de entradas, la escritura en salidas, o cualquier otra operación en el programa.

2. Precisión:

- La función utiliza un temporizador interno para contar el tiempo. Aunque es bastante precisa, no es adecuada para aplicaciones que requieren un tiempo de espera extremadamente exacto, ya que puede haber pequeñas variaciones dependiendo de otros procesos en el microcontrolador.

3. Limitaciones:

- Mientras `delay()` está en efecto, el Arduino no puede responder a eventos, como cambios en entradas digitales o analógicas. Esto puede ser un problema en aplicaciones donde se necesita una respuesta rápida a las entradas.

4. Alternativa:

- Para mantener el Arduino responsivo mientras se espera, se puede usar la función `millis()` junto con una estructura de control. `millis()` devuelve el tiempo transcurrido desde que el Arduino comenzó a ejecutar el programa en milisegundos, lo que permite crear temporizadores sin bloquear el flujo del programa.


```
// Configuración inicial
void setup() {
  // Iniciar la comunicación serial a 9600 baudios
  Serial.begin(9600);
  // Enviar el mensaje "Hola Mundo" al Monitor Serial
  Serial.println("Aquí inicia el setup");
}

// Bucle principal
void loop() {
  // No se necesita hacer nada en el bucle principal
  Serial.println("Aquí inicia el loop");
  delay(1000);
}
```

Ejemplo 3 – Variables en Arduino

Las variables en Arduino son elementos fundamentales en la programación que permiten almacenar y manipular datos durante la ejecución de un programa. Aquí tienes una explicación detallada sobre las variables en el entorno de Arduino:

¿Qué son las Variables?

- **Definición:** Una variable es un espacio de almacenamiento en la memoria del microcontrolador que tiene un nombre simbólico (identificador) y un tipo de dato asociado. Este espacio se puede utilizar para guardar valores que pueden cambiar durante la ejecución del programa.

Tipos de Variables en Arduino

Arduino utiliza varios tipos de variables, que se pueden clasificar según su tamaño y el tipo de datos que almacenan:

1. Variables Enteras (int):

- Almacenan números enteros, sin decimales.
- Tamaño: 2 bytes (16 bits), puede almacenar valores de -32,768 a 32,767.
- Ejemplo:

```
int contador = 0; // Inicializa una variable entera
```

Variables de Punto Flotante (float):

- Almacenan números reales (decimales).
- Tamaño: 4 bytes (32 bits), permite almacenar valores decimales más precisos.
- Ejemplo:

```
float temperatura = 25.5; // Inicializa una variable de punto flotante
```

Variables de Carácter (char):

- Almacenan un solo carácter.
- Tamaño: 1 byte (8 bits).
- Ejemplo

```
char letra = 'A'; // Inicializa una variable de carácter
```

Variables Lógicas (boolean):

- Almacenan valores de verdad: true (verdadero) o false (falso).

- Tamaño: 1 byte (8 bits), aunque solo utiliza 1 bit para representar el valor.
- Ejemplo

```
bool encendido = false; // Inicializa una variable booleana
```

Variables Sin Firmas (unsigned):

- Se pueden usar con enteros (unsigned int, unsigned char, etc.) para almacenar solo números positivos.
- Ejemplo

```
unsigned int edad = 30; // Inicializa una variable entera sin signo
```

Alcance de las Variables

- **Variables Globales:** Se declaran fuera de cualquier función y están disponibles en todo el programa. Ejemplo:

```
int globalVariable = 10; // Variable global
```

- **Variables Locales:** Se declaran dentro de una función y solo son accesibles dentro de esa función. Ejemplo:

```
void miFuncion() {  
    int localVariable = 5; // Variable local  
}
```

Inicialización de Variables

Es una buena práctica inicializar las variables antes de usarlas para evitar comportamientos indeseados. Por ejemplo:

```
int contador = 0; // Inicialización
```

Modificación de Variables

Las variables pueden cambiar su valor durante la ejecución del programa. Se pueden modificar mediante operaciones aritméticas, entradas del usuario o como resultado de otras funciones. Ejemplo:

```
contador++; // Incrementa el valor de contador en 1
```

```
void setup() {  
  // Iniciar la comunicación serial  
  Serial.begin(9600);  
  
  // Definir los números a sumar  
  int num1 = 5;  
  int num2 = 4;  
  
  // Calcular la suma  
  int suma = num1 + num2;  
  
  // Mostrar el resultado en el Monitor Serial  
  Serial.print("La suma de ");  
  Serial.print(num1);  
  Serial.print(" y ");  
  Serial.print(num2);  
  Serial.print(" es: ");
```

```
    Serial.println(suma);
}

void loop() {
    // No hay nada que hacer en el bucle principal
}
```

Ejemplo 4 – Lectura de datos número

```
void setup() {
    Serial.begin(9600);
    Serial.println("Ingresa un numero:");
}

void loop() {
    if (Serial.available() > 0) {
        int numero = Serial.parseInt(); // Lee un número entero
        Serial.print("Has ingresado el numero: ");
        Serial.println(numero);
        delay(1000);
    }
}
```

Ejemplo 5 – Lectura de datos texto

```
void setup() {
  Serial.begin(9600);
  Serial.println("Ingresa un texto:");
}

void loop() {
  if (Serial.available() > 0) {
    String entrada = Serial.readStringUntil('\n'); //
    Lee una línea de texto
    Serial.print("Has ingresado: ");
    Serial.println(entrada);
    delay(1000);
  }
}
```

Ejemplo 6 – Condicionales Booleanos

Definición de Booleanos:

- En Arduino, como en muchos otros lenguajes de programación, el tipo de dato **booleano** (bool) es un tipo que solo puede tener uno de dos valores: **true (verdadero)** o **false (falso)**.
- Estos valores son utilizados para representar condiciones o estados que pueden ser evaluados como verdaderos o falsos.

Uso de true y false:

- true: Representa una condición que es correcta o que se cumple. En otras palabras, significa que una afirmación es válida.

- `false`: Representa una condición que no es correcta o que no se cumple. Esto indica que una afirmación no es válida.

Ejemplos de Uso:

- **Condicionales:** En estructuras de control como `if`, los booleanos se utilizan para tomar decisiones en el código. Por ejemplo:

```
bool luzEncendida = true; // La luz está encendida

if (luzEncendida) {
  Serial.println("La luz está encendida.");
} else {
  Serial.println("La luz está apagada.");
}
```

- En este caso, el mensaje "La luz está encendida." se imprime porque `luz Encendida` es `true`.

Ejemplo 7 – Operaciones Lógicas

Operaciones Lógicas:

- Los booleanos son fundamentales en las operaciones lógicas. Se pueden combinar usando operadores como:
 - `&&` (AND): Verdadero solo si ambas condiciones son verdaderas.

- `||` (OR): Verdadero si al menos una de las condiciones es verdadera.
- `!` (NOT): Invierte el valor booleano (`true` se convierte en `false` y viceversa).
- Por ejemplo:

```
bool sensorActivado = false;
bool sistemaSeguridad = true;

if (sensorActivado || sistemaSeguridad) {
    Serial.println("Sistema de seguridad activado.");
} else {
    Serial.println("Sistema de seguridad desactivado.");
}
```

Interpretación Numérica:

- Internamente, en Arduino (y en muchos lenguajes de programación), los valores booleanos son representados como números:
 - `true` se representa como 1.
 - `false` se representa como 0.
- Esto significa que se pueden utilizar en expresiones matemáticas, pero es recomendable mantener su uso dentro del contexto lógico para evitar confusiones.

Ejemplo 8 – True y False

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

  bool luzEncendida = false; // La luz está inicialmente apagada

  // Cambiamos el estado de la luz
  luzEncendida = true; // Encendemos la luz

  // Usamos un condicional para mostrar el estado de la luz
  if (luzEncendida) {
    Serial.println("La luz está encendida.");
  } else {
    Serial.println("La luz está apagada.");
  }
}

void loop() {
  // No se necesita nada en el bucle
}
```

Ejemplo 9 – Manipulación de variables

Este ejemplo enseña a los estudiantes cómo declarar, inicializar y modificar variables. En este caso, se muestra cómo sumar y restar valores.

```
void setup() {
  // Inicializa el monitor serial
  Serial.begin(9600);

  // Declaración e inicialización de variables
  int numero1 = 10;
  int numero2 = 5;
  int resultado;

  // Realiza operaciones con variables
  resultado = numero1 + numero2; // Suma
  Serial.println("Suma: ");
  Serial.println(resultado);

  resultado = numero1 - numero2; // Resta
  Serial.println("Resta: ");
  Serial.println(resultado);

  resultado = numero1 * numero2; // Multiplicación
  Serial.println("Multiplicación: ");
  Serial.println(resultado);

  resultado = numero1 / numero2; // División
  Serial.println("División: ");
  Serial.println(resultado);
}

void loop() {
  // No se necesita hacer nada en loop para este
  ejemplo
}
```

Ejemplo 10 – Uso de Variables Lógicas (Booleanas)

```
void setup() {
  // Inicializa el monitor serial
  Serial.begin(9600);

  // Declaración de una variable booleana
  bool encendido = true;

  // Mostrar el estado inicial
  Serial.println("Estado inicial: ");
  if (encendido) {
    Serial.println("Encendido");
  } else {
    Serial.println("Apagado");
  }

  // Cambiar el estado
  encendido = !encendido; // Cambia a false

  // Mostrar el estado después del cambio
  Serial.println("Estado después del cambio: ");
  if (encendido) {
    Serial.println("Encendido");
  } else {
    Serial.println("Apagado");
  }
}

void loop() {
  // No se necesita hacer nada en loop para este
  // ejemplo
}
```

Ejemplo 11 – Operaciones con Flotantes

En este ejemplo se utilizan variables de punto flotante (float) para manejar números con decimales, por ejemplo, para realizar cálculos de temperatura.

```
void setup() {
  // Inicializa el monitor serial
  Serial.begin(9600);

  // Declaración de variables de punto flotante
  float temperaturaCelsius = 25.3;
  float temperaturaFahrenheit;

  // Conversión de Celsius a Fahrenheit
  temperaturaFahrenheit = (temperaturaCelsius * 9 /
5) + 32;

  // Muestra las temperaturas en el monitor serial
  Serial.print("Temperatura en Celsius: ");
  Serial.println(temperaturaCelsius);
  Serial.print("Temperatura en Fahrenheit: ");
  Serial.println(temperaturaFahrenheit);
}

void loop() {
  // No se necesita hacer nada en loop para este
ejemplo
}
```

Ejemplo 12 - Uso de `delay()` para Temporizadores

```
void setup() {
  // Inicializa el monitor serial
  Serial.begin(9600);

  // Muestra un mensaje
  Serial.println("Iniciando temporizador...");

  // Espera 5 segundos (5000 milisegundos)
  delay(5000);

  // Muestra el mensaje después de la pausa
  Serial.println("Han pasado 5 segundos.");
}

void loop() {
  // No se necesita hacer nada en loop para este
  ejemplo
}
```

CAPÍTULO III.

Entradas y salidas digitales



Introducción a las Entradas y Salidas Digitales en Arduino

El capítulo sobre Entradas y Salidas Digitales en Arduino explica cómo la placa interactúa con el entorno físico, permitiendo recibir señales externas mediante entradas digitales y controlar dispositivos como LEDs y motores mediante salidas digitales. Se introduce la teoría de los pines digitales y la diferencia entre entradas y salidas, con ejemplos prácticos que enseñan a encender y apagar LEDs, leer el estado de un pulsador y manejar el efecto rebote en botones, un fenómeno importante en el diseño de interfaces de usuario.

Los ejemplos avanzan desde controlar múltiples LEDs con pulsadores hasta crear secuencias y alternar el estado de un LED. También se muestra cómo implementar un LED parpadeante controlado por un pulsador, lo que demuestra la versatilidad de las entradas y salidas digitales en diversas aplicaciones. Como afirma Mckinney, M. (2021), “la

comprensión de las entradas y salidas digitales es esencial para cualquier ingeniero o entusiasta de la electrónica que desee llevar sus habilidades al siguiente nivel” (p. 45). Estos ejercicios ayudan a los lectores a desarrollar habilidades necesarias para proyectos interactivos en Arduino.

Concepto de Señales Digitales:

- **Señales Digitales:** Las señales digitales son aquellas que solo tienen dos posibles estados: **encendido (HIGH) o apagado (LOW)**, representados comúnmente por los valores 1 y 0. Son fundamentales en sistemas electrónicos para comunicarse con dispositivos como sensores, actuadores, y otros componentes.
- **Ejemplos:** Un LED es un dispositivo que puede estar encendido o apagado, mientras que un pulsador puede estar presionado o no presionado.

Definición de Entradas y Salidas Digitales:

- **Entradas Digitales:** Son pines del Arduino que reciben información de dispositivos externos. Un ejemplo común es un botón o sensor que envía señales al Arduino.
- **Salidas Digitales:** Son pines del Arduino que envían señales a dispositivos externos para controlarlos, como encender un LED, activar un relé, etc.

Diferencia entre HIGH y LOW:

- **HIGH (1):** Cuando un pin digital está en estado HIGH, significa que está recibiendo o enviando una señal de voltaje alto (5V en el caso del Arduino Uno).
- **LOW (0):** Cuando un pin digital está en estado LOW, significa que está recibiendo o enviando una señal de voltaje bajo (0V o GND).

Pines Digitales en el Arduino:

- **Pines Digitales:** El Arduino tiene varios pines digitales numerados del 0 al 13 (en el caso del Arduino Uno), que pueden configurarse tanto como entradas como salidas.
- **Pin 13:** El pin 13 está conectado internamente a un LED en la placa Arduino, lo que facilita las pruebas básicas sin necesidad de conectar componentes externos.

Función pinMode():

- **Uso de pinMode(pin, mode):** Esta función se utiliza para configurar el modo de un pin digital. Puede ser INPUT o OUTPUT según si el pin será utilizado para leer o enviar señales.
 - pinMode(2, INPUT); — Configura el pin 2 como entrada.
 - pinMode(13, OUTPUT); — Configura el pin 13 como salida.

Aplicaciones Comunes:

- **Entrada:** Leer el estado de un botón para realizar una acción (encender un LED, enviar datos al monitor serial).
- **Salida:** Encender un LED, controlar un motor, activar un relé para dispositivos más grandes.

Ejemplo 13 – Encender y apagar Led

Este código es un programa simple para una placa Arduino que hace parpadear un LED conectado al pin digital 13. El LED se enciende durante 1 segundo y luego se apaga por 1 segundo, repitiendo este ciclo indefinidamente.

Materiales necesarios:

- 1 Placa Arduino (cualquier modelo que tenga el pin 13 disponible).

- 1 LED (aunque la mayoría de las placas Arduino ya tienen un LED integrado en el pin 13, no necesitarías uno externo).
- 1 Resistor (si se utiliza un LED externo para limitar la corriente, generalmente de 220 ohmios).

```

void setup() {
  pinMode(13, OUTPUT); // Configura el pin 13 como
  salida
}

void loop() {
  digitalWrite(13, HIGH); // Enciende el LED
  delay(1000); // Espera 1 segundo
  digitalWrite(13, LOW); // Apaga el LED
  delay(1000); // Espera 1 segundo
}

```

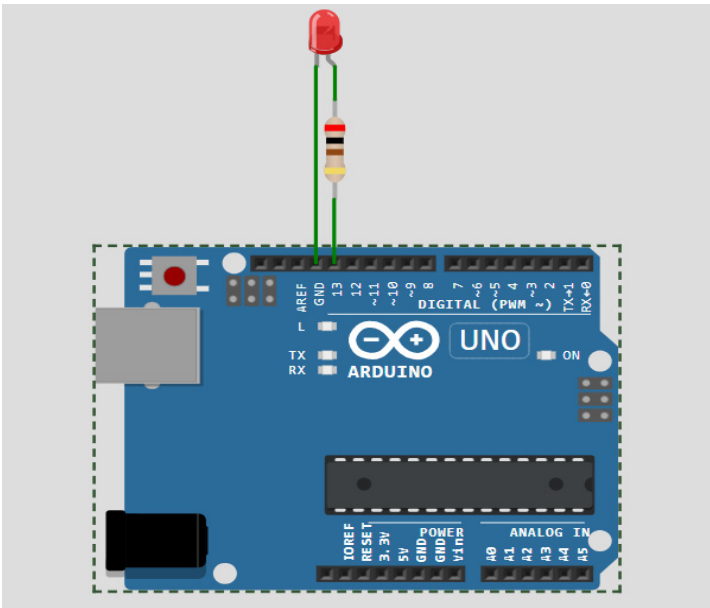


Figura 27. Encender y Apagar Led.

Ejemplo 14 – Leer estado de un pulsador

Este código permite leer el estado de un botón conectado al pin 2 de una placa Arduino y muestra el resultado en el monitor serial. Dependiendo de si el botón está presionado o no, el programa enviará un valor de 1 (si está presionado) o 0 (si no lo está) a través de la comunicación serial, mostrando el estado cada medio segundo.

Materiales necesarios:

- 1 Placa Arduino.
- 1 Botón pulsador.
- 1 Resistor de 10k ohmios (para usar como resistencia pull-down o pull-up, dependiendo de la configuración del botón).
- Cables de conexión.

```
void setup() {  
  pinMode(2, INPUT); // Configura el pin 2 como entrada  
  Serial.begin(9600); // Inicializa la comunicación  
  serial  
}  
  
void loop() {  
  int buttonState = digitalRead(2); // Lee el estado  
  del botón  
  Serial.println(buttonState); // Imprime el estado  
  en el monitor serial  
  delay(500); // Espera medio segundo  
}
```

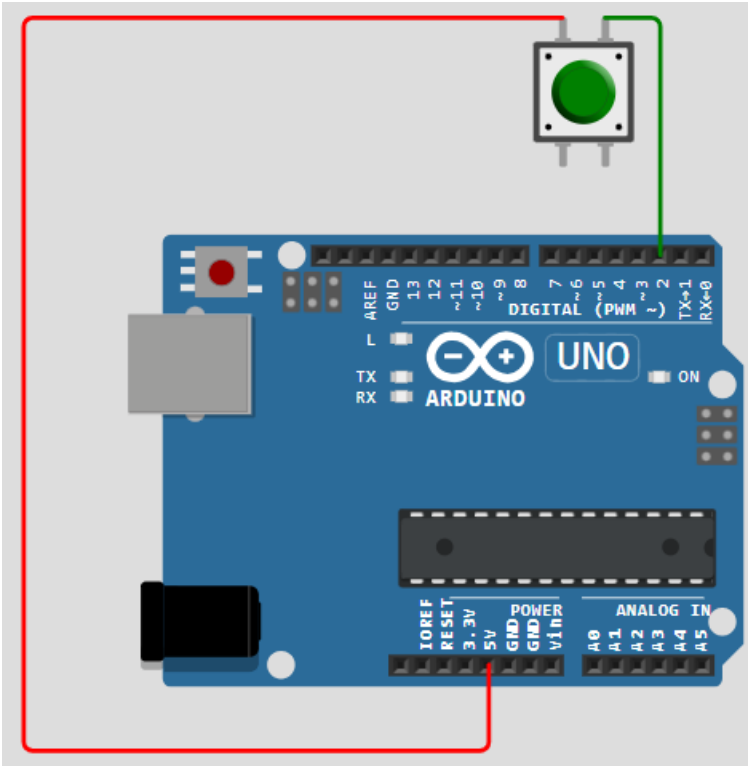


Figura 28. Leer estado de un pulsador.

Ejemplo 15 – Encender led con pulsador

Este código controla un LED utilizando un botón pulsador conectado a una placa Arduino. Cuando se presiona el botón, el LED se enciende; cuando se suelta, el LED se apaga. Es un ejemplo básico de cómo leer una entrada (el botón) y controlar una salida (el LED) en tiempo real.

Materiales necesarios:

- 1 Placa Arduino.
- 1 LED.
- 1 Resistor (generalmente de 220 ohmios para limitar la corriente al LED).
- 1 Botón pulsador.
- 1 Resistor de 10k ohmios (para utilizar como resistencia pull-down o pull-up, según sea necesario).
- Cables de conexión.

```
// Definición de pines
const int ledPin = 13; // Pin donde está conectado
el LED
const int buttonPin = 2; // Pin donde está conectado
el pulsador

void setup() {
  pinMode(ledPin, OUTPUT); // Configura el pin del LED
como salida
  pinMode(buttonPin, INPUT); // Configura el pin del
pulsador como entrada
}

void loop() {
  if (digitalRead(buttonPin) == HIGH) { // Si el
pulsador está presionado
    digitalWrite(ledPin, HIGH); // Enciende el LED
  } else {
    digitalWrite(ledPin, LOW); // Apaga el LED
  }
}
```

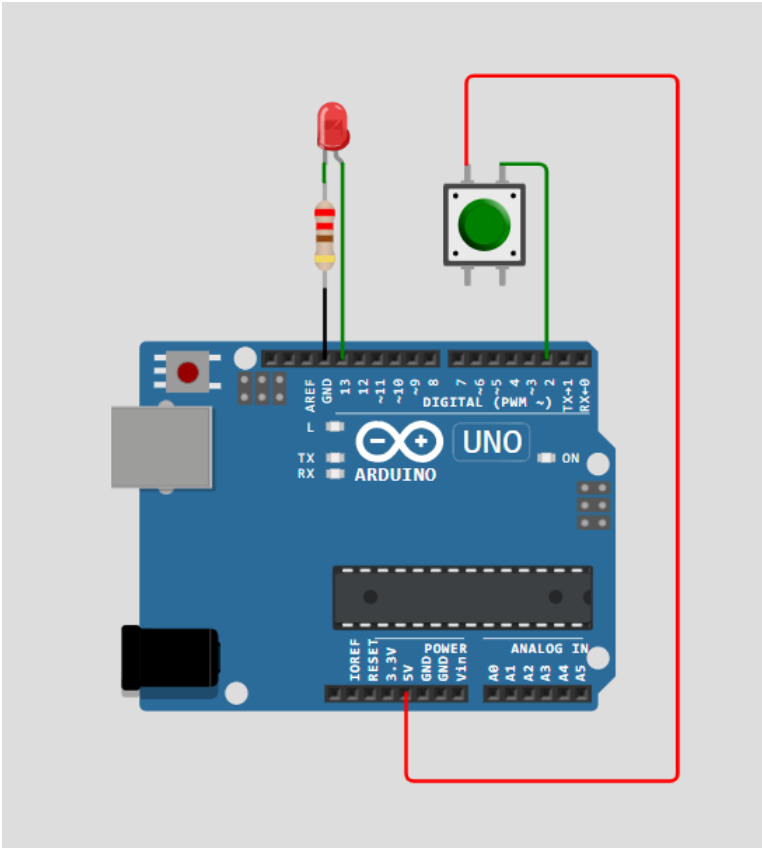


Figura 29. Encender led con pulsador.

Ejemplo 16 – Efecto Rebote

El **efecto rebote** (o *bouncing* en inglés) es un fenómeno que ocurre cuando se presiona un interruptor o pulsador. Cuando el botón se activa, en lugar de tener un cambio limpio y claro de estado (de LOW a HIGH, por ejemplo), el estado del botón puede fluctuar rápidamente entre HIGH y LOW debido a pequeñas vibraciones mecánicas en el interruptor. Esta fluctuación puede generar múltiples lecturas en un corto período de tiempo, lo

que provoca que el sistema interprete varios pulsos cuando, en realidad, solo hubo uno.

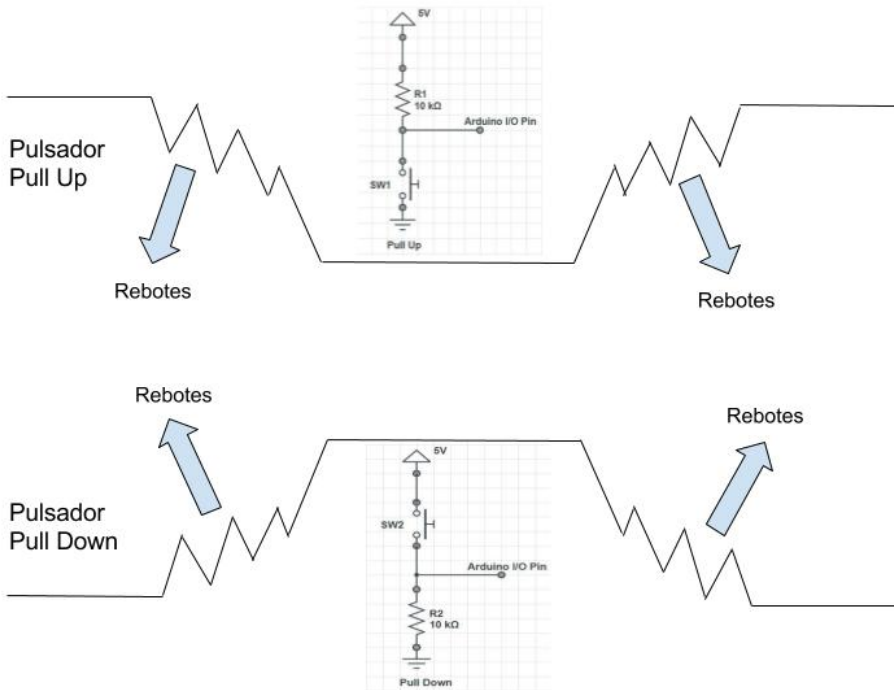


Figura 30. Efecto rebote de un pulsador.

Fuente: Vega (2017).

La figura 30 muestra como al presionar un pulsador se generan señales que provocan el efecto rebote.

Ejemplo del Efecto Rebote

Imagina que presionas un pulsador. En lugar de cambiar de estado de LOW a HIGH de forma directa, el estado puede verse así:

Tiempo: 0ms 1ms 2ms 3ms 4ms

Estado: LOW HIGH LOW HIGH LOW

Esto puede causar que el LED se encienda y apague rápidamente, en lugar de permanecer encendido.

Cómo Evitar el Efecto Rebote

Hay varias formas de evitar el efecto rebote en Arduino:

- 1. Delay (Retardo Simple):** Una forma sencilla es agregar un pequeño retardo después de detectar un cambio en el estado del botón. Esto da tiempo para que el rebote se detenga antes de leer el estado nuevamente.

```
const int ledPin = 13; // Pin donde está conectado
el LED
const int buttonPin = 2; // Pin donde está conectado
el pulsador

void setup() {
  pinMode(ledPin, OUTPUT); // Configura el pin del LED
  como salida
  pinMode(buttonPin, INPUT); // Configura el pin del
  pulsador como entrada
}

void loop() {
  if (digitalRead(buttonPin) == HIGH) { // Si el
  pulsador está presionado
    digitalWrite(ledPin, HIGH); // Enciende el LED
    delay(500);
  } else {
    digitalWrite(ledPin, LOW); // Apaga el LED
    delay(500);
  }
}
```

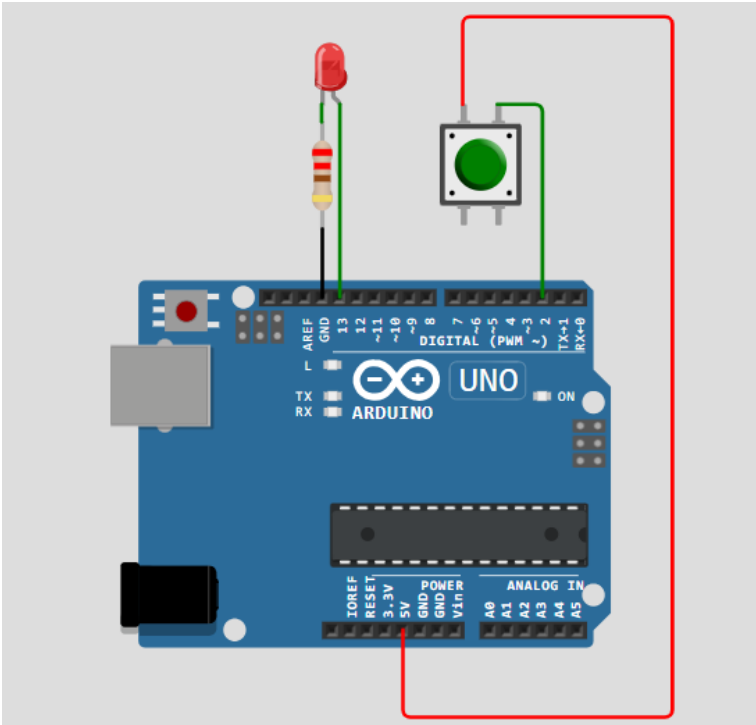


Figura 31. Efecto Rebote.

Ejemplo 17 – Controlar 4 leds con 4 pulsadores

Este código controla múltiples LEDs mediante varios botones pulsadores conectados a una placa Arduino. Cada botón está asociado a un LED específico. Cuando se presiona un botón, el LED correspondiente se enciende; cuando se suelta, el LED se apaga. El programa repite este proceso para cuatro pares de botones y LEDs.

Materiales necesarios:

- 1 Placa Arduino.
- 4 LEDs.

- 4 Resistores (generalmente de 220 ohmios para limitar la corriente a los LEDs).
- 4 Botones pulsadores.
- 4 Resistores de 10k ohmios (para utilizar como resistencias pull-down o pull-up según sea necesario).
- Cables de conexión.

```

const int buttonPins[] = {2, 3, 4, 5}; // Pines de
los pulsadores
const int ledPins[] = {6, 7, 8, 9}; // Pines de los
LEDs

void setup() {
  for (int i = 0; i < 4; i++) {
    pinMode(buttonPins[i], INPUT); // Configura los pines
de los pulsadores como entrada
    pinMode(ledPins[i], OUTPUT); // Configura los pines
de los LEDs como salida
  }
}

void loop() {
  for (int i = 0; i < 4; i++) {
    int buttonState = digitalRead(buttonPins[i]); //
Lee el estado de cada pulsador

    if (buttonState == HIGH) {
      digitalWrite(ledPins[i], HIGH); // Enciende el LED
correspondiente
    } else {

```

```
digitalWrite(ledPins[i], LOW); // Apaga el LED correspondiente
}
}
}
```

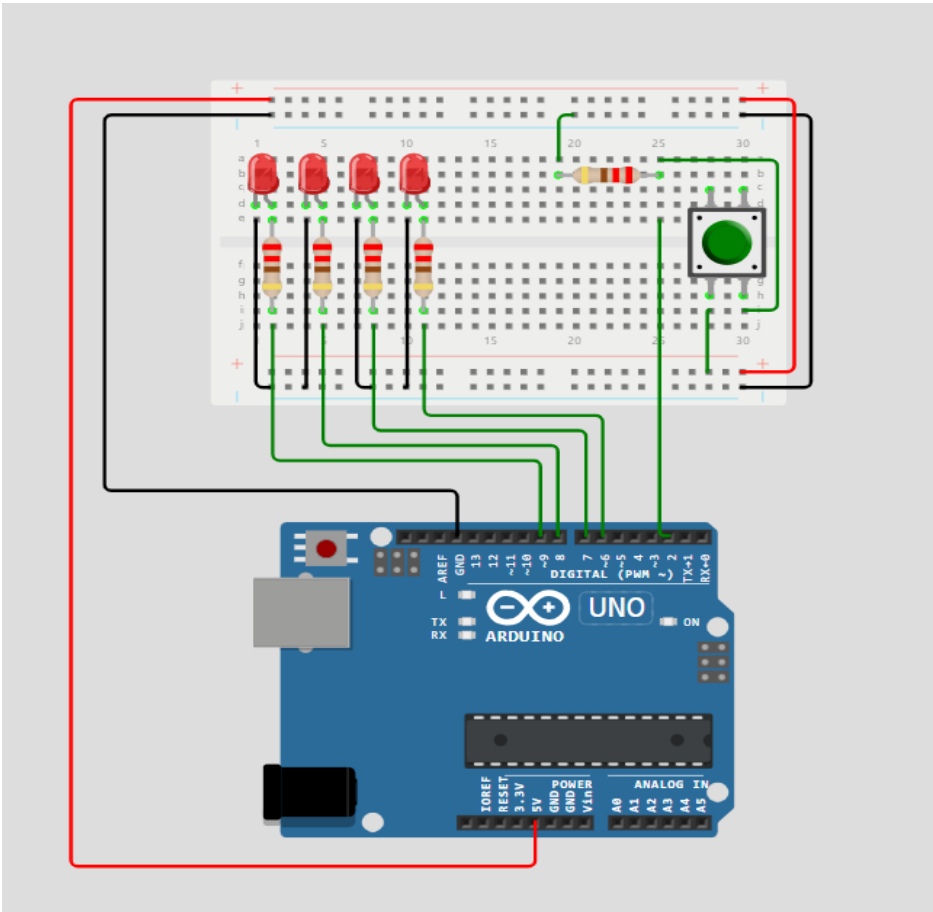


Figura 32. Controlar 4 leds con 4 pulsadores.

Ejemplo 18 - Encender LEDs en secuencia con un pulsador

Botón, se enciende un LED por un breve período de tiempo, luego se apaga y el control pasa al siguiente LED en la secuencia. El proceso continúa cíclicamente para los cuatro LEDs.

Materiales necesarios:

- 1 Placa Arduino.
- 4 LEDs.
- 4 Resistores (generalmente de 220 ohmios para limitar la corriente a los LEDs).
- 1 Botón pulsador.
- 1 Resistor de 10k ohmios (para utilizar como resistencia pull-down o pull-up, dependiendo de la configuración del botón).
- Cables de conexión.

```
const int ledPins[] = {6, 7, 8, 9}; // Pines de los LEDs
const int buttonPin = 2; // Pin del pulsador
int currentLED = 0; // Índice del LED actual

void setup() {
  pinMode(buttonPin, INPUT);

  for (int i = 0; i < 4; i++) {
    pinMode(ledPins[i], OUTPUT);
  }
}
```

```

void loop() {
  if (digitalRead(buttonPin) == HIGH) {
    digitalWrite(ledPins[currentLED], HIGH); // Enciende
    el LED actual
    delay(500); // Espera un poco
    digitalWrite(ledPins[currentLED], LOW); // Apaga el
    LED actual
    currentLED = (currentLED + 1) % 4; // Cambia al
    siguiente LED
  }
}

```

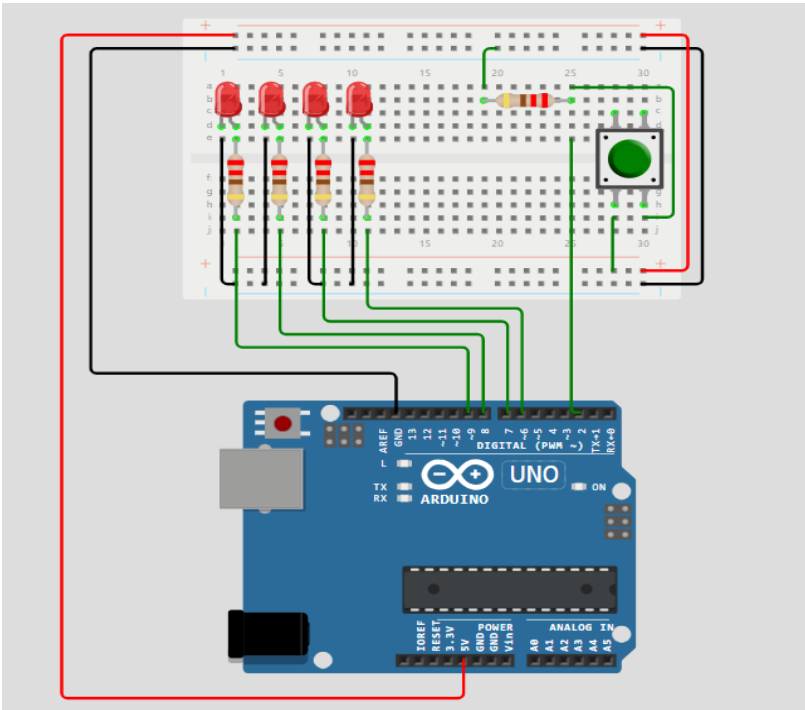


Figura 33. Encender LEDs en Secuencia con un Pulsador.

Ejemplo 19 - Controlar un LED con dos pulsadores (Encender y Apagar)

Este código controla un LED utilizando dos botones pulsadores. Un botón se utiliza para encender el LED, y otro botón para apagarlo. Dependiendo de cuál botón se presione, el LED cambiará su estado entre encendido y apagado.

Materiales necesarios:

- 1 Placa Arduino.
- 1 LED.
- 1 Resistor (generalmente de 220 ohmios para limitar la corriente al LED).
- 2 Botones pulsadores.
- 2 Resistores de 10k ohmios (para usar como resistencias pull-down o pull-up, según sea necesario).
- Cables de conexión.

```
const int buttonOnPin = 2; // Pin del pulsador para
encender
const int buttonOffPin = 3; // Pin del pulsador para
apagar
const int ledPin = 13; // Pin del LED

void setup() {
  pinMode(buttonOnPin, INPUT);
  pinMode(buttonOffPin, INPUT);
  pinMode(ledPin, OUTPUT);
}
```

```

void loop() {
  if (digitalRead(buttonOnPin) == HIGH) {
    digitalWrite(ledPin, HIGH); // Enciende el LED si
    se presiona el botón "encender"
  }
  if (digitalRead(buttonOffPin) == HIGH) {
    digitalWrite(ledPin, LOW); // Apaga el LED si se
    presiona el botón "apagar"
  }
}

```

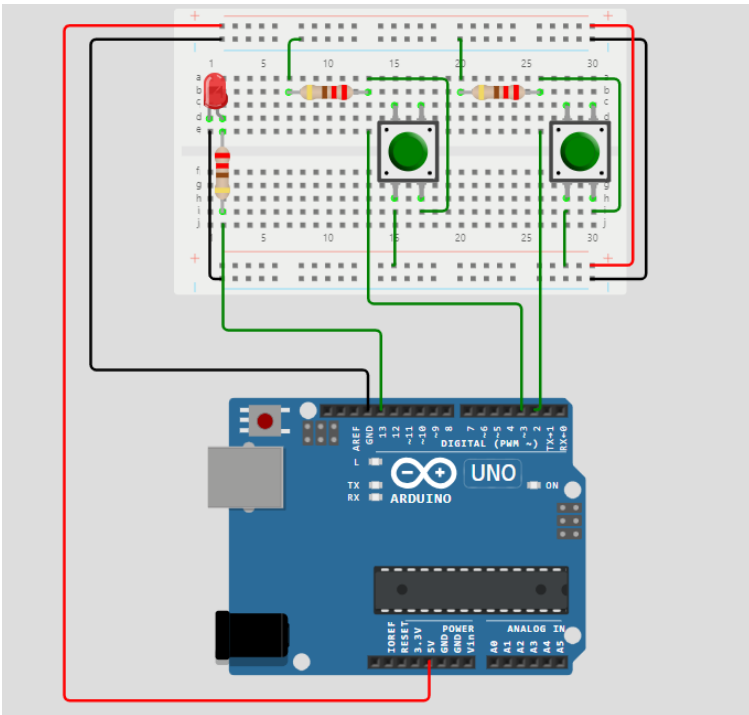


Figura 34. Controlar un LED con dos Pulsadores (Encender y Apagar).

Ejemplo 20 - Secuencia de LEDs con un Pulsador

Este código controla una secuencia de cuatro LEDs con un botón pulsador. Cada vez que el botón se presiona, el LED actual se enciende brevemente, luego se apaga, y el siguiente LED en la secuencia toma su lugar. Este proceso continúa cíclicamente para los cuatro LEDs, con una breve pausa entre el encendido de cada LED.

Materiales necesarios:

- 1 Placa Arduino.
- 4 LEDs.
- 4 Resistores (generalmente de 220 ohmios para limitar la corriente a los LEDs).
- 1 Botón pulsador.
- 1 Resistor de 10k ohmios (para usar como resistencia pull-down o pull-up, según sea necesario).
- Cables de conexión.

```

const int ledPins[] = {6, 7, 8, 9}; // Pines de los LEDs
const int buttonPin = 2; // Pin del pulsador
int currentLED = 0; // Índice del LED actual

void setup() {
  pinMode(buttonPin, INPUT);

  for (int i = 0; i < 4; i++) {
    pinMode(ledPins[i], OUTPUT); // Configura todos los LEDs como salida
  }
}

void loop() {
  if (digitalRead(buttonPin) == HIGH) {
    digitalWrite(ledPins[currentLED], HIGH); // Enciende el LED actual
    delay(200); // Espera un poco
    digitalWrite(ledPins[currentLED], LOW); // Apaga el LED actual
    currentLED = (currentLED + 1) % 4; // Cambia al siguiente LED
    delay(200); // Pausa antes del siguiente ciclo
  }
}

```

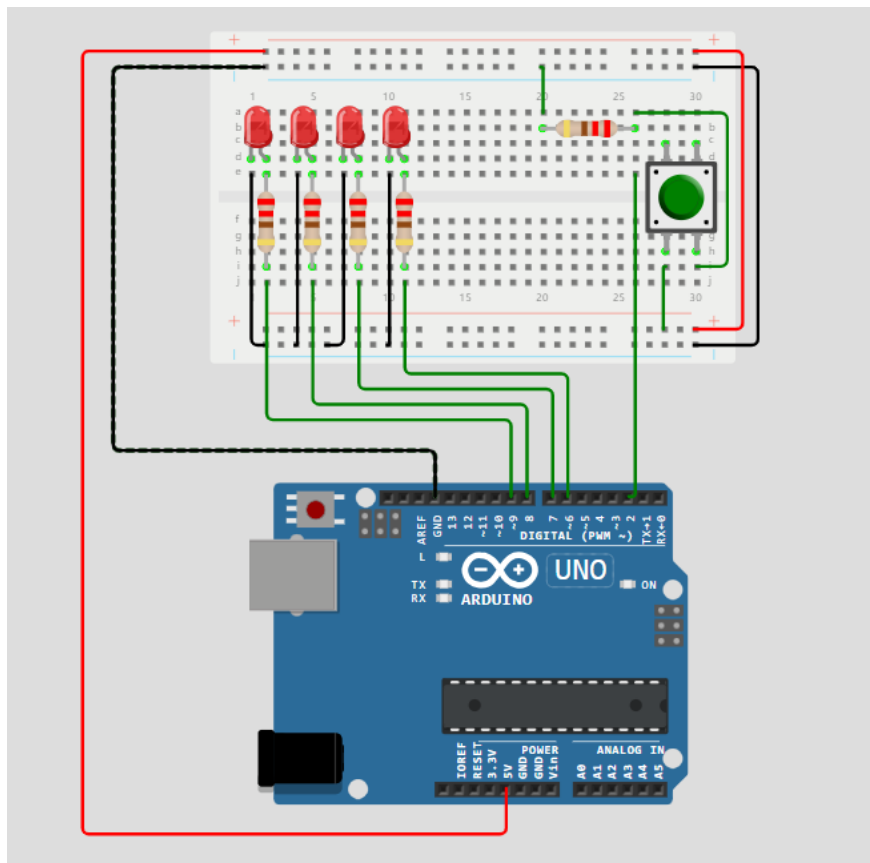



Figura 35. Secuencia de LEDs con un Pulsador.

Ejemplo 21 - Pulsador para cambiar el estado de un LED (Interruptor ON/OFF)

Este código permite alternar el estado de un LED con un solo botón pulsador. Cada vez que se presiona el botón, el LED cambia su estado: si estaba apagado, se enciende, y si estaba encendido, se apaga. El código utiliza la técnica de “detección de flancos” para asegurarse de que el LED solo cambie de estado cuando el botón pasa de no estar presionado a presionado, evitando múltiples cambios durante una única pulsación.

Materiales necesarios:

- 1 Placa Arduino.
- 1 LED.
- 1 Resistor (generalmente de 220 ohmios para limitar la corriente al LED).
- 1 Botón pulsador.
- 1 Resistor de 10k ohmios (para usar como resistencia pull-down o pull-up, según sea necesario).
- Cables de conexión.

```
const int buttonPin = 2; // Pin del pulsador
const int ledPin = 13; // Pin del LED
int ledState = LOW; // Estado inicial del LED
int lastButtonState = LOW; // Último estado del botón
int buttonState; // Estado actual del botón

void setup() {
  pinMode(buttonPin, INPUT);
  pinMode(ledPin, OUTPUT);
}

void loop() {
  buttonState = digitalRead(buttonPin); // Lee el
  estado del botón

  if (buttonState == HIGH && lastButtonState == LOW) {
    ledState = !ledState; // Alterna el estado del LED
    digitalWrite(ledPin, ledState); // Cambia el estado
    del LED
  }

  lastButtonState = buttonState; // Guarda el estado
  del botón

  delay(50); // Pequeño delay para evitar rebotes
}
```

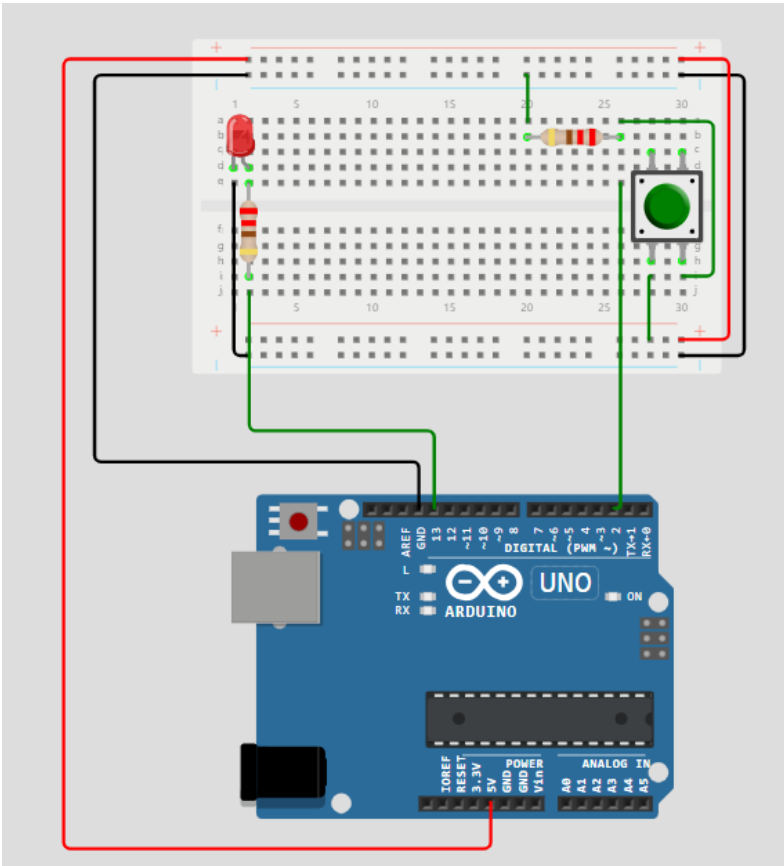


Figura 36. Pulsador para cambiar el estado de un LED (Interruptor ON/OFF).

Ejemplo 22 - LED Parpadeante Controlado por un Pulsador

Este código controla un LED que alterna entre dos modos: parpadeo y apagado, utilizando un solo botón pulsador. Cada vez que se presiona el botón, el LED cambia entre encenderse y apagarse continuamente (parpadeo) o mantenerse apagado. El parpadeo ocurre con un intervalo de 500 ms encendido y 500 ms apagado.

Materiales necesarios:

- 1 Placa Arduino.
- 1 LED.
- 1 Resistor (generalmente de 220 ohmios para limitar la corriente al LED).
- 1 Botón pulsador.
- 1 Resistor de 10k ohmios (para usar como resistencia pull-down o pull-up, según sea necesario).
- Cables de conexión.

```
const int buttonPin = 2; // Pin del pulsador
const int ledPin = 13; // Pin del LED
bool isBlinking = false; // Flag para controlar si
el LED está parpadeando
int lastButtonState = LOW; // Último estado del
pulsador

void setup() {
  pinMode(buttonPin, INPUT);
  pinMode(ledPin, OUTPUT);
}

void loop() {
  int buttonState = digitalRead(buttonPin); // Lee el
estado del pulsador
```

```
if (buttonState == HIGH && lastButtonState == LOW) {
  isBlinking = !isBlinking; // Alterna el estado de
  parpadeo
}

if (isBlinking) {
  digitalWrite(ledPin, HIGH);
  delay(500);
  digitalWrite(ledPin, LOW);
  delay(500);
} else {
  digitalWrite(ledPin, LOW); // Mantiene el LED apagado
  si no está parpadeando
}

  lastButtonState = buttonState; // Actualiza el
  último estado del pulsador
}
```

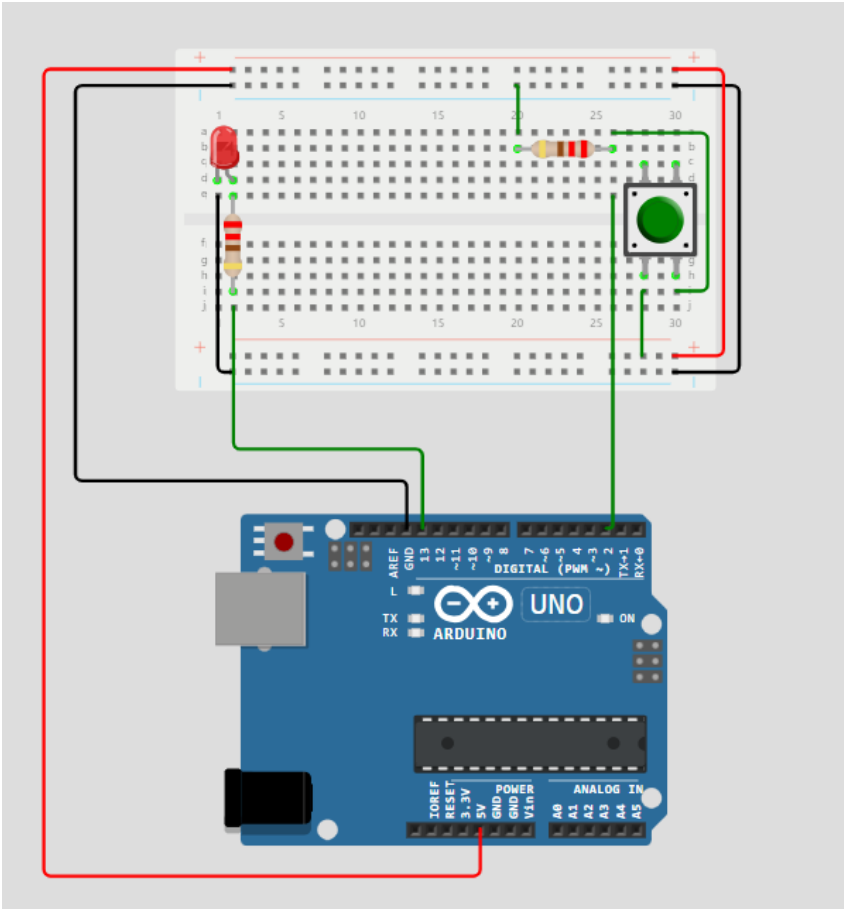


Figura 37. LED Parpadeante controlado por un Pulsador.

CAPÍTULO IV.

Entradas y salidas analógicas (PWM)

Concepto básico

El capítulo sobre Entradas y Salidas Analógicas (PWM) profundiza en cómo Arduino maneja señales que permiten variaciones en el nivel de señal, en lugar de solo encender o apagar. La modulación por ancho de pulso (PWM) permite controlar dispositivos como LEDs y motores que requieren ajustes en intensidad o velocidad. Este capítulo explora el uso de potenciómetros para introducir valores analógicos a la placa Arduino y cómo estos afectan otros componentes.

A través de ejemplos prácticos, los lectores aprenderán a cambiar el brillo de un LED usando un potenciómetro, controlar múltiples LEDs, y usar PWM para mezclar colores en un LED RGB o simular la velocidad de un motor mediante el brillo. También se presenta cómo observar el comportamiento de las señales analógicas en el Monitor Serial, y un proyecto de semáforo controlado por un potenciómetro que integra los conceptos abordados. Como afirma Gordon (2022),

“el manejo de entradas y salidas analógicas es esencial para cualquier proyecto que busque un control más refinado y efectivo sobre los dispositivos conectados” (p. 112). Este capítulo brinda las herramientas para implementar controles analógicos en proyectos de Arduino.

Las entradas analógicas permiten al Arduino leer valores variables que no son simplemente HIGH o LOW como en el caso de las entradas digitales. En lugar de solo detectar si hay voltaje o no, pueden medir cualquier valor dentro de un rango continuo. Esto es útil para sensores que varían su salida de forma gradual, como un sensor de temperatura, un potenciómetro, o un sensor de luz.

El Arduino utiliza un conversor **ADC** (Analog to Digital Converter) para convertir la señal analógica (tensión continua) en un número digital que el microcontrolador puede interpretar. El Arduino UNO, por ejemplo, tiene un **ADC de 10 bits**, lo que significa que puede convertir una señal analógica en valores que van desde 0 hasta 1023 ($2^{10} = 1024$ niveles).

Pines de Entradas Analógicas

En el Arduino UNO, los pines de entrada analógica están etiquetados como **A0, A1, A2, A3, A4, y A5**. Estos pines son capaces de leer tensiones entre **0V y 5V**, que luego se convierten en un valor digital entre 0 y 1023.

Funcionamiento del ADC

- **Rango de voltaje:** En los microcontroladores Arduino (con una referencia de 5V), un valor analógico de 0 significa 0V, y un valor de 1023 significa 5V. Si el voltaje es de 2.5V, el valor leído será aproximadamente 512.

Fórmula para convertir el valor digital a voltaje:

$$\text{Voltaje} = \frac{\text{Valor}_{\text{Leído}} \times \text{Voltaje}_{\text{referencia}}}{1023}$$

Ejemplo: Si el valor leído es 512, el voltaje es:

$$\text{Voltaje} = \frac{512 \times 5}{1023} \approx 2.5V$$

Comando para Leer Entradas Analógicas

La función `analogRead(pin)` se utiliza para leer el valor de un pin analógico. Devuelve un número entre 0 y 1023, dependiendo del voltaje medido.

EJEMPLO 22 – Leer un potenciómetro

Este código lee el valor de un potenciómetro conectado a un pin analógico del Arduino y lo envía al monitor serial. El potenciómetro permite variar la resistencia y, por lo tanto, el voltaje, lo que cambia el valor leído por el Arduino. Este valor se imprime continuamente en el monitor serial, proporcionando una forma de observar cómo cambia la lectura conforme se ajusta el potenciómetro.

Materiales necesarios:

- 1 Placa Arduino.
- 1 Potenciómetro.
- Cables de conexión.

```

int potPin = A0; // Pin donde está conectado el
potenciómetro
int potValue; // Variable para almacenar el valor
leído

void setup() {
  Serial.begin(9600); // Inicializa el monitor serial
}

void loop() {
  potValue = analogRead(potPin); // Lee el valor
analógico
  Serial.println(potValue); // Imprime el valor leído
en el monitor serial
  delay(100);
}

```

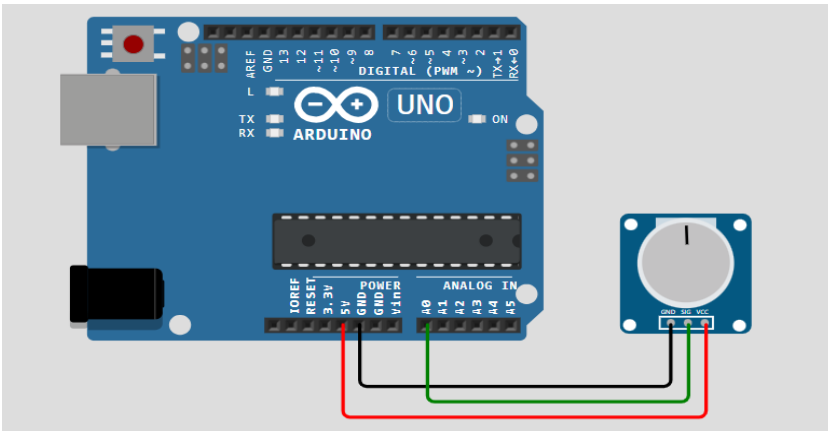


Figura 38. Leer un potenciómetro.

PWM (Modulación por Ancho de Pulso) en Arduino

PWM o Modulación por Ancho de Pulso es una técnica utilizada para emular una señal analógica utilizando una salida digital. Esto se logra mediante el encendido y apagado rápido de una señal digital, controlando el tiempo que la señal permanece en “ALTO” o “BAJO”. Esta técnica es ampliamente utilizada en aplicaciones como el control de velocidad de motores, regulación de intensidad luminosa y control de temperatura en dispositivos electrónicos, ya que es una forma eficiente de controlar la energía sin generar calor excesivo en los componentes (Bonsor, 2023).

¿Cómo funciona el PWM?

En una señal **PWM**, se alterna rápidamente entre los estados **ALTO (5V)** y **BAJO (0V)**, y la clave para simular un comportamiento analógico está en el ciclo de trabajo o **duty cycle**.

Duty Cycle:

El **duty cycle** es el porcentaje de tiempo que la señal permanece en estado “ALTO” dentro de un ciclo completo. Por ejemplo:

- Un duty cycle del **100%** significa que la señal está **siempre** en “ALTO”.
- Un duty cycle del **0%** significa que la señal está **siempre** en “BAJO”.
- Un duty cycle del **50%** significa que la señal está “ALTO” el **50%** del tiempo y “BAJO” el otro **50%**.

Esto da la impresión de que la salida es una señal analógica. Aunque en realidad sigue siendo una señal digital, la rápida alternancia entre “ALTO” y “BAJO” hace que, por ejemplo, un LED parezca variar su brillo o un motor varíe su velocidad.

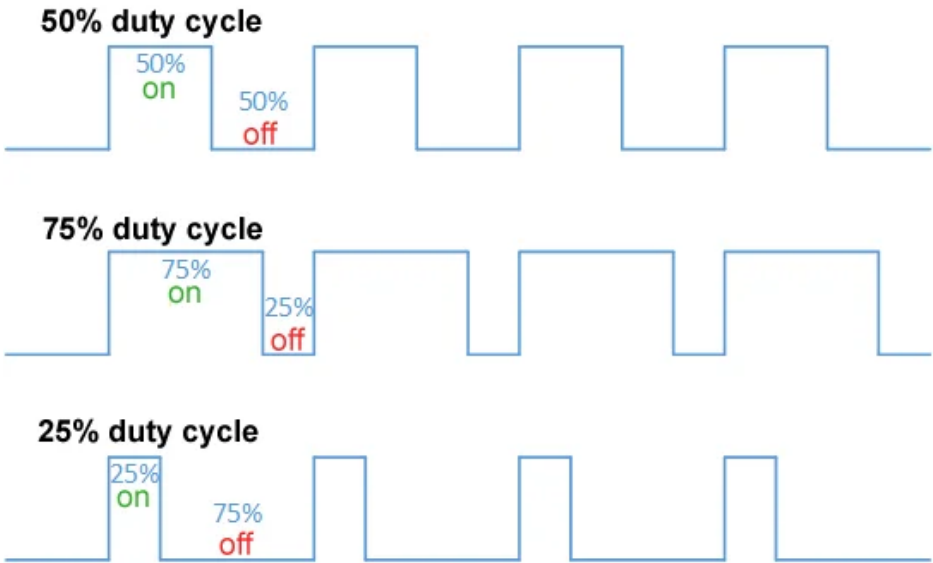


Figura 39. Ciclo de funcionamiento del PWM.

Fuente: Corsair Gaming (2024).

La figura 39 muestra el ciclo de funcionamiento de la señal de PWM usada por el arduino.

Frecuencia:

La frecuencia de una señal PWM determina cuántas veces se completa un ciclo de encendido/apagado por segundo. En Arduino, la frecuencia de los pines PWM estándar es de aproximadamente **490 Hz** (ciclos por segundo), pero en algunos pines, como el pin 5 y 6, es más alta: **980 Hz**.

PWM en Arduino

Arduino tiene varios pines que soportan **PWM**. En una placa como el Arduino Uno, los pines que soportan **PWM** están marcados con una tilde (~) junto al número de pin, y estos son: **3, 5, 6, 9, 10 y 11**.

El PWM en Arduino se genera mediante la función `analogWrite()`. Esta función permite ajustar el **duty cycle** de la señal de 0 a 255, donde:

- `analogWrite(pin, 0)` genera una señal con un **duty cycle** del 0% (siempre apagado).
- `analogWrite(pin, 255)` genera una señal con un **duty cycle** del 100% (siempre encendido).
- `analogWrite(pin, 127)` genera una señal con un **duty cycle** del 50% (encendido la mitad del tiempo).

Función map()

Esta función es muy útil cuando se trabaja con valores analógicos porque permite convertir un valor de un rango a otro.

```
long map(long x, long in_min, long in_max, long out_min, long out_max);
```

Donde:

- `x` es el valor que queremos mapear.
- `in_min` y `in_max` son los valores mínimos y máximos del rango de entrada.
- `out_min` y `out_max` son los valores mínimos y máximos del rango de salida.

Ejemplo: Si tienes un valor entre 0 y 1023 (de una entrada analógica) y quieres convertirlo a un rango de 0 a 255 (para una salida PWM), puedes usar `map()` de la siguiente manera:

```
ledBrightness = map(potValue, 0, 1023, 0, 255);
```

Ejemplo 23 - Cambiar el brillo de un LED según la posición del potenciómetro

Este código permite controlar el brillo de un LED utilizando un potenciómetro. Lee el valor del potenciómetro, que varía entre 0

y 1023, y lo mapea a un rango de 0 a 255, que es el rango que utiliza la función `analogWrite()` para ajustar el brillo de un LED en un pin PWM (modulación de ancho de pulso). A medida que se gira el potenciómetro, el brillo del LED se ajusta en consecuencia.

Materiales necesarios:

- 1 Placa Arduino.
- 1 Potenciómetro.
- 1 LED.
- 1 Resistor (generalmente de 220 ohmios para limitar la corriente al LED).
- Cables de conexión.

```
int potPin = A0; // Pin del potenciómetro
int ledPin = 9; // Pin del LED con capacidad PWM
int potValue; // Variable para almacenar el valor del
potenciómetro
int ledBrightness; // Variable para almacenar el
brillo del LED

void setup() {
  pinMode(ledPin, OUTPUT); // Configura el pin del LED
  como salida
}

void loop() {
  potValue = analogRead(potPin); // Lee el valor del
  potenciómetro (0 a 1023)
```

```
ledBrightness = map(potValue, 0, 1023, 0, 255); //  
Mapea el valor al rango 0-255  
  analogWrite(ledPin, ledBrightness); // Ajusta el  
brillo del LED  
}
```

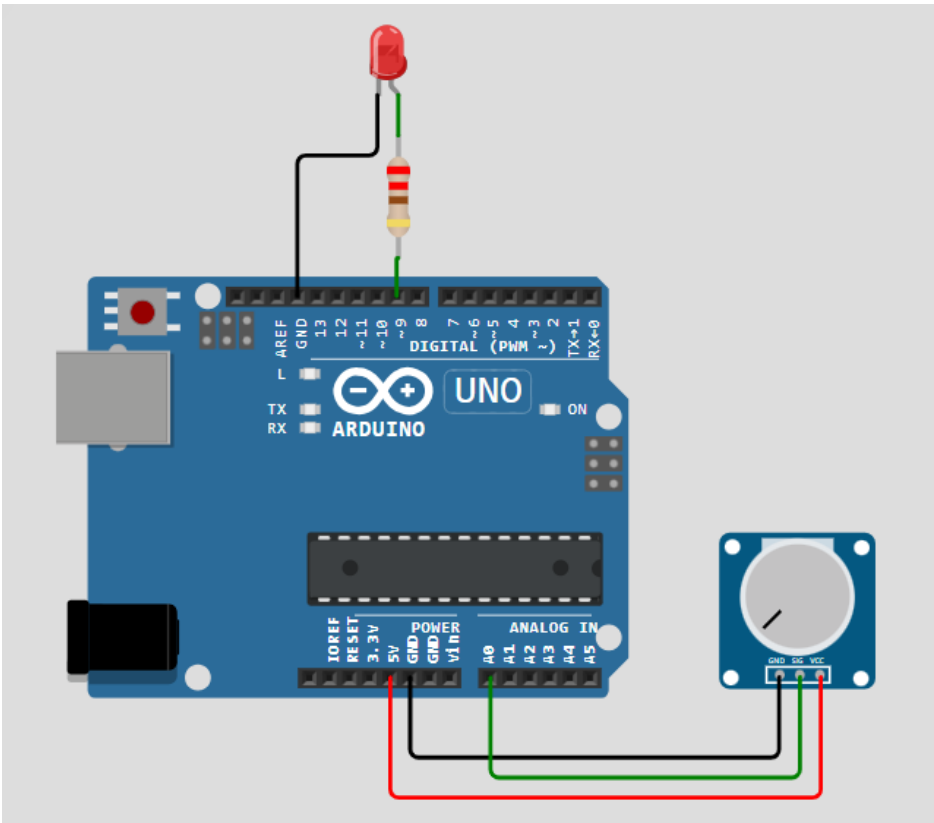


Figura 40. Cambiar el brillo de un LED según la posición del potenciómetro.

Ejemplo 24 - Control de múltiples LEDs con un solo potenciómetro

Este código controla el encendido de múltiples LEDs en función de la posición de un potenciómetro. Lee el valor del potenciómetro y lo mapea a la cantidad de LEDs que se deben encender, desde 0 hasta el número total de LEDs definidos. Dependiendo de la posición del potenciómetro, se encenderán uno, dos o los tres LEDs conectados a los pines designados.

Materiales necesarios:

- 1 Placa Arduino.
- 1 Potenciómetro.
- 3 LEDs.
- 3 Resistores (generalmente de 220 ohmios cada uno para limitar la corriente a los LEDs).
- Cables de conexión.

```

int potPin = A0; // Pin del potenciómetro
int ledPins[] = {9, 10, 11}; // Pines de los LEDs
int potValue; // Almacenar el valor del potenciómetro
int numLEDs = 3; // Número de LEDs

void setup() {
  for (int i = 0; i < numLEDs; i++) {
    pinMode(ledPins[i], OUTPUT); // Configura todos los
    pines de los LEDs como salida
  }
}

void loop() {
  potValue = analogRead(potPin); // Lee el valor del
  potenciómetro
  int ledsToLight = map(potValue, 0, 1023, 0, numLEDs);
  // Mapea el valor a la cantidad de LEDs a encender

  for (int i = 0; i < numLEDs; i++) {
    if (i < ledsToLight) {
      digitalWrite(ledPins[i], HIGH); // Enciende el LED
    } else {
      digitalWrite(ledPins[i], LOW); // Apaga el LED
    }
  }
}

```

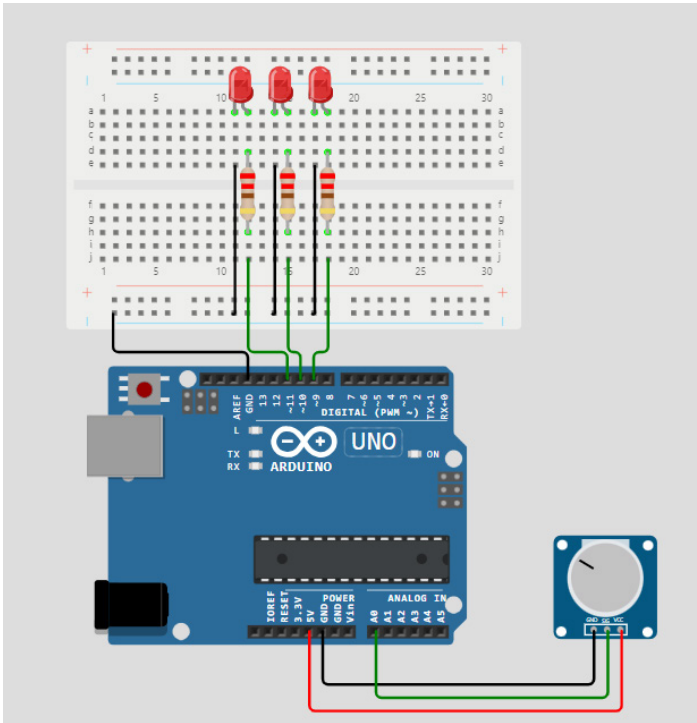


Figura 41. Control de múltiples LEDs con un solo potenciómetro.

Ejemplo 25 - Control de un LED RGB con un potenciómetro

Un LED RGB es un tipo de diodo emisor de luz (LED) que combina tres colores primarios: rojo, verde y azul. Estos tres colores se pueden mezclar en diferentes intensidades para producir una amplia gama de colores. Los LEDs RGB se utilizan comúnmente en aplicaciones de iluminación decorativa, pantallas y dispositivos electrónicos, ya que permiten crear efectos visuales dinámicos y personalizados. El control de los LEDs RGB se realiza mediante la modulación de la intensidad de cada uno de los tres colores, lo que proporciona un control preciso sobre el color final que se emite. Este tipo de LED es especialmente popular en proyectos

de electrónica, como en la creación de pantallas o la iluminación ambiental controlada por microcontroladores como Arduino (Martínez, 2021).

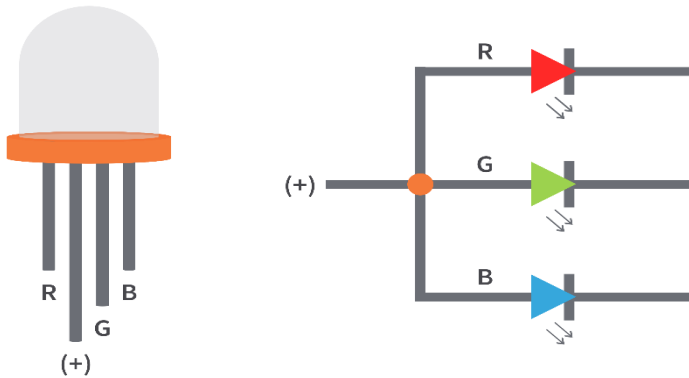


Figura 42. Representación de un led RGB.

Fuente: Interfacing RGB led with Arduino (2022).

La figura 42 muestra la representación interna de un led RGB, para colores rojo, verde y azul.

Este código permite controlar el color de un LED RGB utilizando un potenciómetro. A medida que se ajusta el potenciómetro, los valores de los canales rojo, verde y azul del LED RGB se modifican en función de su posición. El valor del potenciómetro se mapea de manera que el canal rojo varía desde 0 hasta 255, el canal verde varía inversamente (de 255 a 0) y el canal azul tiene un rango intermedio (de 127 a 255). Esto produce una transición de colores en el LED RGB en función de la posición del potenciómetro.

Materiales necesarios:

- 1 Placa Arduino.
- 1 Potenciómetro.
- 1 LED RGB (de cátodo común o ánodo común).

- 3 Resistores (generalmente de 220 ohmios cada uno para limitar la corriente a los LEDs).
- Cables de conexión.

```
int potPin = A0; // Pin del potenciómetro
int redPin = 9; // Pin del LED rojo
int greenPin = 10; // Pin del LED verde
int bluePin = 11; // Pin del LED azul

void setup() {
  pinMode(redPin, OUTPUT); // Configura los pines del
  LED RGB como salida
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
}

void loop() {
  int potValue = analogRead(potPin); // Lee el valor
  del potenciómetro
  int redValue = map(potValue, 0, 1023, 0, 255); //
  Mapea el valor del potenciómetro para el canal rojo

  int greenValue = map(potValue, 0, 1023, 255, 0); //
  Mapea el valor del potenciómetro para el canal verde
  int blueValue = map(potValue, 0, 1023, 127, 255); //
  Mapea el valor para el azul con un valor intermedio
```

```

    analogWrite(redPin, redValue); // Ajusta la
intensidad del canal rojo

    analogWrite(greenPin, greenValue); // Ajusta la
intensidad del canal verde

    analogWrite(bluePin, blueValue); // Ajusta la
intensidad del canal azul
}

```

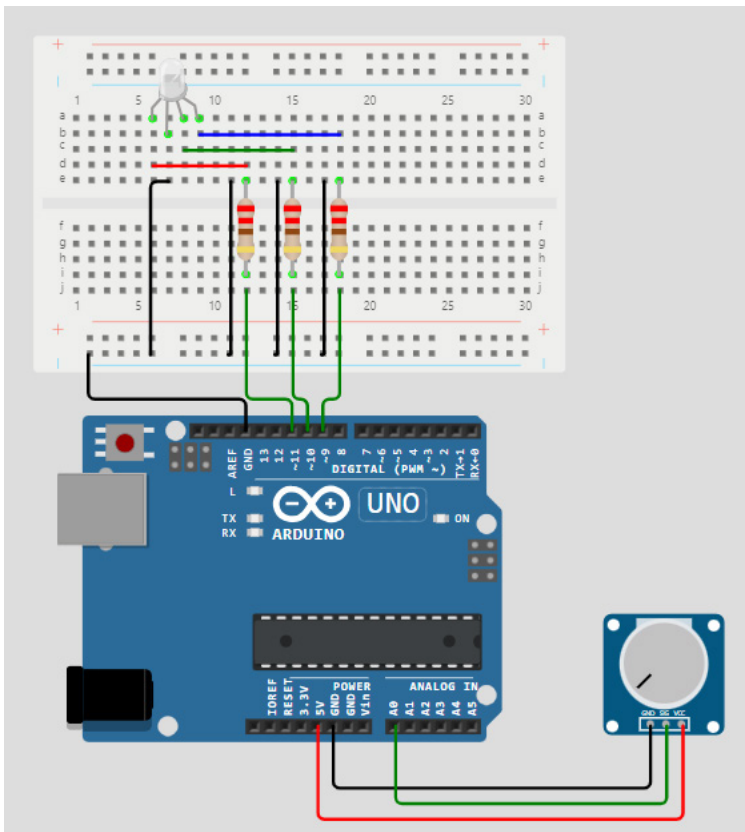


Figura 43. Control de un LED RGB con un potenciómetro.

Ejemplo 26 - Control de la velocidad de un motor (simulado con el brillo de un LED)

Este código controla un LED parpadeante cuya velocidad de parpadeo se ajusta mediante un potenciómetro. La duración del encendido y apagado del LED varía en función de la posición del potenciómetro, permitiendo experimentar con diferentes ritmos de parpadeo.

Materiales necesarios:

- 1 Placa Arduino.
- 1 Potenciómetro.
- 1 LED.
- 1 Resistor (generalmente de 220 ohmios para limitar la corriente al LED).
- Cables de conexión.

Descripción del funcionamiento:

- 1. Lectura del Potenciómetro:** El código lee el valor analógico del potenciómetro conectado al pin A0.
- 2. Mapeo del Valor:** El valor leído (que varía entre 0 y 1023) se mapea a un rango de tiempo entre 10 ms y 1000 ms, que representa cuánto tiempo el LED permanecerá encendido y apagado.
- 3. Encendido y Apagado del LED:** El LED se enciende, espera el tiempo definido por el potenciómetro y luego se apaga. Después, espera el mismo tiempo antes de repetir el ciclo.

```
int potPin = A0; // Pin del potenciómetro
int ledPin = 9; // Pin del LED
int potValue; // Almacenar el valor del potenciómetro
int delayTime; // Almacenar el tiempo de retraso

void setup() {
  pinMode(ledPin, OUTPUT); // Configura el pin del LED
  como salida
}

void loop() {
  potValue = analogRead(potPin); // Lee el valor del
  potenciómetro
  delayTime = map(potValue, 0, 1023, 10, 1000); //
  Mapea el valor a un rango de tiempo de 10ms a 1000ms
  digitalWrite(ledPin, HIGH); // Enciende el LED
  delay(delayTime); // Espera un tiempo dependiente
  del potenciómetro
  digitalWrite(ledPin, LOW); // Apaga el LED
  delay(delayTime); // Espera de nuevo el mismo tiempo
}
```

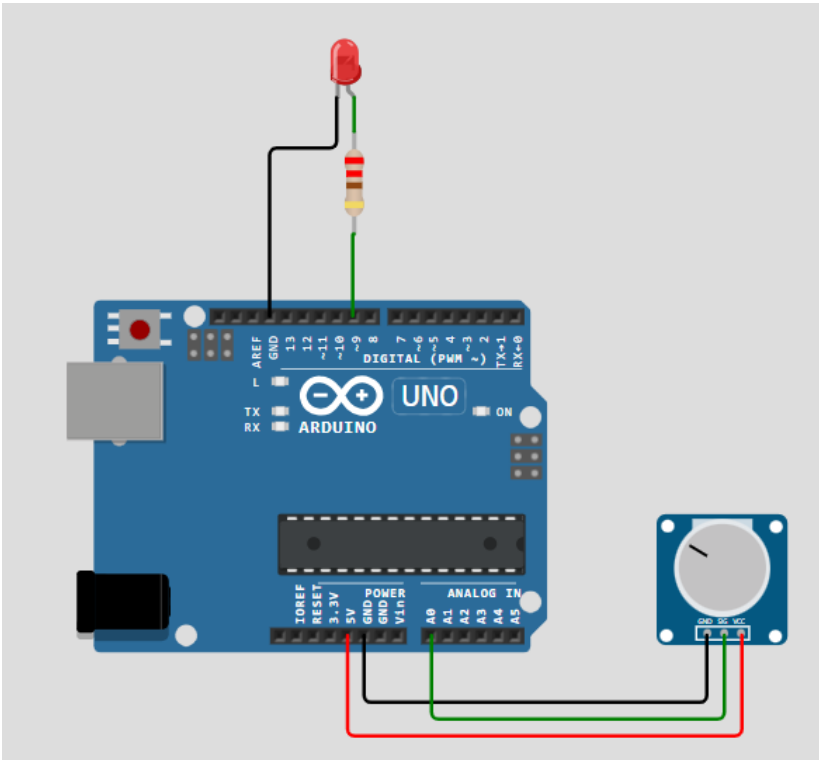



Figura 44. Control de la velocidad de un motor (simulado con el brillo de un LED).

Ejemplo 27 - Medición y visualización del valor del potenciómetro en el Monitor Serial

Este código permite leer y mostrar el valor de un potenciómetro en el Monitor Serial de Arduino. La lectura del potenciómetro se actualiza cada medio segundo, lo que permite observar cómo cambia el valor según se ajuste el potenciómetro.

Materiales necesarios:

- 1 Placa Arduino.
- 1 Potenciómetro.
- Cables de conexión.

Descripción del funcionamiento:

- 1. Inicialización de la Comunicación Serial:** En la función `setup()`, se inicializa la comunicación serial a 9600 baudios, lo que permite enviar datos desde el Arduino a la computadora.
- 2. Lectura del Potenciómetro:** En el bucle `loop()`, se lee el valor analógico del potenciómetro a través del pin A0. Este valor puede oscilar entre 0 (mínimo) y 1023 (máximo).
- 3. Impresión del Valor:** El valor leído se imprime en el Monitor Serial con un mensaje descriptivo, permitiendo al usuario ver el valor actual del potenciómetro.
- 4. Retraso entre Lecturas:** Se utiliza un `delay(500)` para esperar medio segundo antes de realizar la siguiente lectura, evitando una actualización demasiado rápida que podría dificultar la visualización de los cambios.

```

int potPin = A0; // Pin del potenciómetro
int potValue; // Variable para almacenar el valor
del potenciómetro

void setup() {
  Serial.begin(9600); // Inicializa la comunicación
serial a 9600 baudios
}

void loop() {
  potValue = analogRead(potPin); // Lee el valor del
potenciómetro (0 a 1023)
  Serial.print("Valor del potenciómetro: ");
  Serial.println(potValue); // Imprime el valor del
potenciómetro en el Monitor Serial
  delay(500); // Espera medio segundo antes de la
próxima lectura
}

```

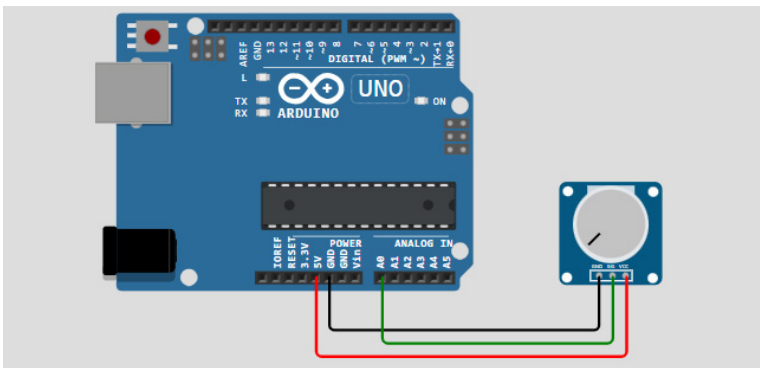


Figura 45. Medición y visualización del valor del potenciómetro en el Monitor Serial.

Ejemplo 28 – Medidor de voltaje

Mide el voltaje de una fuente externa y lo muestra en el monitor serial. Este código permite leer un valor analógico desde un sensor conectado al pin A0 de una placa Arduino y convertir ese valor en voltaje, mostrando el resultado en el Monitor Serial. La lectura se actualiza cada segundo.

Materiales necesarios:

- 1 Placa Arduino.
- 1 Sensor (por ejemplo, un potenciómetro o un sensor de luz).
- Cables de conexión.

Descripción del funcionamiento:

- 1. Inicialización de la Comunicación Serial:** En la función `setup()`, se establece la comunicación serial a 9600 baudios para permitir el envío de datos al computador.
- 2. Lectura del Sensor:** En el bucle `loop()`, se lee el valor analógico del sensor a través del pin A0. Este valor oscila entre 0 (0 V) y 1023 (5 V) debido a la resolución de 10 bits del ADC de Arduino.
- 3. Conversión a Voltaje:** El valor leído se convierte a voltaje utilizando la fórmula:

$$\text{Voltaje} = \text{sensorValue} \times \left(\frac{5.0}{1023.0} \right)$$

Esto permite obtener un valor en voltios correspondiente al valor analógico leído.

- 4. Impresión del Voltaje:** Se imprime el voltaje calculado en el Monitor Serial junto con un mensaje descriptivo.

- 5. Retraso entre Lecturas:** Se utiliza un delay(1000) para esperar un segundo antes de realizar la siguiente lectura, permitiendo al usuario ver los cambios en el voltaje en intervalos de un segundo.

```
const int sensorPin = A0; // Pin de entrada analógica

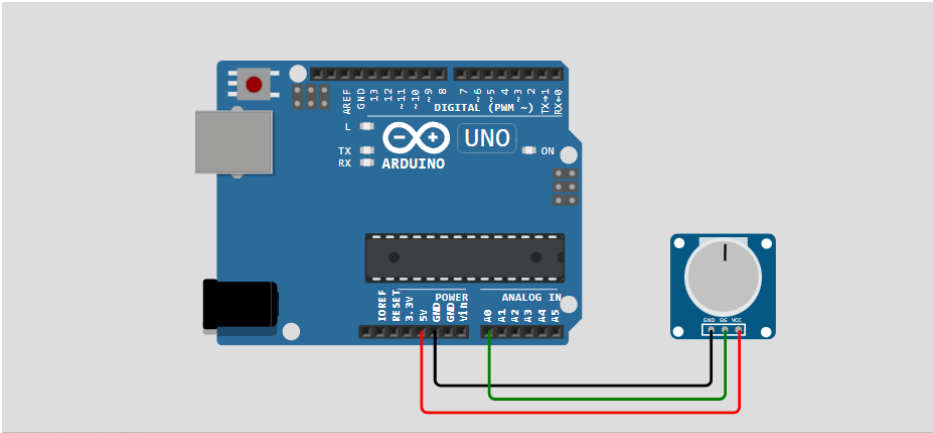
void setup() {
  Serial.begin(9600); // Inicializar el monitor serial
}

void loop() {
  int sensorValue = analogRead(sensorPin); // Leer el
valor del sensor

  float voltage = sensorValue * (5.0 / 1023.0); //
Convertir a voltaje

  Serial.print("Voltaje: ");
  Serial.println(voltage); // Imprimir el voltaje en
el monitor serial

  delay(1000); // Espera 1 segundo
}
```



```
/voltaje: 1.78  
/voltaje: 1.78  
/voltaje: 5.00  
/voltaje: 5.00
```

Figura 46. Medidor de Voltaje.

Ejemplo 29 – Semáforo controlado por potenciómetro

Este código controla un semáforo utilizando tres LEDs (rojo, amarillo y verde) y un potenciómetro para ajustar los tiempos de encendido de los LEDs verde y rojo. A continuación, se detalla el funcionamiento del código.

Materiales necesarios:

- 1 Placa Arduino.
- 3 LEDs (rojo, amarillo y verde).
- 3 resistencias (220 ohmios para los LEDs).
- 1 Potenciómetro (10k ohmios recomendado).
- Cables de conexión.

Descripción del funcionamiento:

- 1. Definición de Pines:** Se definen los pines a los que están conectados los LEDs y el potenciómetro:
 - redPin (Pin 2): LED rojo.
 - yellowPin (Pin 3): LED amarillo.
 - greenPin (Pin 4): LED verde.
 - potPin (A0): Pin del potenciómetro.
- 2. Configuración Inicial:** En la función setup(), se configuran los pines de los LEDs como salidas mediante la función pinMode().
- 3. Bucle Principal:** En el bucle loop():
 - **Lectura del Potenciómetro:** Se lee el valor del potenciómetro a través del pin A0. Este valor varía de 0 a 1023.
 - **Mapeo del Valor:** El valor leído se convierte a un rango de tiempo entre 1 segundo (1000 ms) y 5 segundos (5000 ms) utilizando la función map().
 - **Control de LEDs:**
 - **LED Verde:** Se enciende el LED verde, permaneciendo encendido durante el tiempo definido por el potenciómetro (delayTime). Después se apaga.
 - **LED Amarillo:** El LED amarillo se enciende durante un tiempo fijo de 2 segundos.
 - **LED Rojo:** Se enciende el LED rojo, permaneciendo encendido durante el mismo tiempo que el LED verde (delayTime), y luego se apaga.

```
const int redPin = 2; // Pin LED rojo
const int yellowPin = 3; // Pin LED amarillo
const int greenPin = 4; // Pin LED verde
const int potPin = A0; // Pin del potenciómetro

void setup() {
  pinMode(redPin, OUTPUT);
  pinMode(yellowPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
}

void loop() {
  int potValue = analogRead(potPin); // Leer el valor
  del potenciómetro
  int delayTime = map(potValue, 0, 1023, 1000, 5000);
  // Mapear a un rango de 1s a 5s

  digitalWrite(greenPin, HIGH);
  delay(delayTime); // Tiempo en verde
  digitalWrite(greenPin, LOW);

  digitalWrite(yellowPin, HIGH);
  delay(2000); // 2 segundos en amarillo
  digitalWrite(yellowPin, LOW);

  digitalWrite(redPin, HIGH);
  delay(delayTime); // Tiempo en rojo
  digitalWrite(redPin, LOW);
}
```

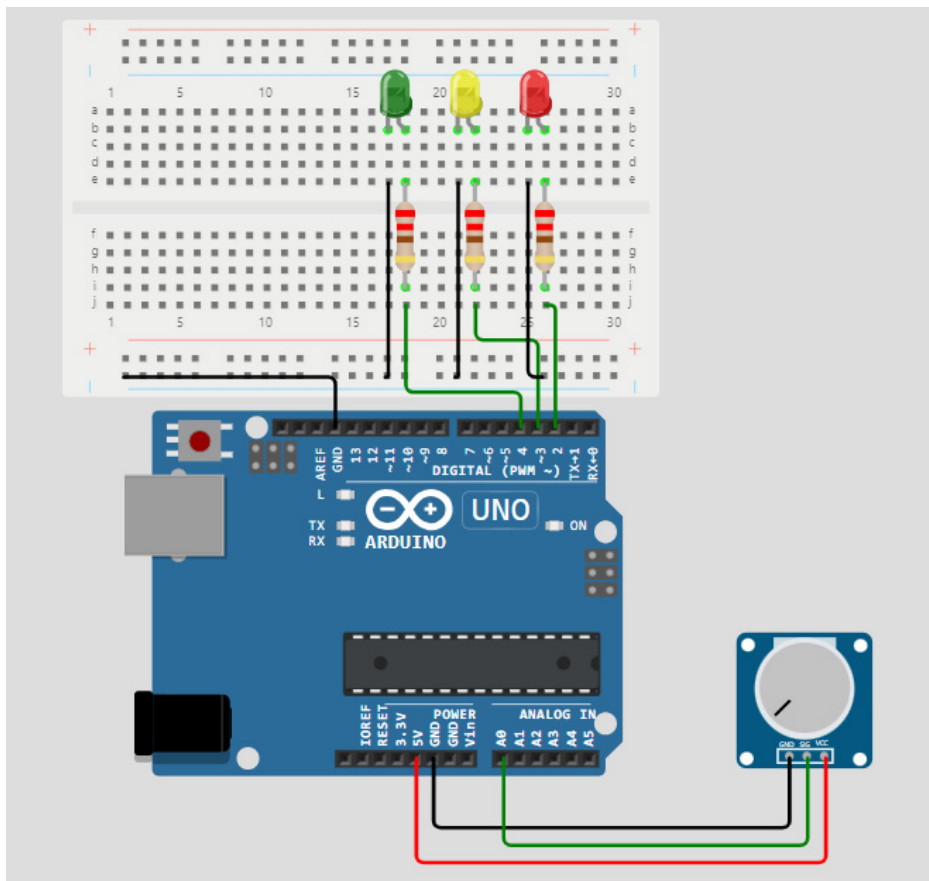



Figura 47. Semáforo controlado por Potenciómetro.

CAPÍTULO V.

Sensores y actuadores



Este capítulo destaca la importancia de incorporar sensores y actuadores en proyectos de Arduino para interactuar con el entorno y ejecutar acciones según las condiciones detectadas. Comienza con el sensor de temperatura LM35, enseñando a los lectores a interpretar datos de temperatura y a realizar acciones basadas en esta información. Luego, aborda el fotoresistor LDR para demostrar cómo la luz ambiental puede activar dispositivos como LEDs, y el sensor de movimiento PIR (HC-SR501), útil en aplicaciones de seguridad y automatización.

Se presentan también el sensor de humedad y temperatura DHT22, esencial en el monitoreo de condiciones ambientales, y el sensor ultrasónico HC-SR04, mostrando cómo medir distancias con o sin librerías. Finalmente, el capítulo incluye el teclado numérico Keypad, un actuador que facilita la entrada de datos del usuario. Como señala McLaughlin (2023), *“la habilidad para medir y responder a condiciones*

ambientales a través de sensores es lo que transforma un simple microcontrolador en una poderosa herramienta de automatización” (p. 158). Este capítulo otorga a los lectores las habilidades necesarias para integrar sensores y actuadores en proyectos de Arduino, permitiéndoles desarrollar sistemas inteligentes y responsivos.

Ejemplo 30 – Sensor de temperatura LM35

El LM35 es un sensor de temperatura analógico que proporciona una salida de voltaje proporcional a la temperatura en grados Celsius. Este sensor es ampliamente utilizado debido a su precisión y facilidad de uso. La salida del LM35 es de 10 mV por grado Celsius, lo que permite medir temperaturas en un rango de -55°C a 150°C. Su diseño simplifica el proceso de medición de temperatura en proyectos electrónicos, ya que no requiere componentes adicionales para su calibración. Además, el LM35 se caracteriza por su bajo consumo de energía y alta precisión, lo que lo hace ideal para aplicaciones de control ambiental y sistemas de monitoreo de temperatura (McKinney, D., 2021).

| Característica | Descripción |
|-----------------------|---------------------------------|
| Tipo de Sensor | Sensor de temperatura analógico |
| Rango de Temperatura | -55 °C a +150 °C |
| Salida de Voltaje | 10 mV por grado Celsius |
| Precisión | ±0.5 °C a 25 °C |
| Alimentación | 4 V a 30 V |
| Consumo de Corriente | 60 µA (típico) |
| Tipo de Conexión | 3 pines (Vcc, salida, GND) |

| Característica | Descripción |
|----------------|---|
| Aplicaciones | Medición de temperatura en entornos HVAC, sistemas de control ambiental, dispositivos portátiles. |
| Interfaz | Análogica (conectada a pines analógicos de Arduino) |

Conexión:

- VCC: Conectar al pin de 5V de Arduino.
- GND: Conectar a GND.
- VOUT: Conectar a un pin analógico (por ejemplo, A0).

```
void setup() {  
  Serial.begin(9600); // Inicializa la comunicación  
  serial  
}  
  
void loop() {  
  int valor = analogRead(A0); // Lee el valor analógico  
  float temperatura = (valor * 500.0) / 1024.0; //  
  Convierte a grados Celsius  
  Serial.print("Temperatura: ");  
  Serial.println(temperatura);  
  delay(1000); // Espera 1 segundo  
}
```


| Característica | Descripción |
|----------------------|--|
| Tipo de Sensor | Resistor dependiente de la luz (LDR) |
| Rango de Resistencia | 1 kΩ (en luz intensa) a 10 MΩ (en oscuridad) |
| Respuesta Espectral | Sensible a la luz visible (aproximadamente 400-700 nm) |
| Alimentación | No requiere alimentación externa (se conecta en un divisor de voltaje) |
| Consumo de Corriente | Bajo, depende de la resistencia del circuito |
| Tipo de Conexión | 2 pines (conectado en un divisor de voltaje) |
| Aplicaciones | Sistemas de iluminación automática, medidores de luz, dispositivos de seguridad. |
| Interfaz | Analógica (conectada a pines analógicos de Arduino) |

```

void setup() {
  Serial.begin(9600); // Inicializa la comunicación
  serial
}

void loop() {
  int valor = analogRead(A0); // Lee el valor analógico
  Serial.print("Luz: ");
  Serial.println(valor);
  delay(1000); // Espera 1 segundo
}

```

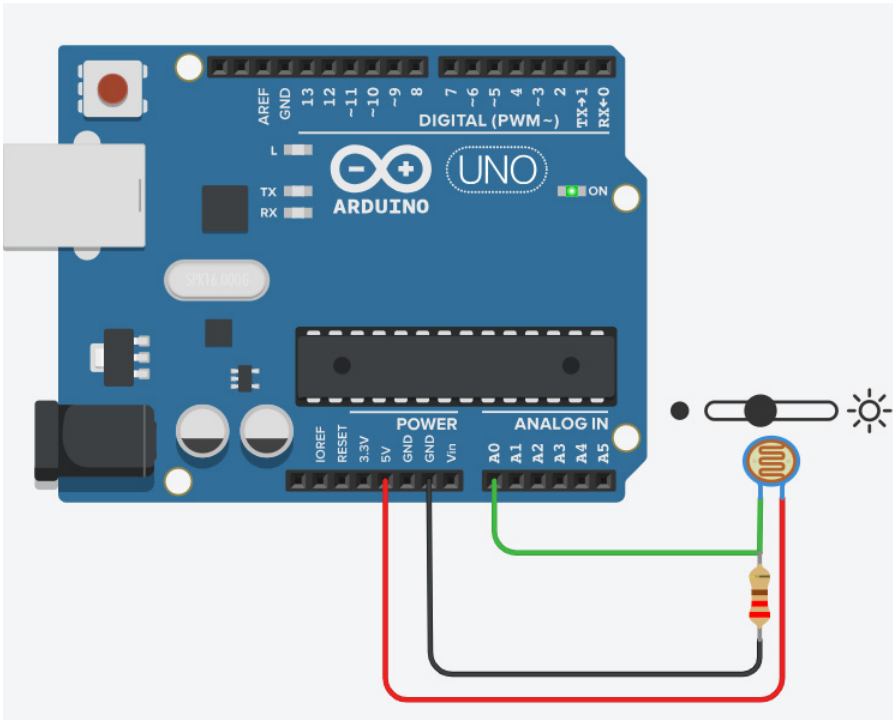


Figura 49. Sensor de luz (Fotoresistor) LDR.

Ejemplo 32 - Sensor de Movimiento PIR (HC-SR501)

El sensor de movimiento PIR (Passive Infrared) HC-SR501 es un dispositivo que detecta la radiación infrarroja emitida por objetos en movimiento, como seres humanos o animales. Este sensor se utiliza comúnmente en sistemas de seguridad y automatización, ya que puede detectar la presencia de personas en su área de alcance sin necesidad de contacto físico. El sensor funciona detectando cambios en la radiación infrarroja en su entorno; cuando un objeto caliente, como una persona, entra en su rango, el sensor envía una señal de activación a un microcontrolador o sistema asociado. El HC-SR501 tiene un alcance de hasta 7

metros y un ángulo de detección de 120 grados, lo que lo convierte en una opción popular para proyectos de automatización del hogar y seguridad. Además, su bajo consumo de energía lo hace ideal para aplicaciones donde la eficiencia energética es crucial (González, 2022).

Conexión:

- **VCC:** Conectar al pin de 5V de Arduino.
- **GND:** Conectar a GND.
- **OUT:** Conectar a un pin digital (por ejemplo, D2).

| Característica | Descripción |
|--------------------------|--|
| Tipo de Sensor | Sensor de movimiento PIR (Passive Infrared Sensor) |
| Voltaje de Alimentación | 5V a 20V |
| Rango de Detección | 3 a 7 metros (ajustable mediante potenciómetro) |
| Ángulo de Detección | 120° (horizontal) |
| Tiempo de Retardo | 5 segundos a 5 minutos (ajustable mediante potenciómetro) |
| Consumo de Corriente | 50 μ A en reposo, 65 mA en funcionamiento |
| Característica | Descripción |
| Nivel de Salida | 3.3V (señal alta) cuando detecta movimiento |
| Tiempo de Inicialización | 1 minuto aproximadamente |
| Aplicaciones | Alarmas de seguridad, iluminación automática, control de dispositivos electrónicos |
| Interfaz | Salida digital (conectada a pines digitales de Arduino) |


```
const int pinPIR = 2; // Pin digital al que está
conectado el sensor
void setup() {
  pinMode(pinPIR, INPUT); // Configura el pin como
entrada
  Serial.begin(9600); // Inicializa la comunicación
serial
}

void loop() {
  int estado = digitalRead(pinPIR); // Lee el estado
del sensor
  if (estado == HIGH) {
    Serial.println("Movimiento detectado!");
  } else {
    Serial.println("Sin movimiento.");
  }
  delay(1000); // Espera 1 segundo
}
```

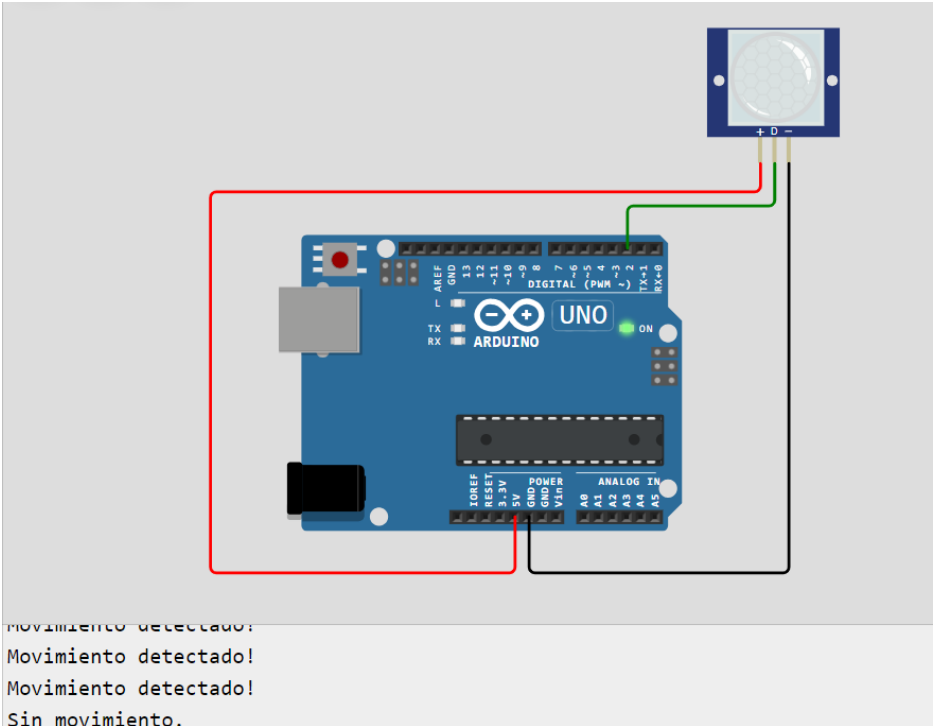


Figura 50. Sensor de Movimiento PIR (HC-SR501).

Librerías en Arduino

Las **librerías** son colecciones de código predefinido que facilitan la programación de hardware específico en el entorno de desarrollo de Arduino. Permiten a los desarrolladores usar funcionalidades complejas sin tener que escribir el código desde cero, lo que ahorra tiempo y esfuerzo.

Concepto de Librerías

Definición: Una librería es un conjunto de funciones, clases y métodos que encapsulan la complejidad del hardware y proporcionan una interfaz más fácil de usar. Se crean para simplificar tareas comunes, como la comunicación con sensores, módulos, pantallas, motores, etc.

Objetivo: Facilitar la reutilización del código y mejorar la organización del proyecto, permitiendo que el usuario se concentre en la lógica de su aplicación en lugar de la implementación de bajo nivel.

Estructura:

- Las librerías suelen consistir en archivos de código fuente (.cpp y .h) que definen las funciones y clases.
- El archivo .h (header) contiene las declaraciones de las funciones y las definiciones de las clases.
- El archivo .cpp contiene las implementaciones de las funciones.

Tipos de Librerías

Librerías de Hardware: Diseñadas para interactuar con dispositivos específicos, como sensores y módulos. Ejemplos:

- **DHT:** Para leer datos de sensores de temperatura y humedad.
- **Servo:** Para controlar servomotores.
- **Wire:** Para la comunicación I2C.
- **SPI:** Para la comunicación SPI.

Librerías de Software: Proporcionan funcionalidades adicionales, como la manipulación de cadenas, el manejo de fechas y horas, y más.

Librerías de Ejemplo: Incluyen ejemplos de cómo usar una librería específica, lo cual es útil para aprender su uso.

Beneficios de Usar Librerías

1. **Simplificación:** Hacen que el código sea más legible y fácil de entender, al abstraer las complejidades del hardware.
2. **Ahorro de Tiempo:** Reducen el tiempo de desarrollo al evitar la necesidad de escribir y depurar código para operaciones comunes.
3. **Comunidad y Soporte:** Muchas librerías son desarrolladas y mantenidas por la comunidad, lo que significa que pueden contar con actualizaciones, mejoras y soporte.
4. **Mejor Organización del Código:** Ayudan a mantener el código organizado y modular, facilitando su mantenimiento y actualización.

Cómo Usar Librerías en Arduino

Instalación:

- Las librerías se pueden instalar desde el Gestor de Librerías del IDE de Arduino.
- Ir a Sketch > Include Library > Manage Libraries.
- Buscar la librería deseada, hacer clic en “Install”.

Inclusión: Una vez instalada, se puede incluir en el código usando la directiva `#include`. Por ejemplo:

```
#include <DHT.h> // Para el sensor DHT
```

Uso: Inicializar la librería creando un objeto, por ejemplo:

```
DHT dht(2, DHT11); // Donde 2 es el pin al que está conectado el sensor
```

Llamar a las funciones de la librería para interactuar con el hardware:

```
dht.begin(); // Inicializa el sensor
```

```
float h = dht.readHumidity(); // Lee la humedad
float t = dht.readTemperature(); // Lee la temperatura
```

Documentación: Es fundamental leer la documentación de la librería, que generalmente incluye ejemplos de uso, funciones disponibles y configuraciones específicas.

En el IDE de Arduino tenemos el LIBRARY MANAGER, el cual se lo utiliza para buscar la librería necesaria entre la gran cantidad que posee, cantidad que cada vez va en aumento.

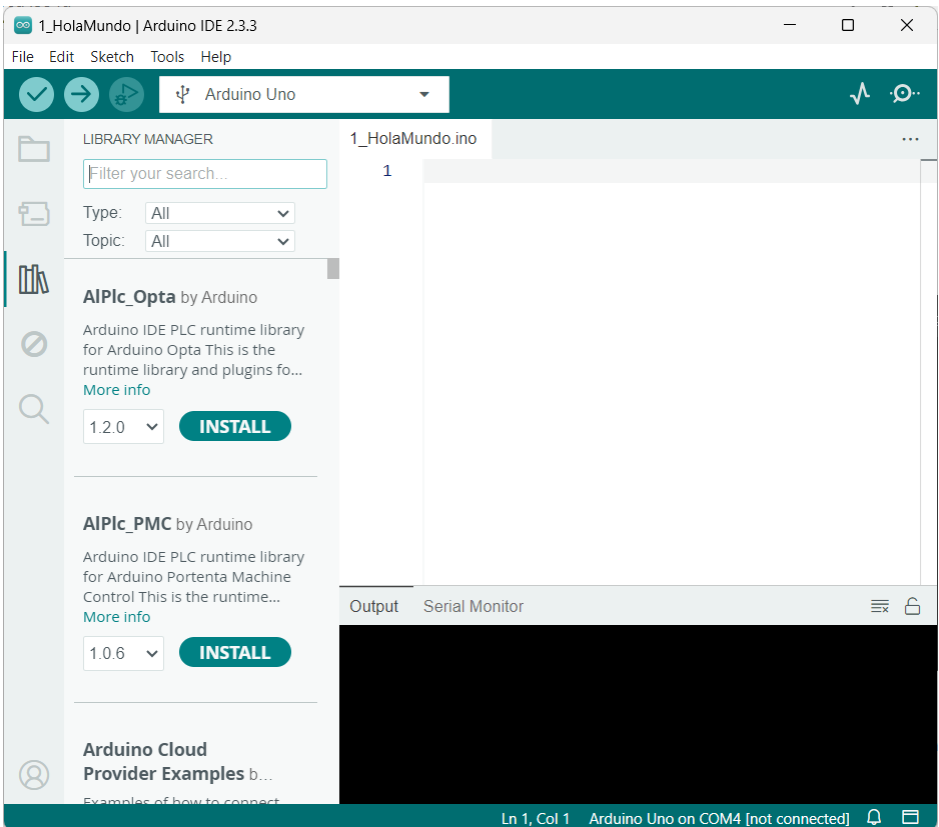


Figura 51. Sección de librerías en el Arduino IDE.

Ejemplo 33 – Sensor de humedad y temperatura DHT22

El sensor DHT22 es un dispositivo utilizado para medir la humedad relativa y la temperatura ambiente. Este sensor digital se basa en un sensor capacitivo para medir la humedad y un termistor para medir la temperatura. El DHT22 es ampliamente utilizado en proyectos de monitoreo ambiental debido a su alta precisión y facilidad de uso.

Tiene un rango de medición de temperatura de -40 a 80°C con una precisión de $\pm 0.5^{\circ}\text{C}$ y un rango de humedad de 0% a 100% con una precisión de $\pm 2-5\%$. El DHT22 es ideal para aplicaciones como estaciones meteorológicas, sistemas de control de clima y proyectos de IoT (Internet de las Cosas), ya que proporciona datos fiables para regular el ambiente en tiempo real (Martínez, 2023).

Conexiones:

- Pin 1 (VCC) del DHT22 al 5V del Arduino.
- Pin 2 (Data) del DHT22 a cualquier pin digital del Arduino (ej. Pin 2).
- Pin 3 sin conexión.
- Pin 4 (GND) al GND del Arduino.
- Coloca una resistencia de $10\text{k}\Omega$ entre VCC y Data.

| Característica | Descripción |
|--------------------------|---|
| Tipo de Sensor | Sensor de temperatura y humedad |
| Rango de Temperatura | -40°C a 80°C |
| Precisión de Temperatura | $\pm 0.5^{\circ}\text{C}$ |
| Rango de Humedad | 0% a 100% de humedad relativa |
| Precisión de Humedad | $\pm 2\%$ - 5% (según el rango de humedad) |
| Voltaje de Alimentación | 3.3V a 5.5V |
| Corriente en reposo | 0.5 mA ($500\text{ }\mu\text{A}$) |
| Corriente máxima | 2.5 mA durante la medición |

| | |
|--------------------------|---|
| Frecuencia de Medición | Una lectura cada 2 segundos |
| Interfaz de Comunicación | Salida digital de un solo cable (protocolo propio) |
| Tiempo de Respuesta | 2 segundos |
| Dimensiones | 15.1 mm x 25 mm x 7.7 mm |
| Temperatura de Operación | -40°C a 80°C |
| Humedad de Operación | 0% a 100% RH |
| Aplicaciones | Medición de clima, sistemas HVAC, monitoreo ambiental, etc. |

```

#include "DHT.h"
#define DHTPIN 2 // Pin digital donde está conectado
el sensor
#define DHTTYPE DHT22 // Tipo de sensor DHT22
DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(9600);
  dht.begin(); // Iniciar el sensor DHT22
  Serial.println("Sensor DHT22 Iniciado");
}

void loop() {
  // Esperar entre lecturas
  delay(2000);

  // Leer la humedad
  float humedad = dht.readHumidity();
  // Leer la temperatura en grados Celsius

```

```
float temperatura = dht.readTemperature();

// Verificar si la lectura fue exitosa
if (isnan(humedad) || isnan(temperatura)) {
  Serial.println("Error al leer el sensor DHT22");
  return;
}

// Imprimir los valores en el monitor serial
Serial.print("Humedad: ");
Serial.print(humedad);
Serial.print("% ");
Serial.print("Temperatura: ");
Serial.print(temperatura);
Serial.println("°C");
}
```

Explicación del Código:

- **Librerías:** Se incluye la librería DHT.h que facilita la interacción con el sensor DHT22.
- **Definiciones:** Se define el pin de datos al que está conectado el sensor y el tipo de sensor (DHT22).
- **Setup:** Se inicializa la comunicación serie a 9600 bps para ver los resultados en el monitor serial, y se activa el sensor DHT22.
- **Loop:** Cada 2 segundos, se lee la temperatura y la humedad. Si hay algún problema en la lectura, se muestra un mensaje de error. Los valores obtenidos se imprimen en el monitor serial.

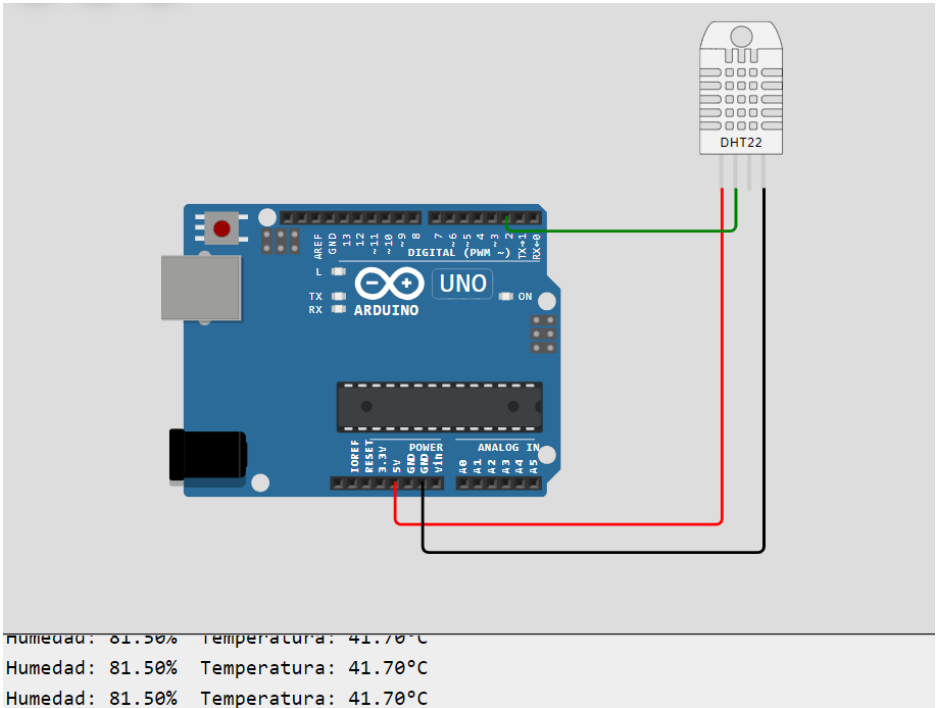


Figura 52. Sensor de humedad y temperatura DHT22.

Ejemplo 34 – Sensor ultrasónico HC-SR04 sin librería

El sensor ultrasónico HC-SR04 es un dispositivo utilizado para medir distancias mediante ondas ultrasónicas. Este sensor emite una señal ultrasónica a través de su transductor, y mide el tiempo que tarda la señal en reflejarse de vuelta desde un objeto. A partir de este tiempo, el sensor calcula la distancia entre el sensor y el objeto.

El HC-SR04 tiene un rango de medición que varía desde 2 cm hasta 400 cm, y es ampliamente utilizado en aplicaciones de

robótica, automatización y sistemas de medición de distancias, como en robots autónomos que navegan evitando obstáculos. Este sensor se caracteriza por su bajo costo, precisión y facilidad de integración con microcontroladores como Arduino (Gómez, 2022).

Conexiones:

- VCC del sensor a 5V del Arduino.
- GND del sensor al GND del Arduino.
- Trig del sensor a cualquier pin digital del Arduino (ej. Pin 9).
- Echo del sensor a otro pin digital del Arduino (ej. Pin 10).

| Características | Detalles |
|--------------------------|---|
| Modelo | HC-SR04 |
| Voltaje de operación | 5V DC |
| Corriente de operación | 15 mA |
| Frecuencia de operación | 40 kHz |
| Ángulo de detección | 15 grados |
| Rango de detección | 2 cm - 400 cm |
| Precisión | ± 3 mm |
| Tiempo de respuesta | 100 ms |
| Dimensiones | 45 mm x 20 mm x 15 mm |
| Características | Detalles |
| Pines de conexión | VCC, GND, Trig, Echo |
| Temperatura de operación | -15°C a 70°C |
| Método de medición | Tiempo de vuelo de un pulso ultrasónico |
| Velocidad de sonido | 0.034 cm/μs (usado para calcular distancia) |

```
// Definir pines
const int trigPin = 9;
const int echoPin = 10;

// Variable para la duración y la distancia
long duracion;
int distancia;

void setup() {
  // Configuración de los pines
  pinMode(trigPin, OUTPUT); // Pin trig como salida
  pinMode(echoPin, INPUT); // Pin echo como entrada

  // Iniciar la comunicación serial
  Serial.begin(9600);
}

void loop() {
  // Asegurarse de que el pin Trig esté bajo
  digitalWrite(trigPin, LOW);
```

```
delayMicroseconds(2);

// Enviar un pulso de 10 microsegundos
digitalWrite(trigPin, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin, LOW);

// Leer el tiempo que tarda el eco en volver
duracion = pulseIn(echoPin, HIGH);

// Calcular la distancia (duración / 2) / velocidad
del sonido (0.034 cm/us)
distancia = duracion * 0.034 / 2;
// Imprimir la distancia en el monitor serial
Serial.print("Distancia: ");

Serial.print(distancia);
Serial.println(" cm");

// Esperar un segundo antes de la siguiente lectura
delay(1000);
```

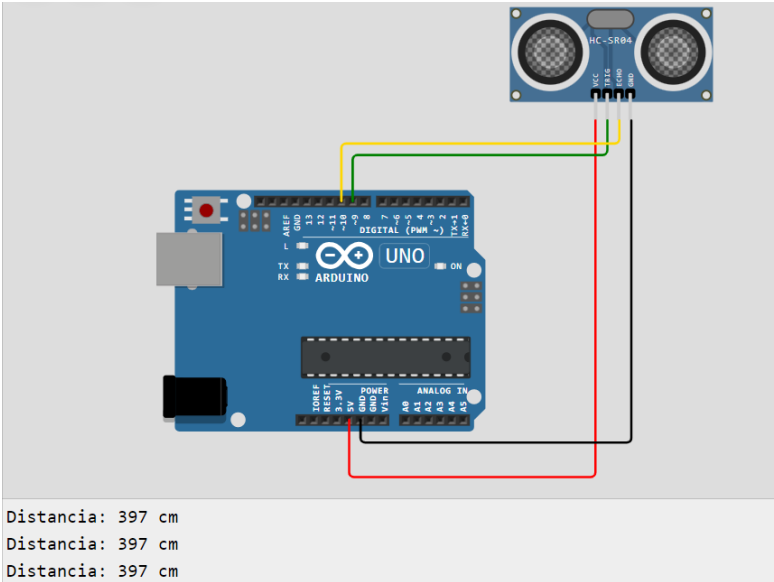


Figura 53. Sensor Ultrasonico HC-SR04 SIN Librería.

Ejemplo 35 – Sensor ultrasónico HC-SR04 con librería

Instalación de la librería "NewPing":

1. Abra el Arduino IDE.
2. Diríjase a **Herramientas > Administrar bibliotecas**.
3. Busque la librería **NewPing** en el cuadro de búsqueda.
4. Seleccione la librería y haz clic en **Instalar**.

Conexión del HC-SR04:

- VCC → 5V (Arduino)
- GND → GND (Arduino)
- Trig → Pin 9 (Arduino)
- Echo → Pin 10 (Arduino)

```
#include <NewPing.h>

#define TRIGGER_PIN 9 // Pin para el Trig del HC-SR04
#define ECHO_PIN 10 // Pin para el Echo del HC-SR04
#define MAX_DISTANCE 200 // Distancia máxima en cm
para las lecturas

NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE);
// Inicializa el sensor

void setup() {
  Serial.begin(9600); // Inicia la comunicación serial
}

void loop() {
  delay(50); // Pausa breve para evitar sobrecargar
el sensor

  unsigned int distancia = sonar.ping_cm(); // Obtiene
la distancia en cm

  Serial.print("Distancia: ");
  Serial.print(distancia);
  Serial.println(" cm");
}
```

Explicación del código:

- **Librería NewPing:** Se utiliza para simplificar el manejo de los pines del sensor HC-SR04 y calcular automáticamente la distancia.
- **TRIGGER_PIN y ECHO_PIN:** Son las conexiones del sensor a los pines del Arduino.
- **MAX_DISTANCE:** Define el rango máximo de distancia que puede detectar el sensor (en este caso, 200 cm).

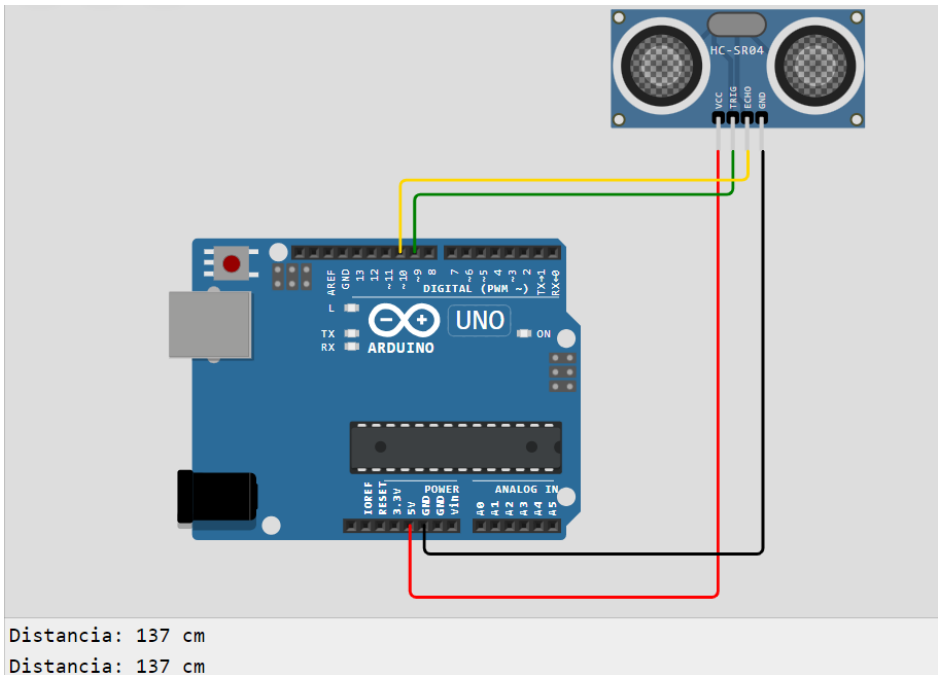


Figura 54. Sensor Ultrasónico HC-SR04 CON Librería.

Ejemplo 36 – Teclado numérico Keypad

Instalación de la Librería Keypad:

1. Abra el **Arduino IDE**.

2. Vaya a **Herramientas > Administrar bibliotecas**.
3. En la ventana de administración de bibliotecas, buscar **Keypad**.
4. Seleccione la librería y haga clic en **Instalar**.

Conexión del Keypad 4x4:

Para este ejemplo, asume un teclado matricial de 4x4 (puedes adaptar el código si es de otro tipo).

Conecte los pines del teclado a los siguientes pines del Arduino:

- **Filas** → pines 9, 8, 7, 6
- **Columnas** → pines 5, 4, 3, 2

```
#include <Keypad.h>

// Definir el tamaño del teclado (4 filas y 4 columnas
// en este caso)
const byte FILAS = 4; // Número de filas en el Keypad
const byte COLUMNAS = 4; // Número de columnas en el
Keypad

// Definir la distribución del Keypad (caracteres que
// tiene cada botón)
char teclas[FILAS][COLUMNAS] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};
```



```

// Conectar los pines de las filas y columnas del
teclado a los pines de Arduino
byte pinesFilas[FILAS] = {9, 8, 7, 6}; // Pines a los
que se conectan las filas
byte pinesColumnas[COLUMNAS] = {5, 4, 3, 2}; // Pines
a los que se conectan las columnas

// Crear el objeto Keypad
Keypad teclado = Keypad(makeKeymap(teclas),
pinesFilas, pinesColumnas, FILAS, COLUMNAS);

void setup() {
  Serial.begin(9600); // Iniciar comunicación serial
  Serial.println("Teclado listo, presiona una tecla:");
}

void loop() {
  char tecla = teclado.getKey(); // Leer si se ha
presionado alguna tecla

  if (tecla) { // Si se presiona una tecla
    Serial.print("Tecla presionada: ");
    Serial.println(tecla);
  }
}

```

Explicación del código:

- **Keypad.h:** Es la librería que facilita el uso de teclados matriciales.
- **teclas[FILAS][COLUMNAS]:** Esta matriz define los caracteres asociados a cada botón del teclado.
- **pinosFilas y pinosColumnas:** Son los pines de Arduino a los que están conectados los cables de las filas y columnas del teclado.
- **teclado.getKey():** Esta función detecta si se ha presionado una tecla y devuelve su valor.
- **Serial.println(tecla):** Muestra la tecla presionada en el Monitor Serial.

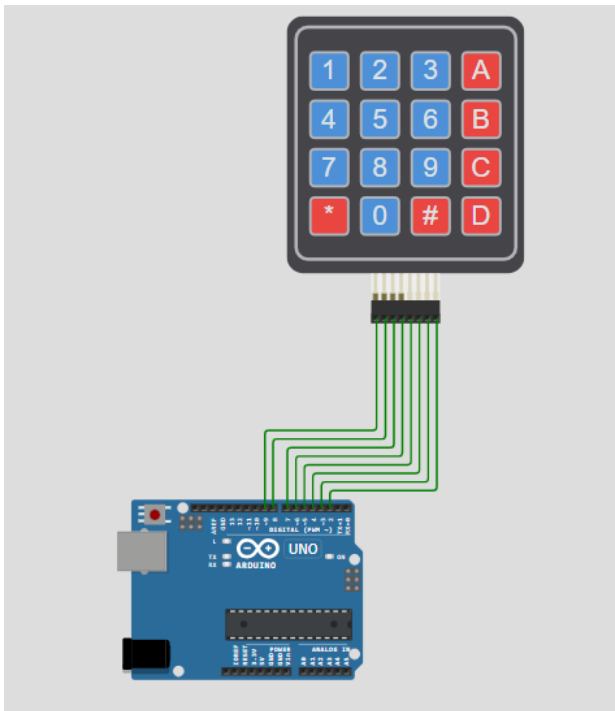


Figura 55. Teclado Numérico Keypad.

CAPÍTULO VI.

Control de motores DC



Este capítulo se centra en el control de motores, una de las aplicaciones más prácticas de Arduino, que permite crear proyectos desde robots móviles hasta sistemas automatizados. Comienza con un ejemplo básico de control de un motor de 5V, seguido de la introducción al Arduino Motor Shield, que facilita la conexión y control de múltiples motores. También se explora el uso del módulo L298N, un controlador de motor que permite gestionar dirección y velocidad mediante PWM, brindando un mayor control sobre los motores.

Además, se abordan los servomotores, fundamentales en robótica, y se enseña cómo controlarlos con un potenciómetro para ajustar su posición. Finalmente, se presenta el control de motores a pasos, ideales para movimientos precisos en aplicaciones como impresoras 3D y robótica. Como señala Wilson (2022), *“el control de motores es una puerta de entrada hacia la robótica moderna, donde el movimiento se convierte en una forma de*

expresión tecnológica” (p. 204). Este capítulo proporciona a los lectores las habilidades necesarias para integrar motores DC en sus proyectos, creando sistemas interactivos e inteligentes.

Ejemplo 37 – Control de motor de 5V

Para controlar un motor de corriente continua (DC) de 5V con un Arduino, puedes usar un transistor como interruptor y un diodo para proteger el circuito. A continuación, se detalla un ejemplo completo que incluye la conexión del hardware, el código y una explicación.

Materiales Necesarios

- **Arduino Uno** (o cualquier otra placa de Arduino)
- **Motor DC** de 5V
- **Transistor NPN** (por ejemplo, 2N2222 o TIP120)
- **Diodo** (1N4001 o similar)
- **Resistor** (1 k Ω para la base del transistor)
- **Fuente de alimentación de 5V** (batería o adaptador)
- **Cables de conexión**
- **Protoboard** (opcional, pero recomendado para pruebas)

Conexiones

1. Conecte el motor:
 - Conecte un terminal del motor al colector del transistor.
 - Conecte el otro terminal del motor a la fuente de alimentación de 5V.
2. Conecte el transistor:
 - Conecte el emisor del transistor a tierra (GND).

- Conecte un resistor de 1 kΩ entre la base del transistor y uno de los pines digitales del Arduino (por ejemplo, el pin 9).

3. Conecte el diodo:

- Conecte el cátodo del diodo (el lado con la línea) al terminal del motor que va al colector del transistor.
- Conecte el ánodo del diodo a tierra (GND).

4. Conexiones de alimentación:

- Asegúrese de que el GND del Arduino esté conectado al GND de la fuente de alimentación.

```
const int motorPin = 9; // Pin donde está conectado
el transistor

void setup() {
  pinMode(motorPin, OUTPUT); // Configura el pin como
salida
}

void loop() {
  digitalWrite(motorPin, HIGH); // Enciende el motor
  delay(1000); // Espera 1 segundo
  digitalWrite(motorPin, LOW); // Apaga el motor
  delay(1000); // Espera 1 segundo
}
```

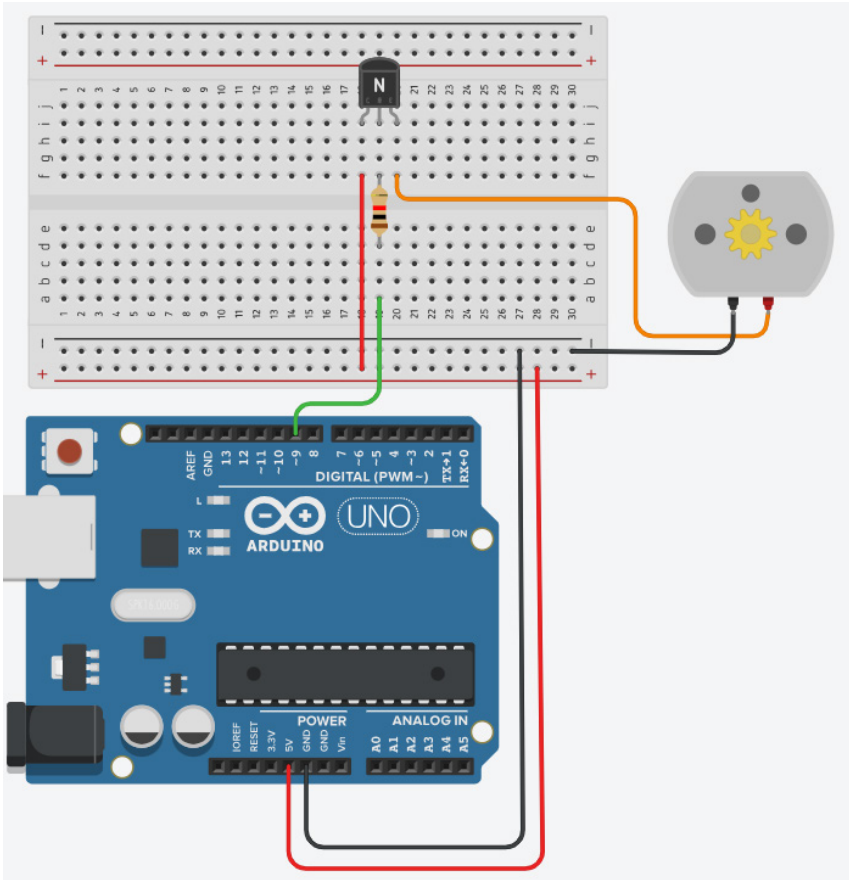


Figura 56. Control de motor de 5V.

Arduino Motor Shield

El Arduino Motor Shield Rev3 es una herramienta versátil y fácil de usar para proyectos de control de motores con Arduino. Proporciona un control eficiente de la velocidad y dirección de motores DC, motores paso a paso, y otros actuadores con capacidad de alta corriente. Su simplicidad y compatibilidad lo hacen ideal para quienes desean integrar motores en sus proyectos de robótica o automatización.

Características Principales del Arduino Motor Shield

1. Basado en el chip L298:
 - El Arduino Motor Shield Rev3 está basado en el controlador de doble puente H **L298**, un IC diseñado para controlar la dirección y velocidad de motores. El L298 permite el control de dos motores DC o un motor paso a paso.
2. Control de dos canales:
 - Tiene dos canales, **A** y **B**, que permiten controlar **dos motores DC** de manera independiente o un **motor paso a paso**.
 - Cada canal puede controlar la dirección del motor y su velocidad mediante la modulación por ancho de pulso (PWM).
3. Corriente de salida:
 - La corriente máxima por canal es de **2A**, lo que es suficiente para la mayoría de los motores pequeños. Si se requiere más corriente, es posible usar una fuente de alimentación externa conectada directamente a la shield.
4. Protección térmica y de sobrecorriente:
 - Incluye protección para evitar daños por sobrecalentamiento o sobrecorriente, lo que es especialmente útil para evitar la quema del chip o la placa.
5. Compatibilidad con fuentes de alimentación externas:
 - Puede alimentar los motores con una fuente externa conectada al terminal de alimentación de la shield. Esto es útil para motores que requieren más voltaje y corriente que lo que el Arduino puede suministrar.

6. Conexión directa con el Arduino:

- Se conecta directamente a la placa Arduino como una shield, aprovechando los pines ya existentes sin necesidad de cables adicionales.

Entradas y Salidas en el Motor Shield

1. Conexiones para motores:

- Los motores se conectan directamente a los terminales **A+ y A-** para el motor **A**, y **B+ y B-** para el motor **B**. Estos terminales permiten controlar la dirección y velocidad del motor.

2. Pines de control:

- El Motor Shield utiliza varios pines del Arduino para controlar los motores:
 - **Dirección:** Los pines digitales del Arduino son responsables de controlar la dirección de los motores.
 - **Velocidad:** Los pines PWM controlan la velocidad de los motores.

3. Alimentación externa:

- Cuando los motores requieren más potencia, se puede conectar una fuente de alimentación externa en el conector de entrada de la shield. Esto alimentará tanto los motores como el Arduino si se ajusta el jumper PWR.

Funcionalidades Clave

• **Control de motores DC:**

- Permite controlar la dirección (adelante y atrás) y la velocidad de hasta dos motores DC de manera independiente.

- **Control de motores paso a paso:**

- Puede manejar un motor paso a paso, controlando cada paso con precisión.

- **Uso de fuentes de alimentación externa:**

- Si el motor requiere más voltaje o corriente de la que el Arduino puede suministrar, puedes conectar una fuente de alimentación externa directamente al shield.

Ejemplo 38 – Control de motor DC con Arduino Shield

Este ejemplo básico muestra cómo controlar un motor DC usando el Arduino Motor Shield Rev3. El código te permite controlar la dirección del motor y la velocidad usando PWM.

```
// Definir pines para el canal A del motor
const int motorDirA = 12; // Pin para la dirección
del motor A
const int motorSpeedA = 3; // Pin para el control de
velocidad (PWM) del motor A

void setup() {
  // Configurar los pines como salida
  pinMode(motorDirA, OUTPUT);
  pinMode(motorSpeedA, OUTPUT);

  Serial.begin(9600); // Para monitorear en el monitor
serial
}
```

```
void loop() {  
  // Gira el motor hacia adelante  
  digitalWrite(motorDirA, HIGH); // Establece la  
  dirección  
  analogWrite(motorSpeedA, 255); // Máxima velocidad  
  Serial.println("Motor adelante a velocidad máxima");  
  delay(2000); // Esperar 2 segundos  
  
  // Gira el motor hacia atrás  
  digitalWrite(motorDirA, LOW); // Cambia la dirección  
  analogWrite(motorSpeedA, 255); // Máxima velocidad  
  Serial.println("Motor atrás a velocidad máxima");  
  delay(2000); // Esperar 2 segundos  
  
  // Detener el motor  
  analogWrite(motorSpeedA, 0); // Detener el motor  
  Serial.println("Motor detenido");  
  delay(2000); // Esperar 2 segundos  
}
```

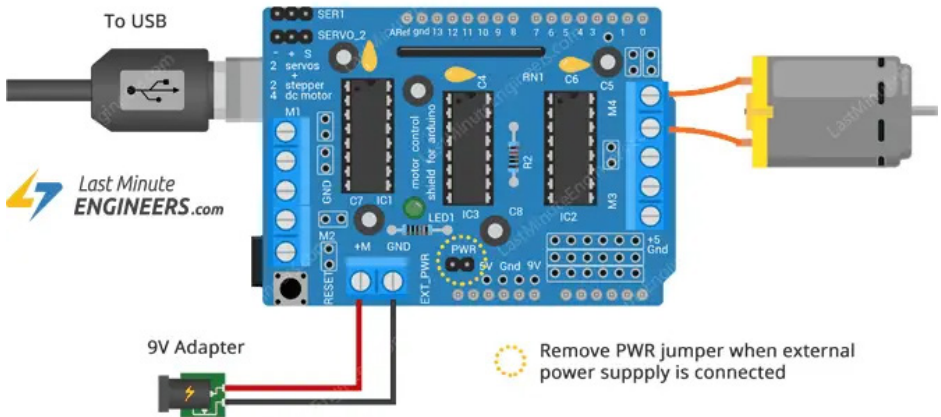


Figura 57. Arduino Shield.

Fuente: Techmake (2020).

El Arduino Shield no se encuentra disponible en los simuladores, sin embargo, el código del ejemplo 38 puede ser usado para este controlador.

¿Cómo funciona?

Control de la dirección: La dirección del motor se controla usando el pin 12 (motorDirA). Cambiar el estado de este pin de HIGH a LOW invierte la dirección del motor.

Control de la velocidad: El pin 3 (motorSpeedA) usa una señal PWM para ajustar la velocidad del motor. El valor de PWM varía entre 0 (apagado) y 255 (máxima velocidad).

Ejemplo 39 – Control de motor DC con L298N

El **L298N** es un controlador de motores de doble puente H que permite controlar la dirección y velocidad de hasta dos motores DC. Es ideal para controlar motores en proyectos de robótica usando un **Arduino**. A continuación, se explica cómo utilizar el **L298N** para controlar un motor DC.

Conexiones del L298N

- **Motor A:** Se conecta a los pines **OUT1** y **OUT2**.
- **Motor B:** Se conecta a los pines **OUT3** y **OUT4**.
- **Alimentación del motor:** Se conecta al pin **12V** (si el motor requiere más de 5V).
- **GND:** Conecta la tierra al pin **GND**.
- **5V (opcional):** El L298N tiene un regulador de voltaje, por lo que puedes alimentar el Arduino desde aquí conectando al pin **5V**.

Para controlar la dirección y velocidad del motor, se usan los siguientes pines:

- **IN1** y **IN2** para el motor A.
- **IN3** y **IN4** para el motor B.
- **ENA** y **ENB** son los pines de habilitación (control de velocidad mediante PWM).

Conexiones para el Motor A

| L298N Pin | Arduino Pin |
|-----------|-------------|
| IN1 | Pin 7 |
| IN2 | Pin 6 |
| ENA | Pin 9 (PWM) |
| GND | GND |

```
// Pines para el motor A (L298N)
const int IN1 = 7; // Pin de dirección 1 para el
motor A
const int IN2 = 6; // Pin de dirección 2 para el
motor A
const int ENA = 9; // Pin de velocidad (PWM) para el
motor A

// Pin para el potenciómetro
const int potPin = A0; // Pin analógico para leer el
valor del potenciómetro

void setup() {
  // Configurar los pines como salida
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(ENA, OUTPUT);

  Serial.begin(9600); // Para monitorear el valor del
potenciómetro
}
void loop() {
  int potValue = analogRead(potPin); // Leer el valor
del potenciómetro (0-1023)
  int motorSpeed = map(potValue, 0, 1023, 0, 255); //
Convertirlo a valor de PWM (0-255)
```

```
// Mostrar el valor del potenciómetro y la velocidad
del motor en el monitor serie
Serial.print("Pot value: ");
Serial.print(potValue);
Serial.print(" - Motor speed: ");
Serial.println(motorSpeed);

// Controlar el motor hacia adelante
digitalWrite(IN1, HIGH);
digitalWrite(IN2, LOW);
analogWrite(ENA, motorSpeed); // Ajustar la velocidad
del motor

delay(100); // Pequeño retraso para estabilidad
}
```

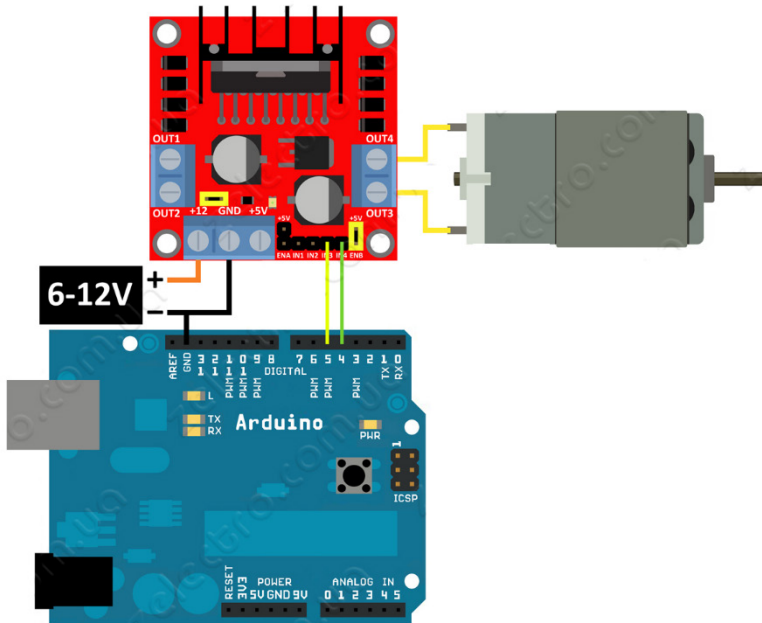


Figura 58. Control de motor con el driver L298N.
Fuente: Cruz (2014).

El driver L298N no se encuentra disponible en ninguno de los simuladores proporcionados en este libro, la imagen de conexión es referencial para el armado real, pero no se puede hacer su simulación.

Explicación del Código

1. Dirección del Motor:

- IN1 = HIGH y IN2 = LOW: El motor gira hacia adelante.
- Invertir los valores haría que el motor girara hacia atrás.

2. Control de la Velocidad:

- Usamos `analogWrite()` para aplicar una señal PWM al pin de habilitación (**ENA**) del motor, lo que ajusta su velocidad.
- El valor del potenciómetro, que varía entre 0 y 1023, se convierte a un rango adecuado de PWM (0 a 255) con la función `map()`.

3. Uso del Potenciómetro:

- El potenciómetro conectado al pin **A0** se usa para variar la velocidad del motor. Cuanto más se gira el potenciómetro, mayor es la velocidad.

Ejemplo 40 – Servomotor

Aquí tienes un ejemplo simple de cómo controlar un **servomotor** con un **Arduino** usando la librería **Servo.h**. Este código permite mover el servomotor a diferentes ángulos.

Conexiones:

- **Cable de señal del servomotor:** Al pin digital **9** del Arduino.
- **Cable de alimentación del servomotor (rojo):** Al pin **5V** del Arduino.
- **Cable de tierra del servomotor (negro/marrón):** Al pin **GND** del Arduino.


```
#include <Servo.h> // Incluir la librería Servo

Servo myServo; // Crear un objeto Servo

int pos = 0; // Variable para almacenar la posición
del servomotor

void setup() {
  myServo.attach(9); // Asignar el servomotor al pin 9
}

void loop() {
  // Hacer que el servomotor se mueva de 0 a 180 grados
  for (pos = 0; pos <= 180; pos += 1) { // Moverse de
0 a 180 grados
    myServo.write(pos); // Enviar la posición al
servomotor
    delay(15); // Esperar 15 ms para que el motor se
mueva
  }
}
```

```

// Ahora mover de 180 a 0 grados
for (pos = 180; pos >= 0; pos -= 1) { // Moverse de
180 a 0 grados
    myServo.write(pos); // Enviar la posición al
servomotor
    delay(15); // Esperar 15 ms para que el motor se
mueva
}
}

```

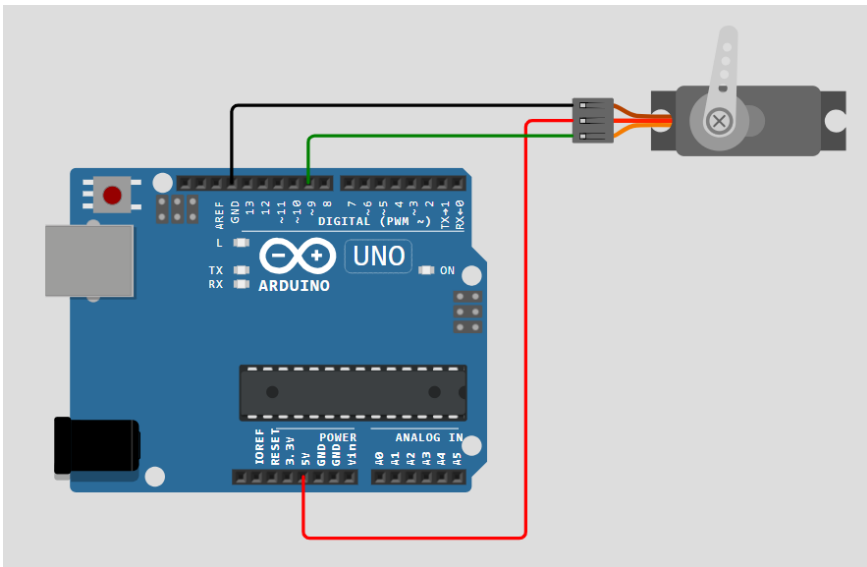


Figura 59. Control de servomotor sencillo.

Explicación del Código:

- 1. Librería Servo:** Se incluye la librería **Servo.h**, que permite controlar fácilmente servomotores.
- 2. Objeto Servo:** Se crea un objeto myServo para controlar el servomotor.
- 3. Pin de Control:** En el setup(), se asigna el pin 9 del Arduino al servomotor usando myServo.attach(9).
- 4. Movimiento del Servomotor:**
 - En el loop(), el código hace que el servomotor se mueva gradualmente de **0 grados a 180 grados** y luego regrese a **0 grados**.
 - La función myServo.write(pos) establece la posición del servomotor.
 - El delay(15) asegura que el servomotor tenga tiempo suficiente para moverse antes de recibir el siguiente comando.

Ejemplo 41 – Servomotor con potenciómetro

En este código, el valor del potenciómetro se lee para determinar el ángulo al que se moverá el servomotor.

Conexiones:

1. Servomotor:

- **Cable de señal del servomotor:** Al pin digital 9 del Arduino.
- **Cable de alimentación del servomotor (rojo):** Al pin **5V** del Arduino.
- **Cable de tierra del servomotor (negro/marrón):** Al pin **GND** del Arduino.

2. Potenciómetro:

- **Terminal 1 (izquierda):** Al pin **5V** del Arduino.
- **Terminal 2 (centro):** Al pin **A0** (entrada analógica) del Arduino.
- **Terminal 3 (derecha):** Al pin **GND** del Arduino.

```
#include <Servo.h> // Incluir la librería Servo

Servo myServo; // Crear un objeto Servo
int potPin = A0; // Pin del potenciómetro (entrada analógica A0)
int potValue; // Variable para almacenar el valor del potenciómetro
int angle; // Variable para almacenar el ángulo del servomotor

void setup() {
  myServo.attach(9); // Asignar el servomotor al pin 9
  Serial.begin(9600); // Inicializar la comunicación serial para depuración
}

void loop() {
  potValue = analogRead(potPin); // Leer el valor del potenciómetro (0 a 1023)
  angle = map(potValue, 0, 1023, 0, 180); // Convertir el valor a un rango de 0 a 180 grados
  myServo.write(angle); // Mover el servomotor al ángulo correspondiente
  delay(15); // Esperar para que el motor se mueva
}
```

Explicación del Código:

- 1. Librería Servo:** Se utiliza la librería **Servo.h** para simplificar el control del servomotor.
- 2. Pin del Potenciómetro:** Se define el pin **A0** como la entrada del potenciómetro.
- 3. Lectura del Potenciómetro:**
 - Se usa `analogRead(potPin)` para leer el valor del potenciómetro, el cual devuelve un número entre **0 y 1023**.
- 4. Mapeo del Rango:**
 - Se utiliza la función `map()` para convertir el valor del potenciómetro (0-1023) en un ángulo entre **0 y 180 grados** para el servomotor.
- 5. Movimiento del Servomotor:**
 - `myServo.write(angle)` mueve el servomotor al ángulo calculado.
 - `delay(15)` asegura que el motor tenga tiempo suficiente para moverse al nuevo ángulo.

Funcionamiento:

- Cuando giras el potenciómetro, el valor leído por el pin **A0** cambia.
- Este valor se convierte en un ángulo (0-180 grados) y el servomotor se moverá a esa posición en función de la rotación del potenciómetro.

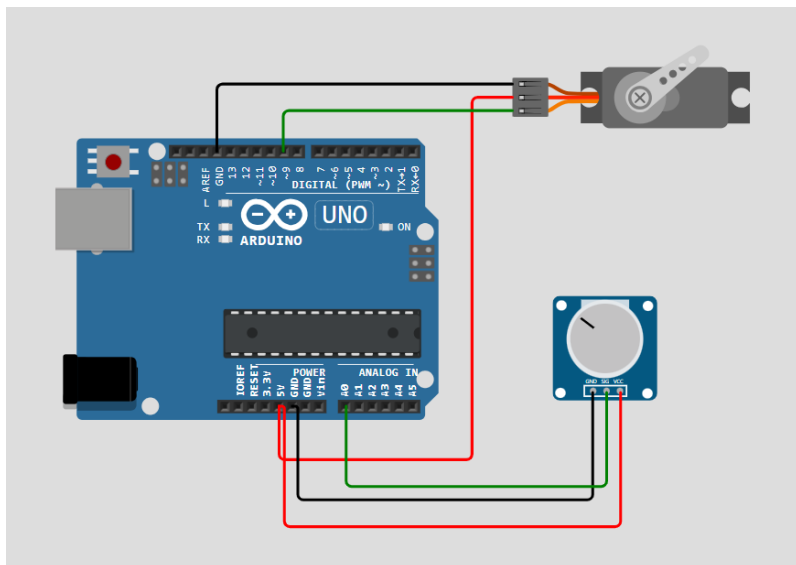


Figura 60. Servomotor con potenciómetro.

Ejemplo 42 – Motor a pasos

El **A4988** es un controlador de motor a pasos ampliamente utilizado para controlar motores paso a paso de manera precisa, y se puede conectar fácilmente a un **Arduino**. A continuación, se mostrará un ejemplo para controlar un motor a pasos utilizando el **A4988**.

Materiales necesarios:

- Arduino UNO.
- Driver de motor paso a paso A4988.
- Motor a pasos (NEMA 17, por ejemplo).
- Fuente de alimentación externa para el motor (dependiendo de las especificaciones de tu motor).
- Cables de conexión.

Conexiones:

1. A4988 Driver:

- **VMOT:** Conecte a la alimentación del motor (por ejemplo, 12V o 24V).
- **GND (Tierra):** Conecte a la tierra de la fuente de alimentación del motor y la tierra del Arduino.
- **VDD:** Conecte al pin de 5V del Arduino.
- **GND (Tierra):** Conecte al pin GND del Arduino.
- **STEP:** Conecte al pin **3** del Arduino.
- **DIR:** Conecte al pin **4** del Arduino.
- **MS1, MS2, MS3:** Deje desconectados si no quiere cambiar la microstepping (por defecto es un paso completo).
- **Enable (EN):** Opcional. Conecte a GND para habilitar el driver.
- **A1, A2, B1, B2:** Conecte los cables del motor paso a paso (dependiendo del motor, consulta la hoja de datos para saber cómo conectarlos).

```
// Definir pines para STEP y DIR
const int stepPin = 3;
const int dirPin = 4;

void setup() {
  // Configurar los pines de dirección y paso como
  salida
  pinMode(stepPin, OUTPUT);
  pinMode(dirPin, OUTPUT);
}
```



```

// Configurar la dirección inicial
digitalWrite(dirPin, HIGH); // Dirección hacia
adelante

// Iniciar la comunicación serial para monitorear
Serial.begin(9600);
}

void loop() {
// Girar en una dirección
Serial.println("Girando en una dirección");
digitalWrite(dirPin, HIGH); // Establecer la
dirección

// Generar pulsos en el pin STEP para mover el motor
for (int x = 0; x < 200; x++) { // 200 pasos para
una revolución completa (ajustar según motor)
digitalWrite(stepPin, HIGH);
delayMicroseconds(1000); // Controlar la velocidad
con un retraso pequeño
digitalWrite(stepPin, LOW);
delayMicroseconds(1000);
}
delay(1000); // Esperar un segundo

```

```
// Cambiar la dirección y girar en sentido contrario
Serial.println("Girando en dirección opuesta");
digitalWrite(dirPin, LOW); // Establecer la dirección
opuesta

for (int x = 0; x < 200; x++) { // Girar en la
dirección opuesta
digitalWrite(stepPin, HIGH);
delayMicroseconds(1000); // Controlar la velocidad
digitalWrite(stepPin, LOW);
delayMicroseconds(1000);
}
delay(1000); // Esperar un segundo
}
```

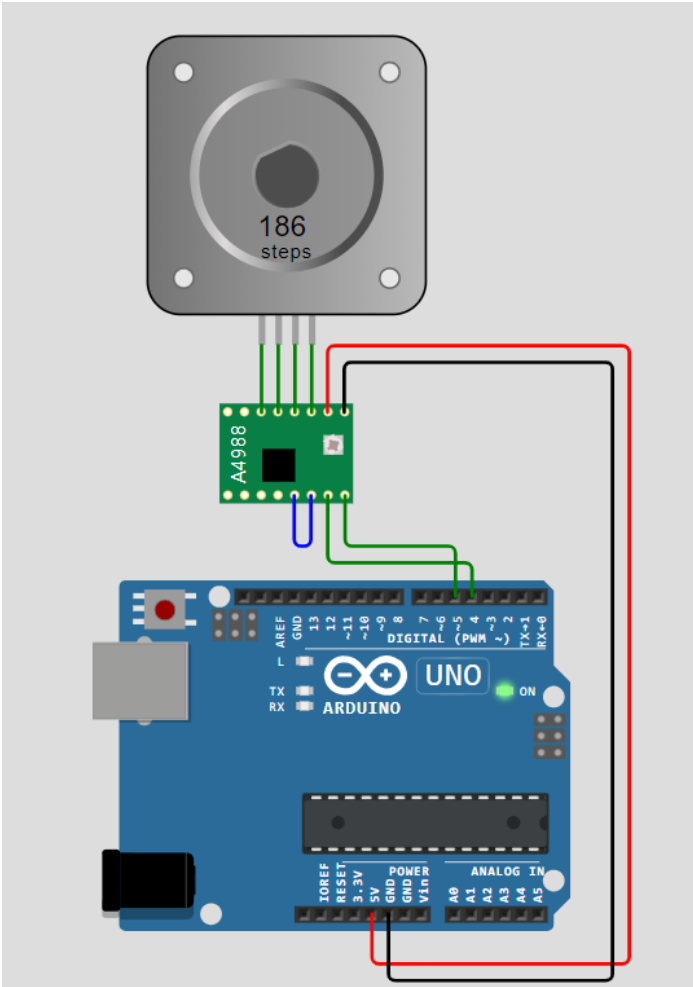


Figura 61. Conexión motor a pasos.

La conexión aplica para motores a pasos de bajo voltaje de funcionamiento, para motores con mayor voltaje se requiere de una fuente adicional, conectada al driver A4988, según los requerimientos del motor (Revisar datasheet).

Explicación del código:

1. Pines de control:

- `stepPin` es el pin que genera los pulsos para que el motor avance un paso cada vez que cambia de estado (de LOW a HIGH).
- `dirPin` controla la dirección del motor (HIGH para una dirección y LOW para la dirección opuesta).

2. En el `setup()`:

- Configuramos los pines **STEP** y **DIR** como salidas.
- También, configuramos la dirección inicial del motor con `digitalWrite(dirPin, HIGH)`.

3. En el `loop()`:

- Se genera una secuencia de pulsos en el pin **STEP** con `digitalWrite(stepPin, HIGH)` seguido de `digitalWrite(stepPin, LOW)` para cada paso del motor.
- Se utilizan 200 pasos para completar una revolución completa (esto depende del motor y la configuración de microstepping).
- Se usa `delayMicroseconds(1000)` para controlar la velocidad. Cuanto menor sea el valor, mayor será la velocidad.
- Se cambia la dirección del motor con `digitalWrite(dirPin, LOW)` para que gire en el sentido opuesto.

Ajustes adicionales:

- **Microstepping:** Puede conectar los pines **MS1**, **MS2**, **MS3** del A4988 a GND o VDD para cambiar la resolución de los pasos (1/2, 1/4, 1/8, 1/16 de paso completo).

- **Velocidad:** Ajuste el valor en `delayMicroseconds(1000)` para cambiar la velocidad de rotación del motor. Un valor más bajo hará que el motor gire más rápido.

Funcionamiento:

- El motor girará 200 pasos en una dirección, hará una pausa de 1 segundo, luego girará 200 pasos en la dirección opuesta.
- Dependiendo del tipo de motor y la configuración de microstepping, el número de pasos puede variar para lograr una revolución completa.

REFERENCIAS BIBLIOGRÁFICAS

- Arduino. (2024). *Arduino Uno Rev3*. <https://www.arduino.cc/en/Main/arduinoBoardUno>
- Autodesk. (2023). *Tinkercad Circuits*. <https://www.tinkercad.com>
- Banzi, M., & Shiloh, M. (2022). *Getting Started with Arduino: The Open Source Electronics Prototyping Platform*. Maker Media.
- Bonsor, K. (2023). *How Pulse Width Modulation Works*. <https://www.howstuffworks.com>
- Corsair Gaming. (2024). *¿Qué es PWM y qué relación tiene con los ventiladores de PC?* Corsair.com. <https://www.corsair.com/es/es/explorer/diy-builder/fans/que-es-pwm/>
- Cruz, A. (2014). *Tutorial Uso Driver L298N para motores DC y paso a paso con Arduino*. Electronilab. <https://electronilab.co/tutoriales/tutorial-de-uso-driver-dual-l298n-para-motores-dc-y-paso-a-paso-con-arduino/>
- Gordon, T. (2022). *Advanced Arduino Programming*. Tech Innovations Press.
- Gordon, R. (2020). *Fundamentals of electronic components*. TechPress.
- González, R. (2022). *Fundamentals of passive infrared sensors in electronic systems*. Electronics Press.
- Gómez, R. (2022). *Tecnologías de sensores en robótica: Aplicaciones y control de distancia*. Editorial Tecnológica.
- Interfacing RGB led with arduino. (2022). Arduino Project Hub. <https://projecthub.arduino.cc/semsemharaz/interfacing-rgb-led-with-arduino-b59902>

- Lemos, A. (2021). *Arduino: El futuro de la programación y la electrónica*. Editorial Universitaria.
- Martínez, L. (2023). *Sistemas de sensores para el monitoreo ambiental en proyectos de IoT*. Tecnología y Automatización.
- Martínez, L. (2021). *Tecnología de iluminación LED y su aplicación en proyectos electrónicos*. Editorial Innovación Electrónica.
- Mckinney, M. (2021). *Digital Electronics with Arduino*. Innovate Press.
- McKinney, D. (2021). *Introduction to temperature sensors in electronics*. TechBooks.
- Mclaughlin, J. (2023). *Arduino Sensors and Actuators: A Practical Guide*. Maker's Academy Press.
- Monk, S. (2022). *Programming Arduino: Getting Started with Sketches*. McGraw Hill.
- Pahlavan, K. (2022). *Fundamentals of Programming in Arduino*. Tech Press.
- Purdum, J. (2020). *Beginning C for Arduino: Learn C Programming for the Arduino (2nd ed.)*. Apress.
- Techmake, E. (2020). Empezando con Arduino - 5C: Motor shield. *Techmake Solutions*. <https://techmake.com/blogs/tutoriales/empezando-con-arduino-5c-motor-shield>
- Tinkercad. (2023). *Tinkercad Circuits: Learn & Program with Arduino*. <https://www.tinkercad.com>
- Vega, E. (2017). *Pulsadores y antirrebote con Arduino*. <https://arduparatodos.blogspot.com/2017/01/pulsadores-y-antirrebote-con-arduino.html>
- Wilson, T. (2022). *The Art of Motor Control with Arduino*. Tech Publishing.
- Wokwi. (2023). *Wokwi Arduino Simulator*. <https://wokwi.com>



Maxwell Arbey Salazar Guilcamaigua

Ingeniero Mecatrónico con Maestría en Ciencias en Robótica y amplia experiencia en docencia universitaria y gestión de proyectos tecnológicos. Actualmente, docente en el Tecnológico Universitario Rumiñahui, donde imparte asignaturas de electrónica, control y modelado 3D. Anteriormente, enseñó informática, redes en la Universidad Metropolitana. Como emprendedor, ha liderado múltiples proyectos en el ámbito de la impresión 3D y la robótica, combinando su pasión por la innovación con la educación. Su experiencia incluye desde el diseño y prototipado de piezas complejas hasta la implementación de soluciones en proyectos de automatización y robótica, lo que le ha permitido desarrollar un enfoque práctico y educativo. Cuenta con una sólida formación académica respaldada por certificaciones en automatización robótica de procesos, bases de datos, ciberseguridad, redes y programación. Su objetivo es crear soluciones que impulsen el avance tecnológico y el aprendizaje interactivo, formando a la próxima generación de ingenieros y tecnólogos.



Paola Alexandra Portero Donoso

Ingeniera en Electrónica, Control y Redes Industriales, con una Maestría en Diseño y Gestión de Proyectos Tecnológicos. Su trayectoria profesional se ha centrado en la docencia y la investigación, con un fuerte enfoque en la robótica educativa y la automatización. Actualmente, es docente en el Tecnológico Universitario Rumiñahui, donde imparte asignaturas acordes a su perfil en la carrera de la Gestión de Mantenimiento Eléctrico y Eficiencia Energética. Su experiencia incluye el desarrollo de proyectos innovadores de electrónica y robótica, con un especial interés en soluciones tecnológicas que promuevan la sostenibilidad ambiental. Le apasiona la enseñanza y el autoaprendizaje, utilizando la tecnología como herramienta clave para mejorar continuamente el proceso educativo. Además, ha colaborado en varias instituciones educativas, brindando conocimientos avanzados en robótica y control automatizado, buscando siempre la innovación en el ámbito tecnológico.

Introducción a la Programación con Arduino: Guía Básica con Ejemplos Prácticos, es una obra destinada a quienes desean iniciarse en el campo de la programación y el desarrollo de proyectos basados en Arduino. Este libro combina de manera estructurada los fundamentos teóricos con una amplia selección de ejemplos prácticos, diseñados para facilitar un aprendizaje progresivo y sólido. A través de sus capítulos, se abordan desde los conceptos esenciales de Arduino, como los tipos de placas, los componentes principales y la instalación del entorno de desarrollo (IDE), hasta la implementación de proyectos funcionales que integran sensores, actuadores y motores. Asimismo, se dedica una atención especial a las entradas y salidas digitales y analógicas, a las estructuras avanzadas de programación y al uso de librerías específicas. Con más de 40 ejemplos detallados, esta guía permite al lector aplicar los conocimientos adquiridos en proyectos concretos, tales como el control de LEDs, la lectura de datos mediante sensores de temperatura, luz y movimiento, y el manejo de motores DC, servomotores y motores a pasos. Además, incluye tutoriales sobre plataformas de simulación como Tinkercad y Wokwi, que ofrecen un entorno ideal para experimentar sin necesidad de contar con hardware físico. Con un enfoque claro y un tono didáctico, esta obra constituye un recurso esencial tanto para quienes se inician en el uso de Arduino como para aquellos interesados en ampliar sus competencias en el desarrollo de proyectos tecnológicos.



ISBN: 978-9942-560-00-1

