Material elaborado en marco del proyecto “Escuelas Resilientes” para las Unidades Educativas de San Antonio de Lomerio

Manual Práctico de Arduino

Componentes y Programación

MSc. CARLA RAQUEL LANDAETA MENDOZA

Área: Ingeniería de Sistemas

Edición: Instituto de Investigación e Interacción Social, Usfa, 2025

ISBN: 978-9917-9753-4-2

Depósito Legal: 4-2-4224-2025

Ciudad, País: La Paz, Bolivia

DOI: https://doi.org/10.5281/zenodo.15728535

Rector: MSc. Jorge Dorado Quiroga

Vicerrectora: Ing. Katerina Rico Cernohorska

Director del instituto de Investigación e Interacción Social: Prof. Dr. Carlos Jorge Landaeta Mendoza

Presentación.

Estimados estudiantes y educadores,

Es muy grato presentar este manual práctico de Arduino, elaborado con dedicación para apoyar el aprendizaje en las Unidades Educativas de San Antonio de Lomerío. Este recurso ha sido desarrollado en el marco del "Proyecto Escuelas Resilientes", con la intención de equipar a los actores educativos del municipio con herramientas prácticas en el ámbito de la electrónica y la programación.

El objetivo de este manual es proporcionar una guía clara y accesible sobre los componentes electrónicos y la programación en Arduino. Cada sección ha sido diseñada para facilitar la comprensión de los temas, desde los componentes básicos hasta la construcción de proyectos prácticos. Espero que encuentren en este material un aliado en su proceso de aprendizaje, que despierte su curiosidad y les permita desarrollar habilidades técnicas valiosas.

A través de este manual, no solo busco fomentar el conocimiento técnico, sino también inspirar un pensamiento crítico y creativo que les permita enfrentar los desafíos del futuro. La educación es una herramienta poderosa, y en este proyecto, espero que cada uno de ustedes pueda descubrir su potencial y contribuir a la construcción de un entorno más resiliente y adaptativo.

Los invito a explorar y experimentar con los conceptos presentados en este manual. Que cada actividad y proyecto que realicen sea una oportunidad para aprender, innovar y crecer.

Con gratitud,

MSc. Carla Raquel Landaeta Mendoza

Contenido

Manual Práctico de Arduino: Componentes y Programación

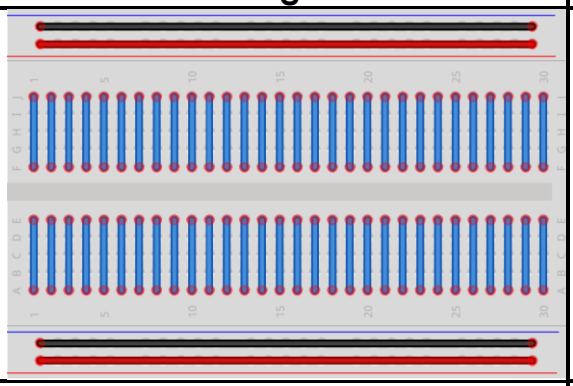
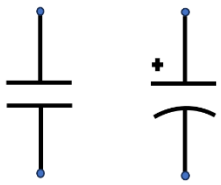
1.- Análisis de los Componentes

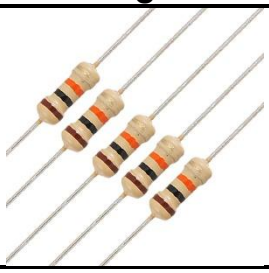
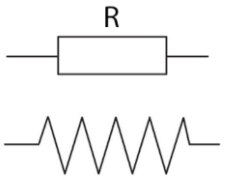
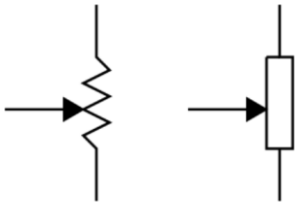
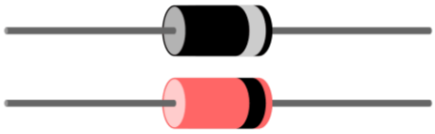
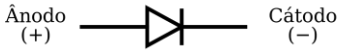
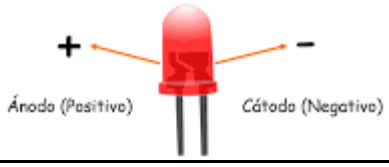
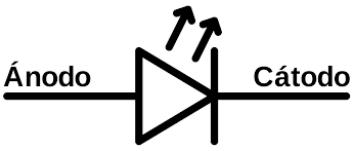
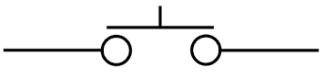
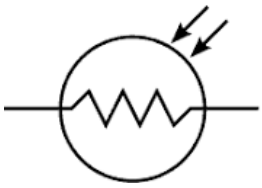
En el ámbito de la electrónica y la programación con Arduino, los componentes electrónicos son los bloques fundamentales para construir circuitos funcionales y desarrollar proyectos interactivos.

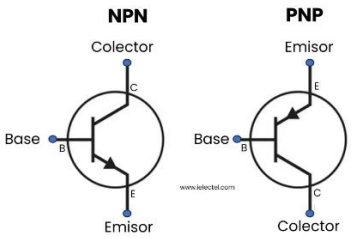
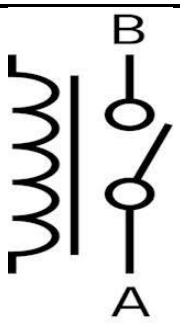
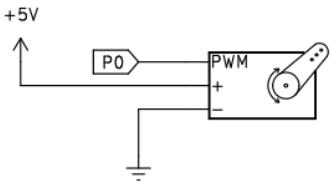
La Tabla 1 presenta una lista de componentes esenciales utilizados en proyectos de Arduino, cada uno con características específicas que los hacen idóneos para diversas aplicaciones, desde prototipos básicos hasta sistemas complejos de automatización e IoT (Internet de las Cosas). Estos componentes, combinados con la plataforma Arduino, permiten a los usuarios experimentar, aprender y crear soluciones tecnológicas innovadoras.

Tabla 1

Tabla de componentes

Nombre	Imagen	Símbolo
Protoboard		
Capacitores - Condensadores		

Nombre	Imagen	Símbolo
Resistencias		
Potenciómetro (Resistencia variable)		
Diodo		
Diodo Led		
Switches - Interruptores		
Pulsadores		
Fotoresistencia - LDR		

Nombre	Imagen	Símbolo
Transistor		
Relé- Relay		
Motor DC		
Puente H L293D		
Servo motor		

Nombre	Imagen	Símbolo
Sensor de humedad ht11		
Sensor de humedad FC28		
Sensor infrarrojo TCRT5000		
Sensor Ultrasónico HC-SR04		
Sensor de Movimiento HC-SR501		
Modulo bluetooh HC-05		

Nombre	Imagen	Símbolo
Buzzer		

1.1. Características de los componentes

En el ámbito de la electrónica y la programación con Arduino, varios componentes juegan un papel importante en la creación de circuitos funcionales. A continuación, se describen las características de estos componentes, que son fundamentales para entender su uso y aplicación.

- **Protoboard**: Una placa de conexiones que permite realizar montajes temporales sin necesidad de soldadura. Es ideal para prototipos y aprendizaje, ya que facilita la modificación de circuitos. Su diseño incluye filas y columnas conductoras que conectan los componentes de manera sencilla.

- **Capacitores (Condensadores)**: Dispositivos que almacenan energía eléctrica en forma de carga. Se utilizan para estabilizar voltajes, filtrar ruido en circuitos y suavizar señales, contribuyendo a la eficiencia y estabilidad de los sistemas electrónicos.

- **Resistencias**: Componentes que limitan el flujo de corriente eléctrica, protegiendo otros elementos del circuito de posibles sobrecargas. Son esenciales para controlar el voltaje y la corriente en LEDs, sensores y otros dispositivos.

- **Potenciómetro (Resistencia variable)**: Una resistencia ajustable manualmente que permite variar la resistencia en un circuito. Se usa comúnmente para controlar la intensidad de luz en LEDs, el volumen en altavoces o la sensibilidad de sensores.

- **Diodo**: Un componente que permite el paso de corriente en una sola dirección, actuando como una válvula eléctrica. Se emplea para proteger circuitos contra polaridad inversa o picos de voltaje.

- **Diodo LED**: Un diodo emisor de luz que ilumina al pasar corriente a través de él. Se utiliza en indicadores visuales, decoración, pantallas y sistemas de señalización, siendo un componente versátil y de bajo consumo.

- **Switches (Interruptores)**: Dispositivos que abren o cierran un circuito eléctrico, permitiendo el control manual de dispositivos como luces, motores o sistemas electrónicos. Son fundamentales para la interacción humana con el circuito.

- **Pulsadores**: Interruptores momentáneos que cierran el circuito solo mientras se presionan. Son comunes en botones de encendido/apagado, controles de entrada en interfaces y sistemas interactivos.

- **Fotoresistencia (LDR)**: Un sensor que varía su resistencia en función de la intensidad de luz recibida. Es ideal para aplicaciones como sistemas de iluminación automática, alarmas fotosensibles y medidores de luz ambiental.

- **Transistor**: Un componente semiconductor que amplifica señales eléctricas o actúa como interruptor electrónico. Es clave en circuitos de control, amplificación y conmutación de potencia.

- **Relé (Relay)**: Un interruptor electromecánico que permite controlar circuitos de alta potencia (como electrodomésticos) utilizando señales de bajo voltaje desde Arduino. Es ideal para automatización del hogar y sistemas de control.

- **Motor DC**: Un motor que convierte energía eléctrica en movimiento mecánico. Se utiliza en ventiladores, ruedas de robots, juguetes y otros dispositivos que requieren movimiento continuo.

- **Puente H (L293D)**: Un circuito integrado que controla la dirección y velocidad de motores DC. Es esencial en robótica, permitiendo que los motores giren en ambas direcciones con precisión.

- **Servo Motor**: Un motor que permite un control preciso del ángulo de rotación, ideal para aplicaciones en robótica, brazos mecánicos, cámaras y sistemas de automatización.

- **Sensor de Humedad HT11**: Un sensor que mide la humedad relativa del aire, útil en estaciones meteorológicas, sistemas de monitoreo ambiental y aplicaciones agrícolas.

- **Sensor de Humedad FC28**: Diseñado para medir la humedad del suelo, es ideal para sistemas de riego automatizado y monitoreo de cultivos, complementando al HT11 en proyectos agrícolas.

- **Sensor Infrarrojo TCRT5000**: Detecta objetos mediante la reflexión de luz infrarroja. Se utiliza en robots seguidores de líneas, contadores de objetos y sistemas de detección de proximidad.

- **Sensor Ultrasónico HC-SR04**: Mide distancias utilizando ondas sonoras, siendo perfecto para sistemas de evitación de obstáculos en robots, medidores de nivel y aplicaciones de mapeo.

- **Sensor de Movimiento HC-SR501**: Detecta movimiento mediante infrarrojos pasivos (PIR). Es común en sistemas de seguridad, iluminación automática y automatización del hogar.

- **Módulo Bluetooth HC-05**: Permite la comunicación inalámbrica entre Arduino y otros dispositivos, como teléfonos móviles o computadoras. Es ideal para proyectos de IoT, control remoto y transmisión de datos.

- **Buzzer**: Un dispositivo que genera sonidos a partir de señales eléctricas. Se utiliza en alarmas, notificaciones, juegos y sistemas de alerta sonora.

1.2. Aplicaciones Prácticas de los Componentes

Los componentes descritos no solo son fundamentales por sus propiedades individuales, sino también por su capacidad de integrarse en proyectos interdisciplinarios. A continuación, se presentan algunas aplicaciones prácticas que combinan estos elementos:

- **Sistemas de Iluminación Inteligente**: Utilizando una fotoresistencia (LDR) y un diodo LED, se puede crear un sistema que encienda luces automáticamente al detectar baja luminosidad, controlado por un potenciómetro para ajustar la sensibilidad.

- **Robótica Básica**: Combinando un motor DC, un puente H (L293D) y un sensor ultrasónico (HC-SR04), es posible construir un robot que evite obstáculos, ideal para introducirse en la robótica educativa.

- **Automatización Agrícola**: Los sensores de humedad HT11 y FC28, junto con un relé, permiten desarrollar un sistema de riego automatizado que active una bomba de agua según las condiciones del suelo y el ambiente.

- **Seguridad Doméstica**: Un sensor de movimiento HC-SR501 y un buzzer pueden integrarse para crear una alarma que se active al detectar intrusos, con notificaciones enviadas vía Bluetooth (HC-05) a un dispositivo móvil.

- **Proyectos de IoT**: El módulo Bluetooth HC-05, combinado con sensores como el HT11 o HC-SR04, permite transmitir datos en tiempo real a una aplicación móvil, ideal para monitoreo remoto de variables ambientales o de seguridad.

1.3. Consideraciones para el Uso de Componentes

Para maximizar el rendimiento y la seguridad en el uso de estos componentes con Arduino, se deben tener en cuenta las siguientes recomendaciones:

- **Compatibilidad de Voltaje**: Verificar que los componentes operen dentro del rango de voltaje proporcionado por la placa Arduino (típicamente 5V o 3.3V) para evitar daños.

- **Resistencias de Protección**: Siempre usar resistencias adecuadas con LEDs y otros componentes sensibles para limitar la corriente y prevenir quemaduras.

- **Conexiones Seguras**: Asegurar que las conexiones en la protoboard sean firmes y correctas, revisando polaridades (especialmente en diodos, LEDs y capacitores).

- **Gestión de Energía**: En proyectos con motores o relés, considerar fuentes de alimentación externas para evitar sobrecargar la placa Arduino.

- **Documentación de Bibliotecas**: Algunos componentes, como el sensor HT11 o el módulo HC-05, requieren bibliotecas específicas en el IDE de Arduino para su correcto funcionamiento.

Este análisis inicial de componentes establece una base sólida para explorar las capacidades de Arduino, desde la construcción de circuitos básicos hasta el desarrollo de sistemas complejos. La combinación de hardware y programación abre un mundo de posibilidades para el aprendizaje y la innovación tecnológica.

2.- Estructura de programación en Arduino

La programación en Arduino se basa en una estructura simple y efectiva que permite a los usuarios crear una amplia variedad de proyectos interactivos. La estructura principal de un programa en Arduino se compone de dos funciones fundamentales: setup() y loop(). A continuación, se detallan las características y funciones de cada una.

2.1. Función setup()

La función setup() se ejecuta una única vez al inicio del programa. Su propósito principal es realizar la configuración inicial del entorno de trabajo. Algunas de las tareas que se llevan a cabo en esta función incluyen:

- Configuración de Pines: Se utilizan comandos específicos para definir los pines del microcontrolador como entradas o salidas. Por ejemplo, un pin puede configurarse como salida para encender un LED o como entrada para leer datos de un sensor.
- Inicialización de Comunicaciones: En esta sección, se pueden establecer comunicaciones entre el Arduino y otros dispositivos. Esto incluye la configuración de interfaces de comunicación como Serial (comunicación por puerto serie), I2C (Inter-Integrated Circuit) y SPI (Serial Peripheral Interface).
- Configuración de Periféricos: Aquí se pueden inicializar otros dispositivos conectados al Arduino, como pantallas, módulos de comunicación, y sensores, asegurando que estén listos para su uso durante la ejecución del programa.

2.2. Función loop()

La función loop() se ejecuta de forma continua después de que setup() ha finalizado. Esta sección es el corazón del programa y se utiliza para implementar la lógica principal del mismo. Algunas de las tareas que se pueden realizar en loop() incluyen:

- Lectura de Sensores: Se pueden leer los valores de los sensores conectados, como temperatura, humedad o luz, y actuar en consecuencia.
- Control de Actuadores: Aquí se puede encender o apagar dispositivos como LEDs, motores o relés basándose en las lecturas de los sensores o en condiciones específicas.
- Respuesta a Eventos: La función loop() permite responder a eventos, como la presión de un botón o la detección de movimiento, realizando acciones inmediatas según la lógica del programa.

3.- Estructuras de control en programación Arduino

Las estructuras de control son fundamentales en la programación, ya que permiten tomar decisiones y repetir bloques de código según ciertas condiciones. En Arduino, al igual que en otros lenguajes de programación, se utilizan diversas estructuras de control que facilitan la lógica del programa.

Tabla 2

Estructuras de control, mas comunes junto con su traducción al español

Ingles	Español
if	si?
if...else	si?…en caso contrario?
for	por
switch case	casos
while	mientras
do... while	hacer mientras
break	finaliza una ejecución
continue	continua
return	retorno
goto	ir a

4.- Sintaxis en Programación Arduino

La sintaxis es el conjunto de reglas que define cómo se deben escribir las instrucciones en un lenguaje de programación. En Arduino, seguir la sintaxis correcta es esencial para que el código funcione adecuadamente. En la tabla 3, se presentan algunos de los elementos más importantes de la sintaxis en Arduino

Tabla 3

Síntesis en Programación Arduino

Ingles	**Español**
;	punto y coma
{ }	llaves
/ /	comentario de una sola línea
/ * * /	comentario de varias líneas
# define	Sirve para definir constantes o macros sin ocupar memoria. Ej: #define LED_PIN 13 (Define el pin del LED)
# include	Se usa para incluir bibliotecas en el código. Ej: #include <Wire.h> (Biblioteca para comunicación I2C)

5.- Operadores en Programación Arduino

Los operadores son símbolos que permiten realizar diversas operaciones en programación, desde cálculos matemáticos hasta comparaciones y manipulación de datos. En Arduino, se utilizan varios tipos de operadores. A continuación, se describen los operadores más comunes, organizados en categorías: operadores matemáticos, de comparación, booleanos y compuestos.

5.1.- Operadores matemáticos

Los operadores matemáticos permiten realizar operaciones aritméticas básicas. En la tabla 4 se presentan los operadores más comunes.

Tabla 4

Operadores matemáticos

Ingles	Español
=	operador de asignación
+	suma
-	resta
*	multiplicación
/	división
%	módulo

5.2.- Operadores de comparación

Los operadores de comparación se utilizan para comparar dos valores y devolver un resultado booleano (verdadero o falso). Los operadores más comunes se presentan en la tabla 5

Tabla 5

Operadores de comparación

Ingles	Español
==	igual que
! =	diferente de
<	menor que
>	mayor que
<=	menor o igual a
>=	mayor o igual a

5.3.- Operadores booleanos

Los operadores booleanos permiten realizar operaciones lógicas. Son fundamentales para construir condiciones más complejas. Los operadores más utilizados se presentan en la tabla 6

Tabla 6

Operadores booleanos

Ingles	Español
&&	y
| |	o
!	no

5.4.- Operadores compuestos

Los operadores compuestos combinan una operación con una asignación, facilitando la escritura de código más conciso. Los operadores compuestos se presentan en la tabla 7

Tabla 7

Operadores compuestos

Ingles	Español
+ +	incremento
--	decremento
+ =	compuesto adición
- =	compuesto substracción
* =	compuesto multiplicación
/ =	compuesto división
-& = (	compuesto bit a bit AND
| =	compuesto bit a bit OR

6.- Tipos de datos en Programación Arduino

En la programación de Arduino, los tipos de datos son importantes para definir la naturaleza de la información que se va a manejar en un programa. Cada tipo de dato tiene un propósito específico y permite a los programadores optimizar el uso de la memoria y la eficiencia del código. Conocer los diferentes tipos de datos y sus aplicaciones es clave para el desarrollo efectivo de proyectos electrónicos.

En la tabla 8 se presentan los tipos de datos más comunes en Arduino, junto con su utilidad, ejemplos y explicaciones detalladas. Estos tipos de datos abarcan desde aquellos que almacenan valores numéricos simples hasta estructuras más complejas para manejar texto y listas de valores. Comprender cómo y cuándo usar cada tipo de dato permitirá crear aplicaciones más eficientes en sus proyectos de Arduino.

Tabla 8

Tipos de datos en Programación Arduino

Tipo de Dato	¿Para qué sirve?	Ejemplo	Explicación
void	Se usa en funciones que no devuelven nada.	void miFuncion() { }	Cuando se necesita ejecutar instrucciones sin devolver nada. Por ejemplo, en la función setup() o loop().
boolean	Guarda true (verdadero) o false (falso).	boolean encendido = true;	Para controlar cosas como si un botón está presionado o si una luz está encendida o apagada
byte	Guarda números pequeños (0 a 255).	byte edad = 25;	
char	Guarda un solo carácter o un número ASCII.	char letra = 'A';	
int	Guarda números enteros (positivos o negativos).	int temperatura = -10;	Para trabajar con **números enteros:** • int para valores normales • long para valores más grandes.
unsigned int	Guarda números enteros solo positivos.	unsigned int velocidad = 50000;	
word	Similar a unsigned int, guarda números positivos.	word distancia = 60000;	
long	Guarda números enteros muy grandes o pequeños.	long tiempo = 1000000;	Para trabajar con **números enteros:** • int para valores normales • long para valores más grandes.
unsigned long	Guarda números grandes sin	unsigned long millisActual = millis();	

Tipo de Dato	¿Para qué sirve?	Ejemplo	Explicación
	signo, ideal para tiempo.		
float	Guarda números con decimales.	float temperatura = 36.75;	
double	Igual que float, pero en algunas placas tiene más precisión.	double pi = 3.1415926535;	Para trabajar con **números con decimales**, usar float. double porque solo tiene más precisión en algunas
string (arreglo de char)	Guarda texto de manera eficiente.	char saludo[] = "Hola";	string (char[]) es más eficiente en memoria, pero más difícil de manipular.
String (objeto)	Guarda y manipula texto fácilmente, pero usa más memoria.	String mensaje = "Hola, mundo!";	String (objeto) es más fácil de usar, pero consume más memoria y puede ralentizar el programa si se usa mal.
array	Guarda varios valores del mismo tipo en una lista.	int numeros[] = {1, 2, 3, 4};	Cuando se necesita almacenar varios valores del mismo tipo, como varias mediciones de un sensor.

7.- Constantes en Programación Arduino

Las constantes son valores que no cambian durante la ejecución de un programa. En Arduino, se utilizan para representar información fija que es fundamental para la lógica y el control de los dispositivos. Usar constantes en lugar de valores literales mejora la legibilidad del código y facilita su mantenimiento, ya que cualquier cambio se puede realizar en un solo lugar.

En la Tabla 9 se presentan las constantes más comunes en programación Arduino, junto con sus usos y ejemplos ilustrativos. Estas constantes abarcan desde indicaciones de estado en pines digitales hasta representaciones numéricas en diferentes bases, así como valores matemáticos predefinidos. Comprender el uso de estas constantes permite escribir código más claro y eficiente, optimizando el control de sus proyectos electrónicos.

Tabla 9

Constantes en Programación Arduino

Constante	¿Para qué sirve?	Ejemplo de Uso
HIGH / LOW	Se usan para indicar **encendido (HIGH) o apagado (LOW)** en pines digitales.	**digitalWrite(13, HIGH);** // Enciende el LED en el pin 13
INPUT / OUTPUT	Se usan en pinMode() para definir si un pin es **entrada (INPUT) o salida (OUTPUT)**.	**pinMode(2, INPUT);** // Configura el pin 2 como entrada pinMode(13, OUTPUT); // Configura el pin 13 como salida
true / false	Verdero/Falso. Son equivalentes a 1 y 0. Se usan en condiciones o en variables tipo boolean.	**boolean estado = true;** if (estado) { digitalWrite(13, HIGH); }
Constantes Enteras (DEC, HEX, OCT, BIN)	Se usan para indicar cómo se quiere representar un número: • Decimal (DEC) • Hexadecimal (HEX) • Octal (OCT) • Binario (BIN).	• **Serial.println(255, DEC);** // Imprime "255" • **Serial.println(255, HEX);** // Imprime "FF" • **Serial.println(255, OCT);** // Imprime "377" • **Serial.println(255, BIN);** // Imprime "11111111"
Constantes Flotantes (PI, EULER)	Son valores matemáticos predefinidos en Arduino: PI (3.1415926535) y EULER (2.7182818284).	**float circunferencia = 2 * PI * radio;**

8.- Conversión de tipos de datos en programación Arduino

Imagina que tienes cajas de diferentes tamaños para guardar cosas. Algunas cajas son pequeñas (como para guardar letras), otras medianas (como para guardar números enteros) y otras grandes (como para guardar números con decimales). La conversión de tipos de datos es como cambiar una cosa de una caja a otra.

A veces, necesitas usar una cosa que está en una caja pequeña en una caja grande, o viceversa. Por ejemplo, si tienes un número entero y quieres sumarle un número con decimales, necesitas convertir el número entero a un número con decimales para que Arduino pueda hacer la suma correctamente.

Arduino tiene algunas herramientas especiales (funciones) que te ayudan a cambiar las cosas de la caja. Algunas de las más comunes se dan a conocer en la tabla 10.

Tabla 10

Conversión de Tipos de Datos en Programación Arduino

Función de Conversión	¿Para qué sirve?	Ejemplo de Uso	Explicación
char()	Convierte un número o carácter en un tipo char.	**char letra = char(65); Serial.println(letra);** // Imprime 'A'	Convierte el número 65 en su **código ASCII**, que es 'A'.
byte()	Convierte un número o carácter en un byte (0-255).	**byte valor = byte(200.75); Serial.println(valor);** // Imprime 200	Toma solo la parte entera del número.
int()	Convierte un número o texto en un entero (int).	**int numero = int(23.9); Serial.println(numero);** // Imprime 23	Redondea **hacia abajo** eliminando los decimales.
word()	Convierte un número en un word (0-65535).	**word w = word(50000); Serial.println(w);**	Ideal para valores grandes **sin signo**.
long()	Convierte un número en un long (valores más grandes que int).	**long largo = long(100000); Serial.println(largo);**	Se usa cuando int no es suficiente.
float()	Convierte un número en un float (con decimales).	**float numero = float(10); Serial.println(numero, 2);** // Imprime 10.00	Añade precisión con decimales.

Usar estas funciones correctamente te ayuda a evitar errores en tus proyectos de Arduino. Por ejemplo, si intentas sumar una letra con un número entero sin convertirlos primero, Arduino no sabrá qué hacer y te dará un error.

9.- Digital I/O (Entradas y Salidas Digitales)

Imagina que tu Arduino es como un interruptor gigante que puede encender y apagar cosas. Las entradas y salidas digitales son como los cables que conectan ese interruptor a diferentes dispositivos.

¿Para qué sirven las entradas y salidas digitales?

- Encender y apagar luces (LEDs): Para hacer que las luces se prendan y apaguen cuando queramos.
- Leer botones: Para saber si un botón está presionado o no.
- Interactuar con sensores: Para saber si algo está cerca, lejos, o si hay movimiento.

Las funciones digitales en Arduino se detallan en la tabla 11

Tabla 11

Digital I/O (Entradas y Salidas Digitales)

Función	¿Para qué sirve?	Ejemplo de Uso	Explicación
pinMode()	Configura un pin como **entrada** o **salida**.	**pinMode(13, OUTPUT);**	Define el modo de un pin específico. INPUT para leer valores (por ejemplo, botones) o OUTPUT para enviar señales (por ejemplo, LEDs).
digitalWrite()	Escribe un **valor digital** (alto o bajo) a un pin de salida.	**digitalWrite(13, HIGH);**	Establece el valor HIGH (encendido) o LOW (apagado) a un pin digital.
digitalRead()	Lee el **estado** de un pin digital de entrada (alto o bajo).	**int valor = digitalRead(2);**	Lee un pin configurado como INPUT y devuelve HIGH o LOW según el valor que recibe.

10.- Analógico I/O (Entradas y Salidas Analógicas)

En el universo de Arduino, las señales que se extienden más allá del simple "encendido" o "apagado" abren un abanico de posibilidades. A diferencia de las señales digitales, que se limitan a dos estados, las señales analógicas varían de forma continua, permitiendo una interacción matizada con el

entorno. Para aprovechar al máximo esta capacidad, Arduino ofrece un conjunto de herramientas esenciales: las funciones de Analógico I/O.

Estas funciones son la clave para interpretar y manipular la información proporcionada por dispositivos como sensores de temperatura, potenciómetros y motores, que requieren una precisión más allá de lo binario. Entre las más destacadas se encuentran detalladas en la tabla 12

Tabla 12

Analógico I/O (Entradas y Salidas Analógicas)

Función	¿Para qué sirve?	Ejemplo de Uso	Explicación
analogReference()	Define la **referencia de voltaje** para las lecturas analógicas.	**analogReference(DEFAULT);**	Configura la referencia de voltaje para las entradas analógicas. Por defecto, es 5V (en Arduino Uno).
analogRead()	Lee un valor **analógico** de un pin (0 a 1023).	**int valor = analogRead(A0);**	Lee el valor de un pin analógico y devuelve un número entre 0 y 1023, dependiendo del voltaje (0-5V).
analogWrite() (PWM)	Escribe un **valor de PWM** (modulación por ancho de pulso) a un pin.	**analogWrite(9, 128);**	Usa PWM para simular voltajes analógicos en pines configurados como OUTPUT. El valor va de 0 a 255.

11.- Tiempo en Programación Arduino

Imagina que tu programa de Arduino es como un director de orquesta, ¡pero necesita un reloj! Por eso, Arduino tiene herramientas para manejar el tiempo, como un cronómetro súper preciso. Esto es importante para que tus proyectos hagan cosas en el momento justo, como encender y apagar luces, mover robots o medir cuánto dura un experimento.

Saber usar estas herramientas es como tener superpoderes para controlar el tiempo en tus proyectos. Puedes hacer que las cosas sucedan exactamente cuando quieras, ¡como un relojito suizo! Desde hacer parpadear LEDs hasta crear juegos interactivos, ¡el tiempo es tu aliado en el mundo de Arduino! Las funciones de tiempo en Programación Arduino se detallan en la tabla 13

Tabla 13

Funciones de tiempo en Programación Arduino

Función	¿Para qué sirve?	Ejemplo de Uso	Explicación
millis()	Este es nuestro cronómetro principal. Cuenta los milisegundos (¡millas de veces por segundo!) que pasan desde que prendemos el Arduino. Es como un reloj que nunca se detiene.	**long tiempo = millis();**	Útil para medir el tiempo sin bloquear el programa. Devuelve el tiempo en milisegundos desde que el Arduino empezó a ejecutarse.
micros()	Si necesitamos ser aún más precisos, ¡usamos micros()! Esta cuenta en microsegundos, que son millonésimas de segundo. ¡Es como tener un cronómetro de alta velocidad!	**long tiempo = micros();**	Al igual que millis(), pero con más precisión (devuelve valores más pequeños).
delay()	A veces, necesitamos que el programa espere un poco. Con delay(), le decimos al Arduino: "¡Espera aquí unos milisegundos antes de seguir!". Es útil para hacer pausas entre acciones.	**delay(1000);**	Pausa el programa por el tiempo indicado en milisegundos.

12.- Funciones Matemáticas en programación Arduino

¿Sabías que las matemáticas también pueden ser divertidas? En Arduino, las usamos todo el tiempo para hacer cosas increíbles. Imagina que tu Arduino es un pequeño robot que necesita pensar y calcular cosas. ¡Ahí es donde entran las matemáticas!

¿Para qué sirven las matemáticas en Arduino?

- Controlar motores: Para que un robot se mueva a la velocidad correcta o gire en el ángulo exacto.
- Leer sensores: Para saber la temperatura, la luz o la distancia, ¡y usar esos datos para hacer algo útil!
- Crear juegos: Para calcular puntajes, movimientos y hacer que todo sea más emocionante.

Funciones matemáticas se detallan a continuación, pero se puede revisar la tabla 14, para mayor detalle:

- min() y max() : Imagina que tienes dos números, ¿cuál es el más grande? ¿Y el más pequeño? Estas funciones te lo dicen al instante.
    - Ejemplo: Si tienes las edades de dos amigos, puedes usar max()para saber quién es el mayor.
- abs() : Convierte cualquier número en positivo. ¡Útil para medir distancias sin importar la dirección!
    - Ejemplo: Si un robot se mueve 5 pasos hacia atrás, abs()te dirá que se movió 5 pasos.
- constrain() : Imagina que tienes un control de volumen que no debe pasar de 10. Esta función asegura que un número se mantenga dentro de un límite.
    - Ejemplo: Puedes usarla para que un motor no gire demasiado rápido.
- map() : Transforma un número de un rango a otro. ¡Como cambiar de escala!
    - Ejemplo: Puedes convertir la lectura de un sensor de luz (de 0 a 1023) a un rango de 0 a 100 para mostrarlo en una pantalla.
- pow() : Calcula la potencia de un número. ¡Como elevarlo al cuadrado o al cubo!
    - Ejemplo: Útil para calcular áreas y volúmenes.
- sqrt() : Encuentra la raíz cuadrada de un número. ¡Lo opuesto a elevar al cuadrado!

- Ejemplo: Se usa en cálculos de distancia y velocidad.

Las matemáticas son como un lenguaje secreto que le da superpoderes a tu Arduino. Con estas funciones, puedes hacer proyectos más inteligentes, precisos y divertidos.

La tabla 14 visibiliza cada una de las funciones explicadas con un ejemplo de uso y la explicación respectiva

Tabla 14

Matemáticas en Arduino

Función	¿Para qué sirve?	Ejemplo de Uso	Explicación
min()	Devuelve el **valor más pequeño** entre dos números.	**int menor = min(10, 5);**	Devuelve el número más bajo entre los dos parámetros.
max()	Devuelve el **valor más grande** entre dos números.	**int mayor = max(10, 5);**	Devuelve el número más alto entre los dos parámetros.
abs()	Devuelve el **valor absoluto** de un número (sin signo).	**int positivo = abs(-5);**	Convierte un número negativo a positivo, o deja los números positivos igual.
constrain()	Restringe un valor dentro de un **rango específico**.	**int valor = constrain(300, 0, 255);**	Asegura que el valor esté dentro de un rango determinado. Si es menor, lo pone en el límite inferior; si es mayor, en el límite superior.
map()	**Re-mapea** un número de un rango a otro.	**int nuevoValor = map(50, 0, 100, 0, 255);**	Re-mapea un valor de un rango de entrada a otro rango de salida.
pow()	Calcula un **número elevado a la potencia** de otro.	**float resultado = pow(2, 3);**	Eleva un número a la potencia que le indiques.
sqrt()	Calcula la **raíz cuadrada** de un número.	**float raiz = sqrt(16);**	Devuelve la raíz cuadrada del número proporcionado.

13.- Funciones Trigonométricas en Programación ardunio (ángulos y movimientos

¿Sabías que las matemáticas de los ángulos pueden ser súper útiles en tus proyectos de Arduino? ¡Así es! Las funciones trigonométricas nos ayudan a calcular cosas cuando tenemos ángulos y queremos crear movimientos circulares o cosas que se repiten. Estas funcionen sirven para:

- Gráficos: Para dibujar círculos, ondas y otras formas en una pantalla.
- Movimiento: Para hacer que un robot se mueva en círculos o siga una trayectoria curva.
- Sonido: Para crear ondas de sonido y música.

Funciones trigonométricas en Arduino:

- sin(): Imagina un columpio que se mueve de un lado a otro. Esta función nos dice qué tan alto está el columpio en cada momento.
- cos(): Similar a sin(), pero nos da la posición del columpio hacia adelante y hacia atrás.
- tan(): Nos ayuda a calcular la pendiente de una línea, ¡útil para saber la dirección de un movimiento!

Estas funciones usan radianes, que son como una forma especial de medir ángulos. Pero no te preocupes, ¡no necesitas ser un experto en trigonometría para usarlas! Arduino hace la mayor parte del trabajo por ti. Revisa la tabla 15, donde tendrás más detalles al respecto

Tabla 15

Funciones Trigonométricas en Programación Arduino

Función	¿Para qué sirve?	Ejemplo de Uso	Explicación
sin()	Calcula el **seno** de un ángulo (en radianes).	**float resultado = sin(PI / 2);**	Devuelve el seno del ángulo en radianes.
cos()	Calcula el **coseno** de un ángulo (en radianes).	**float resultado = cos(PI);**	Devuelve el coseno del ángulo en radianes.
tan()	Calcula la **tangente** de un	**float resultado = tan(PI / 4);**	Devuelve la tangente del

Función	¿Para qué sirve?	Ejemplo de Uso	Explicación
	ángulo (en radianes).		ángulo en radianes.

14.- Comunicación Serial en Programación Arduino

Imagina que tu Arduino necesita hablar con otros dispositivos, como tu computadora o sensores. La comunicación serial es como un lenguaje secreto que les permite intercambiar mensajes. ¡Es como un chat entre tu Arduino y el mundo exterior!

¿Para qué sirve la comunicación serial?

- Enviar y recibir datos: Para que tu Arduino pueda enviar información a tu computadora y viceversa.
- Controlar otros dispositivos: Para dar órdenes a motores, luces u otros aparatos.
- Ver qué está pasando: Para mostrar mensajes en la pantalla de tu computadora y saber si tu programa está funcionando bien.

Las funciones seriales de Arduino se detallan en la tabla 16.

Tabla 16

Función	¿Para qué sirve?	Ejemplo de Uso	Explicación
Serial.begin()	Inicializa la comunicación serial.	**Serial.begin(9600);**	Establece la velocidad de la comunicación serial (por ejemplo, 9600 baudios).
Serial.end()	Termina la comunicación serial.	**Serial.end();**	Detiene la comunicación serial.
Serial.available()	Devuelve el número de **bytes disponibles** para leer.	**bytes = Serial.available();**	Indica cuántos datos están disponibles para ser leídos.

Función	¿Para qué sirve?	Ejemplo de Uso	Explicación
Serial.read()	Lee un **byte** de datos seriales.	**char dato = Serial.read();**	Lee un byte de los datos recibidos a través de la comunicación serial.
Serial.peek()	Muestra el siguiente byte disponible, pero **no lo lee**.	**char siguienteByte = Serial.peek();**	Muestra el siguiente byte sin eliminarlo del buffer de recepción.
Serial.flush()	Espera a que todos los datos pendientes sean enviados.	**Serial.flush();**	Espera a que los datos enviados se transmitan completamente.
Serial.print()	Imprime un **valor** en la comunicación serial (sin salto de línea).	**Serial.print("Hola");**	Imprime texto o valores en el monitor serial.
Serial.println()	Imprime un **valor** y hace un **salto de línea**.	**Serial.println("Hola");**	Imprime texto o valores en el monitor serial con un salto de línea.
Serial.write()	Envía **bytes** de datos sin conversión.	**Serial.write(65);**	Envía un byte de datos sin procesar.