# Universidad Autónoma de Tamaulipas
# Facultad de Ingeniería Tampico

VERDAD, BELLEZA, PROBIDAD

INGENIERÍA TAMPICO

## ASIGNATURA
**Diseño Electrónico Basado en Sistemas Embebidos**

8vo. Semestre – Grupo “G”

2025 -1

## TRABAJO
**Programas en Clase y Ejercicios**

## UNIDAD
**1 - Nombre de la Unidad**

**Docente:** Dr. García Ruiz Alejandro H.

| Integrante del Equipo | Nivel de Participación |
|---|---|
| Hermandez Lara Ana patricia | |
| Antonio Guzmán lris Lucero | |
| Segoviano Arbona Frida Fernanda | |
| Total: | 100% |

# Índice

*Índice* ..... 1

*Repositorio(s)* ..... 2

*C1. Nombre del Programa* ..... 2

*C2. Nombre del Programa* ..... 3

*C3. Programa 3* ..... 4

*C4. Programa 4* ..... 4

*C5.Programa 5* ..... 5

**C.6.Programa 6** ..... **6**

**C.7 Programa 7** ..... **7**

**C.8Programa 8** ..... **8**

**C.9 Programa 9** ..... **9**

**C.10 Programa10** ..... **10**

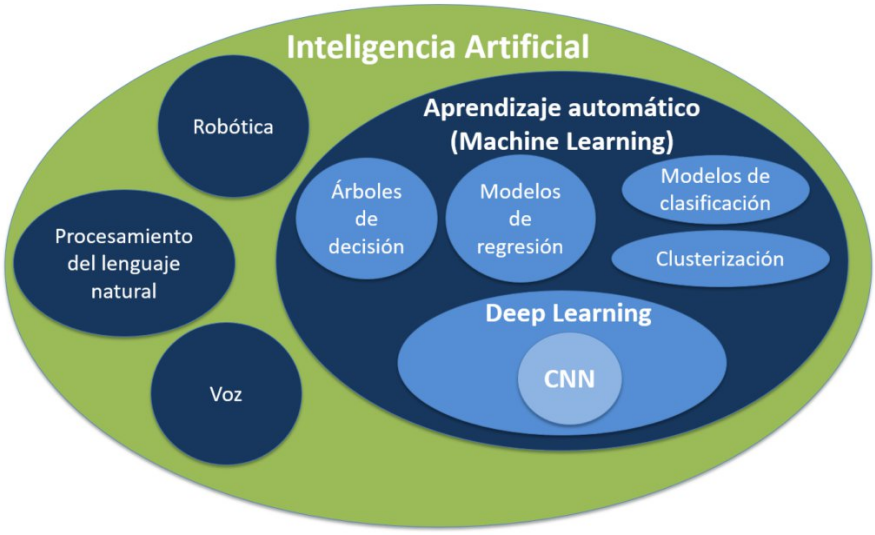
**C.11Programa11** ..... **11**

**C.12Programa12** ..... **12**

## Repositorio(s)

| Actividad | Repositorio |
|---|---|
| Practica 1 a la 3 | https://github.com/ahgarciar |
| Practica 4 | https://github.com/ahgarciar2 |

## C1. Programa 1




**Funcionalidad**

**1-.Este programa toma lecturas del sensor analógico cada segundo y las muestra en el monitor serial. Si el sensor está midiendo, por ejemplo, la luz, la temperatura o alguna otra variable, este código te permitirá ver cómo cambian esos valores con el tiempo.**

**2-.Identificar las instrucciones y estructuras propias del lenguaje, así como, su sintaxis y propósito**

1. **`int sensor = A0;`**: Declara una variable `sensor` que hace referencia al pin analógico A0, donde se conecta el sensor.
2. **`void setup() { Serial.begin(9600); }`**: Inicializa la comunicación serial a 9600 baudios en la función `setup()`, que se ejecuta una sola vez al inicio.
3. **`int v;`**: Declara una variable `v` donde se almacenará el valor leído desde el sensor.
4. **`void loop() { ... }`**: La función `loop()` se ejecuta repetidamente:
   - **`v = analogRead(sensor);`**: Lee el valor del sensor conectado a A0.
   - **`Serial.println(v);`**: Envía el valor leído al monitor serial para ser mostrado.

- **`delay(1000);`**: Pausa la ejecución del programa durante 1 segundo antes de repetir el ciclo.

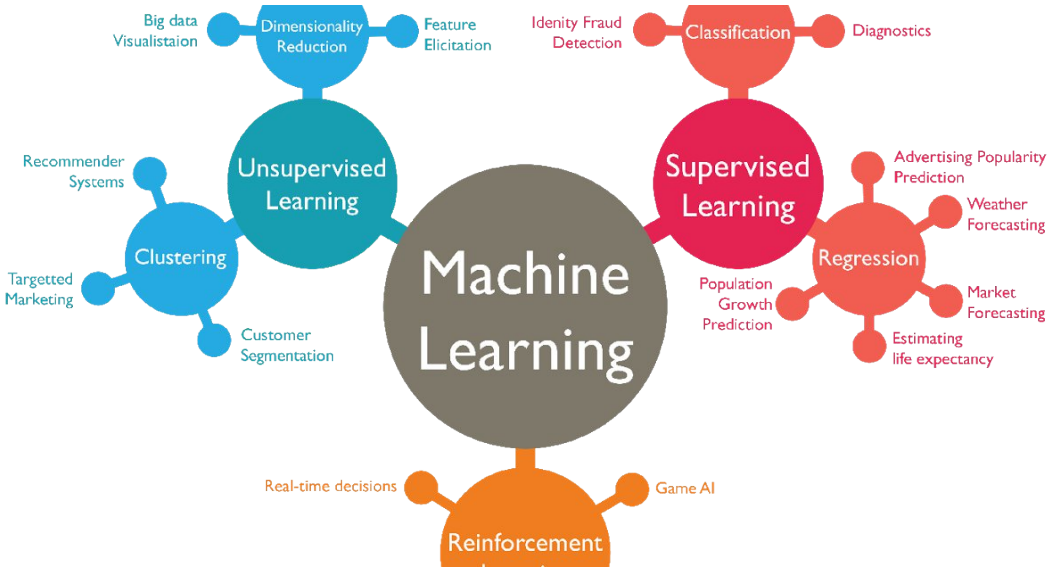
## Sintaxis

- **Declaraciones de variables**: Utiliza `int`, `float`, `char`, entre otros para declarar variables.
- **Funciones predefinidas**: `setup()` y `loop()` son funciones esenciales en los programas de Arduino. `setup()` se ejecuta una vez, mientras que `loop()` se ejecuta repetidamente.
- **Funciones de entrada/salida**: `analogRead()`, `digitalWrite()`, `Serial.begin()`, `Serial.println()`, `delay()`, son algunas de las funciones más comunes utilizadas en el lenguaje de Arduino.

## Propósito general del programa:

Este código lee el valor de un sensor analógico conectado al pin `A0` del Arduino, y muestra el valor leído en el monitor serial cada segundo. Esto permite visualizar cómo varía el valor en función de lo que mida el sensor (por ejemplo, una lectura de temperatura, luz, humedad, etc.).

## C2. Programa 2




**Funcionalidad 2-.Este programa lee el estado de un sensor digital (conectado al pin 10) cada segundo y muestra el valor en el monitor serial. Si el sensor detecta una señal**

**(HIGH), el valor mostrado será 1; si no detecta señal (LOW), el valor será 0. Es un ejemplo simple para leer entradas digitales, como un botón o un interruptor.**

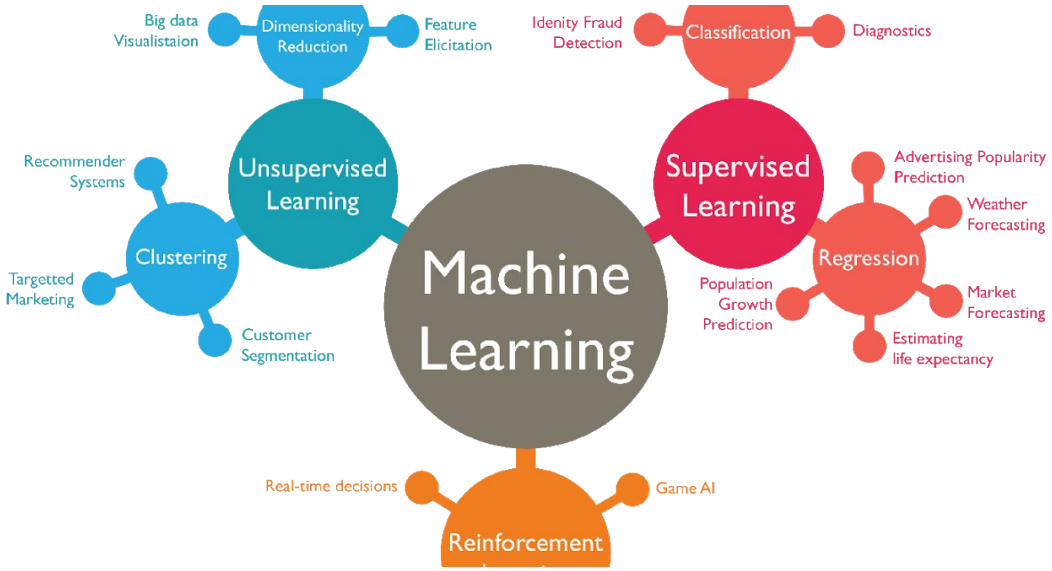
## 2-Identificar las instrucciones y estructuras propias del lenguaje, así como, su sintaxis y propósito

1. **`int sensor = 10;`: Declara la variable `sensor` y le asigna el número del pin 10.**
2. **`void setup() { Serial.begin(9600); pinMode(sensor, INPUT); }`: Inicializa la comunicación serial a 9600 baudios y configura el pin 10 como entrada para leer datos de un sensor.**
3. **`int v;`: Declara la variable `v` que almacenará el valor leído desde el pin 10.**
4. **`void loop() { ... }`: La función que se ejecuta de manera continua:**
    - **`v = digitalRead(sensor);`: Lee el valor digital del pin 10 y lo almacena en `v`.**
    - **`Serial.println(v);`: Imprime el valor de `v` en el monitor serial.**
    - **`delay(1000);`: Pausa la ejecución durante 1 segundo antes de volver a leer el valor.**

## Propósito general del programa

**Este programa lee el estado de un sensor digital conectado al pin 10 del Arduino, y luego imprime el valor (`HIGH` o `LOW`) en el monitor serial cada segundo. Este tipo de programa es comúnmente usado para leer el estado de dispositivos como botones, interruptores o sensores digitales.**

## C3. Programa 3




**Funcionalidad**

**2-.Este programa lee el estado de un sensor digital (conectado al pin 10) cada segundo y muestra el valor en el monitor serial. Si el sensor detecta una señal (HIGH), el valor mostrado será 1; si no detecta señal (LOW), el valor será 0. Es un ejemplo simple para leer entradas digitales, como un botón o un interruptor.**

### 2-.Identificar las instrucciones y estructuras propias del lenguaje, así como, su sintaxis y propósito

1. **`int actuador = 12;`: Declara la variable `actuador` y le asigna el número de pin 12.**
2. **`void setup() { ... }`: Configura la comunicación serial a 9600 baudios y establece el pin 12 como salida.**
3. **`void loop() { ... }`:**
    - **`digitalWrite(actuador, HIGH);`: Enciende el actuador conectando el pin 12 a HIGH (5V).**
    - **`delay(1000);`: Pausa la ejecución durante 1 segundo.**
    - **`digitalWrite(actuador, LOW);`: Apaga el actuador conectando el pin 12 a LOW (0V).**
    - **`delay(1000);`: Pausa la ejecución durante 1 segundo.**

## Propósito general del programa

**Este programa enciende y apaga un actuador (como un LED o un relé) conectado al pin 12 del Arduino en intervalos de 1 segundo. El ciclo de encendido y apagado se repite indefinidamente en el `loop()`.**

## C4. Programa 4

**Funcionalidad**

**4-El programa hace que el actuador (conectado al pin 12) se encienda durante 1 segundo y luego se apague durante 1 segundo, repitiendo este ciclo indefinidamente. El uso de 1 y 0 en digitalWrite es perfectamente válido, pero la convención recomendada es usar HIGH (1) y LOW (0), ya que es más claro y consistente en el código.**

## 2-.Identificar las instrucciones y estructuras propias del lenguaje, así como, su sintaxis y propósito

## Propósito general del programa

**El programa hace que un actuador (por ejemplo, un LED) conectado al pin 12 del Arduino se encienda y se apague en intervalos de 1 segundo. El ciclo se repite de forma continua gracias a la función `loop()`.**

## Sintaxis clave

1. **Declaración de variables**: `int actuador = 12;`
2. **Definición de funciones**:
   - `void setup() { ... }`
   - `void loop() { ... }`
3. **Instrucciones de control de pines**:
   - `digitalWrite(pin, valor);` para encender y apagar el actuador.
   - `delay(tiempo);` para introducir pausas en la ejecución del programa.
4. **Punto y coma `;`**: Cada instrucción termina con un punto y coma.
5. **Llaves `{}`**: Utilizadas para encerrar los bloques de código en funciones.

**C5 .programa 5**

**Funcionalidad**

**El programa hace que la señal PWM en el pin 6 pase por un ciclo de aumento gradual de intensidad (de 0 a 255) y luego disminución gradual (de 255 a 0). Este ciclo se repite continuamente.**

## 2-.Identificar las instrucciones y estructuras propias del lenguaje, así como, su sintaxis y propósito

**Se declara la variable `pinPwm`, que almacena el número del pin utilizado para enviar la señal PWM.**

1. **Funciones:**
   - **`setup()`: Función especial que se ejecuta una sola vez al inicio. En este caso, inicializa la comunicación serial con `Serial.begin(9600);`.**
   - **`loop()`: Función especial que se ejecuta repetidamente. En este caso, controla el comportamiento del PWM en un ciclo de aumento y disminución.**
2. **Instrucciones de control de pines:**
   - **`analogWrite(pin, valor)`: Se utiliza para generar señales PWM.**
   - **`delayMicroseconds(tiempo)`: Introduce un retraso corto en microsegundos entre cada cambio de valor de la señal PWM.**
   - **`delay(tiempo)`: Introduce**

Principio del formulario

Final del formulario

**C6.Programa 6**

**Funcionalidad**

**6-El programa muestra el tiempo transcurrido (en milisegundos) desde que el microcontrolador fue encendido o reiniciado. Este valor aumenta constantemente, y el programa imprime el tiempo en milisegundos en el monitor serie cada segundo.**

**Cada vez que el loop() se repite, se obtiene el valor de millis() (que cuenta el tiempo desde que se inició el programa).**

**Después de 1 segundo (delay(1000)), el valor se vuelve a imprimir, mostrando el tiempo total acumulado en milisegundos.**

## 2-.Identificar las instrucciones y estructuras propias del lenguaje, así como, su sintaxis y propósito

1. **Funciones especiales de Arduino:**
   - `setup()`: **Función que se ejecuta una sola vez al inicio. Aquí se configura la comunicación serial.**
   - `loop()`: **Función que se ejecuta repetidamente después de `setup()`. Aquí se obtiene el tiempo transcurrido y se imprime.**
2. **Funciones integradas:**
   - `millis()`: **Función que devuelve el número de milisegundos desde que se inició el programa.**
   - `Serial.begin()`: **Inicializa la comunicación serial con un puerto de la computadora o dispositivo.**
   - `Serial.println()`: **Imprime datos en el monitor serial y agrega un salto de línea.**
   - `delay()`: **Detiene la ejecución del programa durante un número especificado de milisegundos.**

## Propósito:

**Este programa mide el tiempo transcurrido desde el inicio del programa utilizando la función `millis()`, y lo imprime cada segundo en el monitor serial. Esto puede ser útil para llevar a cabo tareas que dependen del tiempo o simplemente para monitorear el tiempo de ejecución de un programa en Arduino.**

**C7.Programa 7**

**Funcionalidad**

**7-Este programa lee el valor analógico de un sensor conectado al pin A0 y lo mapea a un rango de 0 a 255. Luego, usa ese valor mapeado para controlar un actuador (como un LED o motor) conectado al pin 6, ajustando la señal PWM del actuador. Cuanto mayor sea el valor del sensor, más fuerte será la señal de salida al actuador.**

## 2-.Identificar las instrucciones y estructuras propias del lenguaje, así como, su sintaxis y propósito

1. **Declaración de variables:**
   - `int sensor = A0;`: **Declara la variable `sensor` y la asigna al pin analógico `A0`.**
   - `int actuador = 6;`: **Declara la variable `actuador` y la asigna al pin `6`, que es compatible con señales PWM.**

2. **Función especial de Arduino:**
    - `setup()`: **Configura el entorno de ejecución, en este caso, inicializa la comunicación serial.**
    - `loop()`: **Es la función que se ejecuta repetidamente, donde se leen los datos del sensor, se mapean, y se envían al actuador.**
3. **Funciones integradas de Arduino:**
    - `analogRead()`: **Lee un valor analógico del pin especificado.**
    - `map()`: **Mapea un valor de un rango a otro.**
    - `analogWrite()`: **Envia una señal PWM a un pin específico.**
    - `delay()`: **Pausa la ejecución del programa durante un número de milisegundos.**

---

## Propósito general del programa:

**El propósito de este programa es leer un valor de un sensor analógico (como un potenciómetro) conectado al pin `A0`, mapear ese valor en un rango de 0 a 255, y luego usar ese valor para controlar un actuador (por ejemplo, un LED) conectado al pin `6` a través de una señal PWM. El valor del actuador se ajusta cada 100 milisegundos según la lectura del sensor.**

**C8.Programa 8**

**Funcionalidad**

**8-El código lee el valor del sensor conectado a A0 usando digitalRead(). Si el sensor genera una señal digital (HIGH o LOW), esto se guarda en la variable v.**

## 2-.Identificar las instrucciones y estructuras propias del lenguaje, así como, su sintaxis y propósito

1. **Declaración de variables:**
    - `int sensor = A0;`: **Asocia el pin `A0` al sensor.**
    - `int actuador = 6;`: **Asocia el pin `6` al actuador.**
2. **Funciones especiales de Arduino:**
    - `setup()`: **Inicializa la configuración, incluyendo la comunicación serial (aunque no se usa en este caso para imprimir).**
    - `loop()`: **Función que se ejecuta continuamente para controlar el actuador según el estado del sensor.**
3. **Funciones integradas de Arduino:**
    - `digitalRead()`: **Lee el estado digital de un pin (alto o bajo).**
    - `analogWrite()`: **Envía una señal PWM a un pin.**

- `delay()`: Pausa el programa durante el número de milisegundos especificado.

4. Operaciones aritméticas:
   - La instrucción `v = v / 4;` ajusta el valor leído del sensor, aunque en este caso, no tiene un efecto significativo debido a que `digitalRead()` solo devuelve `0` o `1`.

---

## Propósito:

Este programa lee el estado digital de un sensor (conectado al pin `A0`), lo ajusta a un valor entre 0 y 1 (aunque esto se logra de manera ineficaz con la operación de división), y usa esa información para controlar un actuador (por ejemplo, un LED) conectado al pin `6` a través de una señal PWM. La frecuencia de lectura y ajuste es controlada por un retraso de 100 milisegundos.

**C9.Programa 9**

**Funcionalidad**

el programa configura los pines 8, 9, 10, y 11 como salidas. Ciclo principal (loop()): En el loop(), el programa enciende y apaga los actuadores (uno por uno) en secuencia. Cada actuador permanece encendido durante 100 milisegundos y luego apagado durante 100 milisegundos.Hay un pequeño retraso adicional de 100 milisegundos después de cada ciclo completo, antes de comenzar de nuevo.

## 2-.Identificar las instrucciones y estructuras propias del lenguaje, así como, su sintaxis y propósito

1. Declaración de variables:
   - `int actuadores[4] = {8, 9, 10, 11};`: Se declara un arreglo que almacena los pines de los actuadores.
2. Funciones especiales de Arduino:
   - `setup()`: Se ejecuta una vez al inicio, donde se inicializa la comunicación serial y los pines como salidas.
   - `loop()`: Se ejecuta repetidamente, controlando los actuadores.
3. Funciones de Arduino:
   - `Serial.begin()`: Inicializa la comunicación serial.
   - `pinMode()`: Configura un pin como entrada o salida.
   - `digitalWrite()`: Envia una señal digital a un pin (HIGH o LOW).
   - `delay()`: Introduce un retraso en milisegundos.
4. Estructura de repetición `for`:

- **Se utiliza para recorrer el arreglo de actuadores y encender/apagar cada uno secuencialmente.**

---

## Propósito :

**El programa controla 4 actuadores conectados a los pines `8`, `9`, `10` y `11` de la placa Arduino. Cada actuador se enciende y apaga secuencialmente, con un retardo de 100 milisegundos entre cada cambio de estado. Este proceso se repite continuamente en la función `loop()`, lo que crea un patrón de encendido y apagado para los actuadores (probablemente LEDs o motores).**

**C9.Programa 10**

**Funcionalidad**

**El programa hace que los actuadores conectados a los pines 8, 9, 10 y 11 se enciendan y apaguen uno por uno, con un pequeño retardo entre cada cambio de estado. Cada actuador parpadeará durante 100 milisegundos, y todo el ciclo se repite infinitamente.**

**2-.Identificar las instrucciones y estructuras propias del lenguaje, así como, su sintaxis y propósito**

1. **Declaración de variables:**
   - **`int actuadores[4] = {8, 9, 10, 11};`: Define un arreglo de 4 pines de salida que controlan los actuadores.**
2. **Funciones especiales de Arduino:**
   - **`setup()`: Inicializa la configuración, como la comunicación serial y la configuración de los pines como salidas.**
   - **`loop()`: Se ejecuta repetidamente, controlando los actuadores.**
3. **Funciones de control de pines:**
   - **`pinMode()`: Configura un pin como entrada o salida.**
   - **`digitalWrite()`: Envía un valor alto (HIGH) o bajo (LOW) a un pin, controlando el estado de un actuador.**
   - **`delay()`: Introduce un retraso en milisegundos.**
4. **Estructura de repetición `for`:**
   - **Utilizada para recorrer el arreglo de actuadores y controlar cada uno de ellos de manera secuencial.**

**Propósito :**

**Este programa enciende y apaga 4 actuadores (por ejemplo, LEDs) conectados a los pines 8, 9, 10 y 11 de la placa Arduino. Cada actuador se enciende y apaga en secuencia, con un retraso de 100 milisegundos entre cada cambio de estado. Este ciclo se repite continuamente en la función `loop()`.**

**C10.Programa 11**

**Funcionalidad**

**Este programa tiene como objetivo controlar un actuador (por ejemplo, un LED o un relé) conectado al pin 10 de tu Arduino a través de la comunicación serial. La idea es que el Arduino reciba un valor por el puerto serie y luego use ese valor para encender o apagar el actuador**

**2-.Identificar las instrucciones y estructuras propias del lenguaje, así como, su sintaxis y propósito**

1. **Declaración de variables:**
   - `int actuador = 10;`: **Define el pin de salida para el actuador (pin 10).**
   - `int v;`: **Declara una variable para almacenar el valor recibido desde el puerto serial.**
2. **Funciones especiales de Arduino:**
   - `setup()`: **Configura la comunicación serial y los pines de salida.**
   - `loop()`: **Ejecuta continuamente el ciclo de lectura de datos seriales y control de actuadores.**
3. **Funciones de Arduino:**
   - `Serial.begin()`: **Inicializa la comunicación serial.**
   - `Serial.setTimeout()`: **Establece el tiempo máximo de espera para leer datos desde el puerto serial.**
   - `pinMode()`: **Configura un pin como entrada o salida.**
   - `Serial.available()`: **Verifica si hay datos disponibles para leer.**
   - `Serial.readString()`: **Lee una cadena de caracteres del puerto serial.**
   - `toInt()`: **Convierte una cadena de caracteres a un valor entero.**
   - `digitalWrite()`: **Controla un pin de salida.**
   - `delay()`: **Introduce un retraso en el programa.**

4. **Estructura de control:**
    - `if`: **Verifica si hay datos disponibles para ser leídos.**

---

**Propósito:**

**Este programa lee valores enviados desde la comunicación serial (por ejemplo, 0 o 1) y usa esos valores para controlar un actuador (como un LED o motor) conectado al pin 10 de la placa Arduino. El valor recibido controla si el actuador se enciende (1) o apaga (0). El programa espera 1 segundo entre cada ciclo de lectura y control del actuador.**

## C12.Programa 12

## Funcionalidad

**El programa lee el valor del sensor conectado al pin `A0` de forma continua. Este valor es una representación numérica de la tensión analógica en el pin, en el rango de 0 a 1023.**

## 2-.Identificar las instrucciones y estructuras propias del lenguaje, así como, su sintaxis y propósito

1. **Declaración de variables:**
    - `int sensor = A0;`: **Define el pin de entrada analógica donde está conectado el sensor.**
    - `int v;`: **Define una variable para almacenar el valor leído desde el sensor.**
2. **Funciones especiales de Arduino:**
    - `setup()`: **Se ejecuta una vez al principio para configurar la comunicación serial.**
    - `loop()`: **Se ejecuta repetidamente para leer y mostrar el valor del sensor.**
3. **Funciones de Arduino:**
    - `Serial.begin()`: **Inicializa la comunicación serial.**
    - `analogRead()`: **Lee el valor analógico de un pin.**
    - `Serial.println()`: **Imprime el valor en la consola serial.**
    - `delay()`: **Introduce un retraso entre las lecturas.**
4. **Estructura de control:**
    - **No se utilizan estructuras de control como `if` o `for`, ya que el programa es simple y realiza las mismas acciones repetidamente.**

**Propósito:**

**Este programa lee el valor de un sensor conectado al pin `A0` (un sensor analógico) y luego imprime ese valor en la consola serial cada segundo. La lectura se realiza continuamente en la función `loop()`, lo que permite monitorear de manera continua el valor del sensor en el monitor serial de la IDE de Arduino.**

## 3.- Observar y analizar la capacidad de contar con variantes para llevar a cabo una misma actividad

Hay varios programas que sirven para leer sensores y mostrar lo que están detectando, como los programas 1, 2, 7 y 12. Cada uno hace algo parecido, pero con un enfoque diferente:

- Programa 1: Usa analogRead() para leer valores analógicos. Por ejemplo, si tienes un sensor de luz, puedes medir qué tan iluminado está el lugar. Este programa imprime los valores en el monitor serial cada segundo.

- Programa 2: En vez de leer valores continuos, este usa digitalRead() para saber si algo está prendido o apagado (HIGH o LOW). Por ejemplo, podrías usarlo para saber si un botón está presionado.

- Programa 7: Toma un valor analógico como en el programa 1, pero lo convierte a un rango de 0 a 255 usando map(). Esto es útil si necesitas usar ese valor para controlar algo, como la intensidad de un LED.

- Programa 12: Es como el programa 1, pero además agrega un texto explicativo al valor que se imprime, para que sea más fácil de entender.

La diferencia está en qué tipo de sensor estás usando (analógico o digital) y en cómo quieres mostrar o usar los datos que obtienes.

Los programas 3, 4, 5, 9 y 10 controlan cosas como LEDs o motores. Aunque hacen cosas parecidas, cada uno tiene un enfoque diferente:

- Programas 3 y 4: Ambos encienden y apagan un actuador (por ejemplo, un LED) con digitalWrite(). La diferencia es que el programa 4 usa valores más claros (HIGH y LOW) en lugar de números (1 y 0).

- Programa 5: Aquí no solo prendes y apagas, sino que controlas la intensidad del actuador con señales PWM. Por ejemplo, puedes hacer que un LED se vea más tenue o más brillante.

- Programa 9: Este controla varios actuadores (como cuatro LEDs) en secuencia, uno por uno. Es útil si quieres hacer patrones de luces.

- Programa 10: Similar al programa 9, pero con la ventaja de que puedes cambiar lo que hace usando comandos enviados desde tu computadora.

Depende de si quieres controlar una sola cosa, varias, o si necesitas que el sistema responda a comandos en tiempo real.

Los programas 10 y 11 permiten interactuar con el Arduino desde tu computadora, lo que significa que puedes enviarle órdenes:

- Programa 11: Recibe un valor (0 o 1) para encender o apagar un actuador. Es simple y directo.

- Programa 10: Más avanzado, ya que puedes mandar cadenas de texto para controlar varios actuadores al mismo tiempo.

La complejidad de los comandos que puedes enviar y la cantidad de cosas que puedes controlar. El programa 10 es más flexible porque permite manejar más dispositivos.

## 4.- Corregir errores y/o optimizar los códigos, o en su defecto, señalar porque no es necesario

**Programa 5:** La variable pinPwm no está configurada como salida con pinMode.

p5.ino p6.ino p7.ino

```ino
int pinPwm = 6;

void setup(){
    Serial.begin(9600);
    pinMode(pinPwm, OUTPUT);
}

void loop()
{
    for (int i = 0; i < 255; i++)
    {
        analogWrite(pinPwm, i);
        delayMicroseconds(10);
    }
    delay(10);

    for(int i = 255; i > 0; i--)
    {
        analogWrite(pinPwm, i);
        delayMicroseconds(10);
    }
    delay(10);
}
```

**Programa 8** Usar digitalRead() en un pin analógico (A0) no tiene sentido, debería ser analogRead().

```ino
int sensor = A0;
int actuador = 6;

setup(){
    Serial.begin(9600);
        pinMode(sensor, INPUT);
}

int v;

loop(){
    int v = analogRead(sensor);
    v = v / 4;
    analogWrite(actuador, v);
    delay(1000);
}
```

**Programa 9:** El código funciona correctamente, pero puede optimizarse:

```ino
int actuadores[] = {8, 9, 10, 11};

void setup() {
  for (int i = 0; i < 4; i++) {
    pinMode(actuadores[i], OUTPUT);
  }
}

void loop() {
  for (int i = 0; i < 4; i++) {
    digitalWrite(actuadores[i], HIGH);
    delay(100);
    digitalWrite(actuadores[i], LOW);
    delay(100);
  }
  delay(100);
}
```

**Programa 10:** El arreglo actuadores[i] no se usa correctamente al apagar el actuador; se utiliza una variable incorrecta.

```
int actuadores[] = {8, 9, 10, 11};

void setup() {
  for (int i = 0; i < 4; i++) {
    pinMode(actuadores[i], OUTPUT);
  }
  Serial.begin(9600);
}

void loop() {
  if (Serial.available() > 0) {
    int v = Serial.parseInt();
    for (int i = 0; i < 4; i++) {
      digitalWrite(actuadores[i], v & (1 << i) ? HIGH : LOW);
    }
  }
}

```

**Programa 12:** Ningún error encontrado, pero se puede optimizar.

```
int sensor = A0;

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

void loop() {
  int v = analogRead(sensor);
  Serial.print("Valor del sensor: ");
  Serial.println(v);
  delay(1000);
}

```

## 5.- Concluir el análisis basado en las observaciones realizadas y señalizar al menos una aplicación para cada programa

El programa 1 sirve para medir valores de un sensor analógico, como la luz o la temperatura. Es muy útil, por ejemplo, en la agricultura, donde se podría usar para saber cuánta luz o calor reciben las plantas y así cuidarlas mejor.
El programa 2 lee si un sensor digital está encendido o apagado, como un botón o un interruptor. Este programa sería muy bueno para un sistema de seguridad que detecte si una puerta está abierta o cerrada.
El programa 3 controla un LED o un motor encendiéndolo y apagándolo cada segundo. Podría usarse en un adorno para que las luces se prendan y se apaguen, como en decoraciones de fiestas.
El programa 4 hace lo mismo que el tercero, pero de una manera más ordenada usando las palabras HIGH y LOW. Esto es más fácil de entender y es ideal para quienes están aprendiendo a programar con Arduino.
El programa 5 es especial porque permite controlar la intensidad de un LED o la velocidad de un motor de manera suave. Se podría usar para hacer una lámpara que simule un amanecer, con la luz aumentando poco a poco.
El programa 6 muestra cuánto tiempo ha pasado desde que se encendió el Arduino. Esto puede ser útil para medir cuánto tiempo ha durado un proceso o para saber cuánto tiempo ha estado encendido un dispositivo.
El programa 7 toma el valor de un sensor analógico, lo ajusta y lo usa para controlar algo como un LED. Podría servir para que la luz de una habitación se ajuste automáticamente dependiendo de cuánta luz del sol entre por la ventana.
El programa 8 trata de hacer algo parecido al séptimo, pero usa un sensor digital, lo cual no es muy eficiente. Aun así, podría usarse para proyectos básicos, como controlar un LED con un botón o un sensor simple.
El programa 9 controla varios actuadores, como LEDs o motores, encendiéndolos uno por uno en una secuencia. Sería perfecto para decorar algo con luces que se prendan y apaguen en un orden bonito, como en un árbol de Navidad.
El programa 10 permite manejar los actuadores desde una computadora enviando comandos. Esto sería muy útil si quisieras controlar las luces de tu casa desde tu computadora o incluso desde tu celular.
En el programa 11, se puede encender o apagar un actuador según lo que le digas desde la computadora. Esto es útil para probar dispositivos, como relés o LEDs, de manera sencilla y directa.
En el programa 12, se leen los valores de un sensor y se muestran con un mensaje en la computadora. Esto podría ser útil para monitorear cosas como el nivel de agua en un tanque o la cantidad de luz en un lugar y ver esos datos en tiempo real.