"Introducción a la Robótica para la Enseñanza Técnico Profesional"

Módulo 3 – Programación aplicada a la robótica

Lección 7 – ¿Cómo funciona un programa?

📘 ¿Qué es un programa en robótica?
Un programa es un conjunto ordenado de instrucciones que le indican al controlador —como el Arduino Uno— cómo debe comportarse frente a diferentes situaciones. En otras palabras, es la lógica que organiza el funcionamiento del robot, paso a paso, de forma secuencial.
En robótica educativa, programar no significa que el sistema "piense" por sí mismo, sino que responde a condiciones previamente definidas por quien lo diseña. El programa es el puente entre lo físico (cables, sensores, motores) y lo lógico (decisiones, repeticiones, condiciones).

¿Qué tareas cumple un programa robótico?
Un programa en un sistema robótico tiene tres funciones principales:

Leer información del entorno (entradas): A través de sensores, el robot detecta luz, temperatura, movimiento, humedad, etc.
Procesar la información (decisión): Compara los datos con condiciones definidas (por ejemplo: “si hay poca luz...”) y decide qué hacer.
Ejecutar una acción (salidas): Enciende un LED, activa un motor, emite un sonido, muestra datos en pantalla, etc.

📌 El sistema robótico no actúa solo: Hace exactamente lo que el programa le indica que debe hacer.

🗣️ Analogía simple: Así como una receta le dice a una persona qué ingredientes usar, cuánto tiempo cocinar y qué pasos seguir, un programa le dice a un sistema robótico qué entradas leer, qué condiciones analizar y qué acciones ejecutar.

🔁 La estructura básica de todo programa en Arduino
Todo sketch en Arduino —así se llama cada archivo de programa— tiene dos bloques fundamentales:

Bloque	Función
void setup()	Se ejecuta una vez al inicio. Se usa para preparar el sistema.
void loop()	Se ejecuta de forma repetida. Contiene la lógica principal.

Esta estructura permite que el sistema comience configurando sus pines o componentes, y luego entre en un ciclo continuo de toma de decisiones y acciones, como lo haría un sistema robótico que debe funcionar sin intervención constante.

🛠️ void setup() – El punto de partida del programa
El bloque void setup() es una sección obligatoria de todo programa en Arduino. Su función principal es realizar las configuraciones iniciales, es decir, preparar el sistema para que funcione correctamente.

📌 ¿Cuándo se ejecuta?
Solo una vez, al comenzar el programa:

Cuando se enciende el Arduino.
Cuando se pulsa el botón de reset.

Cuando se vuelve a cargar el programa desde la computadora.

❓ ¿Qué se hace en setup()?
Se definen los modos de los pines: Entrada (INPUT), salida (OUTPUT) o entrada con resistencia interna (INPUT_PULLUP).

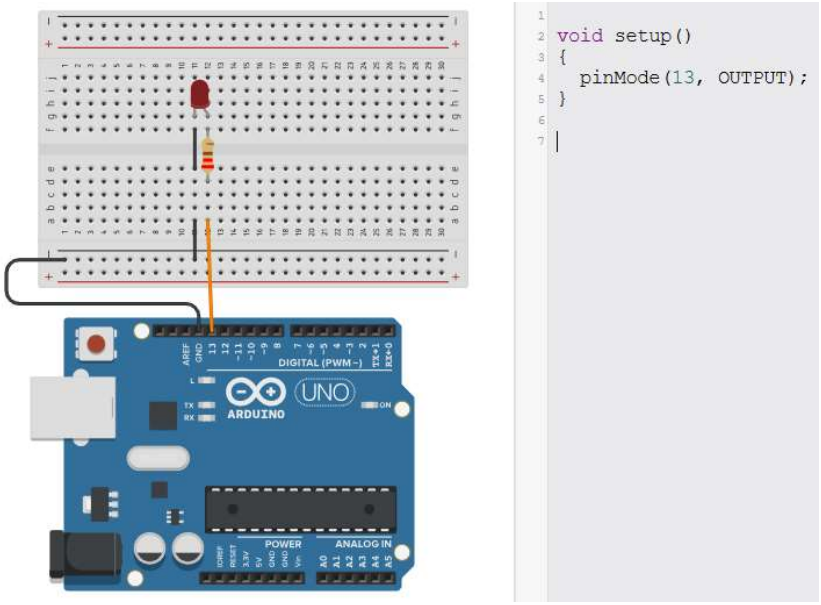
Se puede iniciar la comunicación serial con el monitor de la computadora (Serial.begin()).
Se cargan valores iniciales o se inicializan sensores, pantallas, o módulos de comunicación.

💬 Analogía educativa
setup() es como el momento de preparación antes de comenzar un experimento en el laboratorio: se conectan los instrumentos, se encienden los equipos, se pone todo en condiciones para empezar a trabajar.

Ejemplo básico:

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

Este código le dice a Arduino: "Cuando arranques, prepará el pin 13 para que se use como salida digital."
Eso permite, por ejemplo, conectar un LED a ese pin y encenderlo o apagarlo más adelante desde el bloque loop().



Nota importante

Si no se define correctamente un pin como OUTPUT, Arduino no enviará señal de salida en ese pin, aunque el código en loop() lo intente. Por eso, setup() es el paso previo esencial para que el sistema responda como esperamos.

🔁 void loop() – El comportamiento dinámico del sistema
Después de que el controlador se inicializa con setup(), comienza a ejecutar el segundo bloque esencial: void loop().

Este bloque define el comportamiento repetitivo y permanente del sistema robótico, y es donde realmente ocurre la interacción continua entre el robot y su entorno.

📌 ¿Qué hace loop()?
Se ejecuta en bucle infinito mientras el Arduino esté encendido.
Repite una y otra vez todas las instrucciones que contiene.
Permite al sistema observar su entorno, tomar decisiones y actuar en tiempo real.
Es el corazón del ciclo sensar–procesar–actuar, que define a todo sistema robótico.

¿Qué se escribe en loop()?
Instrucciones para leer sensores (como botones, sensores de luz o movimiento).
Condiciones para tomar decisiones (if, else).
Instrucciones para activar salidas (LEDs, motores, pantallas).
Repeticiones, temporizadores, conteos o eventos.

💬 Analogía educativa: loop() es como el ciclo de atención de una persona que está trabajando: Observa constantemente, decide qué hacer, actúa, vuelve a observar, y así sucesivamente. Nunca se detiene, salvo que se apague.

¿Qué es delay()?
Dentro del bloque loop(), a veces necesitamos esperar un tiempo antes de ejecutar la siguiente instrucción. Para eso se usa la función:

```
delay(tiempo_en_milisegundos);
```

delay(1000) pausa el programa durante 1000 milisegundos (1 segundo).

Mientras dura la pausa, Arduino no ejecuta ninguna otra acción.

🔍 Es útil para:

Crear efectos visibles (como parpadeos o pausas).
Simular tiempos de espera en automatismos.
Evitar que acciones se repitan demasiado rápido.

⚠ En proyectos más complejos, delay() puede no ser recomendable si se necesita una respuesta continua o multitarea, pero es ideal para los primeros pasos.

Ejemplo básico: parpadeo de un LED

```
void loop()
{
  digitalWrite(13, HIGH);// Enciende el LED
  delay(500); // Espera medio segundo
  digitalWrite(13, LOW);// Apaga el LED
  delay(500); // Espera otro medio segundo
}
```

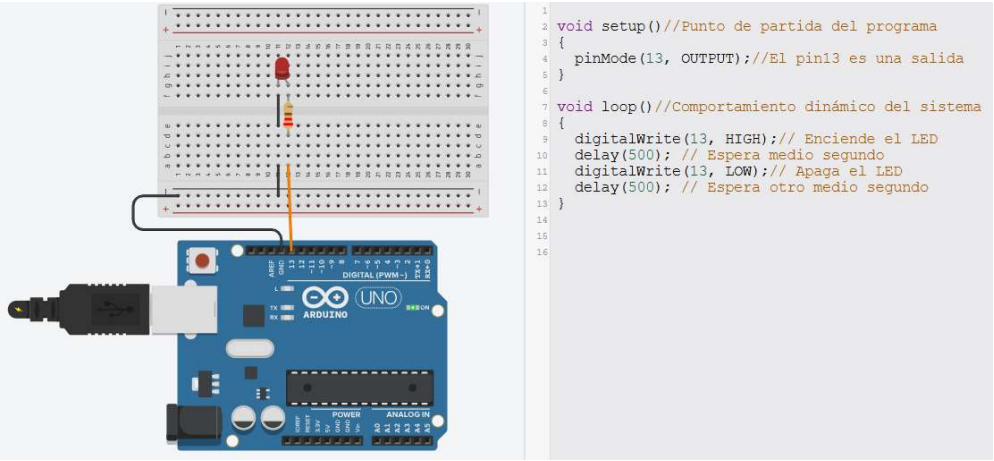
Este programa hace que el LED conectado al pin 13 se encienda y apague cada 500 milisegundos. Mientras el Arduino esté encendido, este ciclo se repite indefinidamente.

🔍 Análisis del flujo:

Enciende el LED (HIGH)
Espera 500 ms (delay(500))
Apaga el LED (LOW)
Espera 500 ms
Vuelve a empezar…

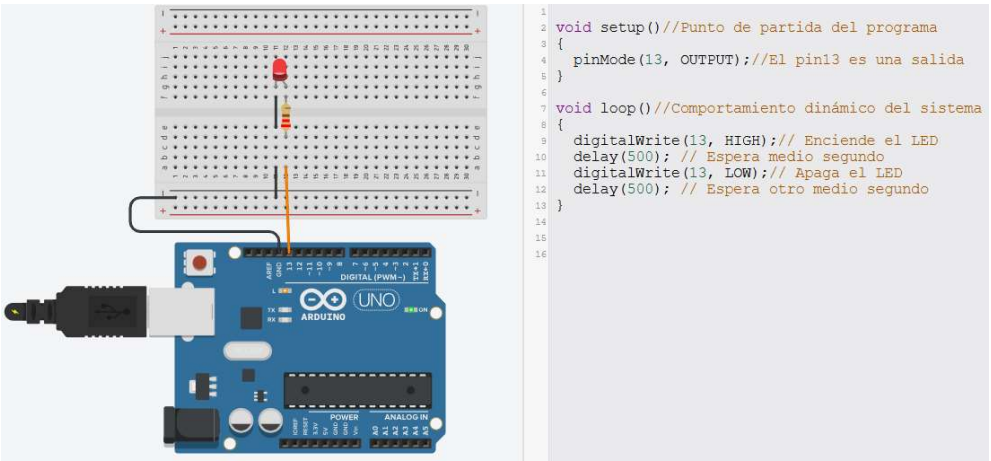
🎓 Relevancia didáctica
Trabajar con loop() y delay() permite a docentes y estudiantes:
Modelar procesos cíclicos (semaforización, lectura continua de sensores, control de motores).
Comprender el principio de tiempo real y de sistemas reactivos.
Ver cómo las instrucciones escritas generan comportamiento físico continuo y temporizado.

Veamos el ejemplo completo:



```
void setup()//Punto de partida del programa
{
  pinMode(13, OUTPUT);//El pin13 es una salida
}

void loop()//Comportamiento dinámico del sistema
{
  digitalWrite(13, HIGH);// Enciende el LED
  delay(500); // Espera medio segundo
  digitalWrite(13, LOW);// Apaga el LED
  delay(500); // Espera otro medio segundo
}
```




```
void setup()//Punto de partida del programa
{
  pinMode(13, OUTPUT);//El pin13 es una salida
}

void loop()//Comportamiento dinámico del sistema
{
  digitalWrite(13, HIGH);// Enciende el LED
  delay(500); // Espera medio segundo
  digitalWrite(13, LOW);// Apaga el LED
  delay(500); // Espera otro medio segundo
}
```


https://www.tinkercad.com/things/9oVlaU9CaXW-como-funciona-un-programa-ejemplo-1

Sintaxis básica de la programación en Arduino
Cuando escribimos un programa en Arduino (llamado sketch), usamos un lenguaje de programación basado en C/C++. Esto implica respetar ciertas reglas de escritura, llamadas sintaxis. Así como en español una oración debe tener sujeto y verbo, en programación cada instrucción debe seguir una forma precisa para que el sistema la entienda.

📌 **Reglas esenciales de sintaxis**

Cada instrucción termina con punto y coma (;)
Esto indica que esa línea de código ha finalizado.
Ejemplo:

```
digitalWrite(13, HIGH);
```

¿Cuándo se usa la coma (,) en Arduino?
En programación con Arduino, la coma se utiliza principalmente para separar múltiples argumentos dentro de una función. Es decir, cuando una función necesita recibir más de un dato para funcionar, se colocan separados por comas dentro de los paréntesis.
Ejemplo:

```
digitalWrite(13, HIGH);
```

digitalWrite es la función
13 es el número de pin.
HIGH es el estado que se quiere aplicar al pin.
La coma , separa esos dos parámetros.

Los bloques de código se agrupan con llaves { }
Se usan para encerrar las instrucciones que pertenecen juntas, por ejemplo dentro de setup() o loop().
Ejemplo:

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

Los comentarios se escriben con // o /* */
No afectan el funcionamiento del programa, sirven para anotar o explicar el código.

Comentario de una línea:

```
// Este pin controla un LED
```

Comentario de varias líneas:

```
/*
  Este programa
  enciende y apaga un LED
*/
```

Distingue entre mayúsculas y minúsculas
digitalWrite no es lo mismo que ~~DigitalWrite~~ o ~~digitalwrite~~. La escritura debe ser exacta.

Evitar espacios innecesarios en medio de instrucciones
Aunque se puede usar espacios en blanco para ordenar visualmente el código, no se debe interrumpir una palabra clave o nombre de función.

Analogía educativa
La sintaxis en programación es como la ortografía y la gramática en una lengua: si escribimos mal una palabra o no ponemos los signos correctos, el mensaje no se entiende. Del mismo modo, si nos olvidamos un punto y coma o escribimos mal una función, el Arduino no sabrá qué hacer.

Ejemplo aplicado: Semáforo con tres LEDs

Problemática
Imaginemos que queremos simular un semáforo de cruce peatonal simple, utilizando tres LEDs: rojo, amarillo y verde. Cada LED representa una fase del semáforo, y debe encenderse en un orden específico, respetando los tiempos de espera entre cada uno.

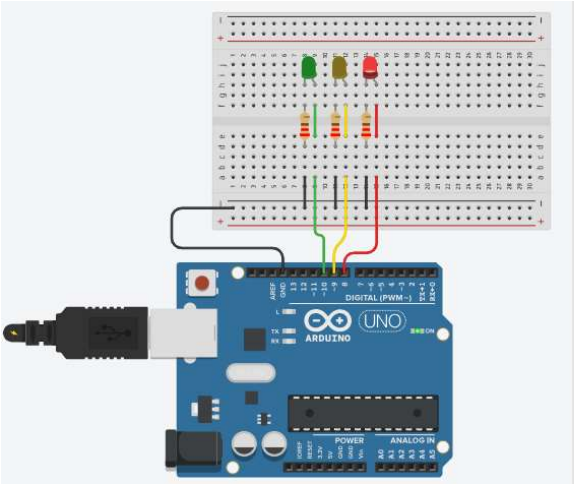
Este tipo de ejercicio permite a los estudiantes relacionar programación secuencial con automatismos del mundo real, como los que se encuentran en cruces urbanos, entradas de talleres o espacios escolares.

Requisitos del sistema
El LED rojo debe encenderse durante 4 segundos.
Luego, se apaga el rojo y se enciende el LED amarillo por 1 segundo.
Finalmente, se apaga el amarillo y se enciende el LED verde por 3 segundos.
Al terminar el ciclo, todo vuelve a empezar automáticamente.

Materiales físicos
Arduino Uno
3 LEDs (rojo, amarillo, verde)
3 resistencias de 220 Ω
Protoboard
Cables de conexión

Esquema de conexión

LED	Pin Arduino	Resistencia
Rojo	8	220 Ω
Amarillo	9	220 Ω
Verde	10	220 Ω



```ino
void setup() {
  pinMode(8, OUTPUT);   // LED rojo
  pinMode(9, OUTPUT);   // LED amarillo
  pinMode(10, OUTPUT);  // LED verde
}

void loop() {
  // Fase 1: Rojo encendido
  digitalWrite(8, HIGH);
  digitalWrite(9, LOW);
  digitalWrite(10, LOW);
  delay(4000);  // 4 segundos

  // Fase 2: Amarillo encendido
  digitalWrite(8, LOW);
  digitalWrite(9, HIGH);
  digitalWrite(10, LOW);
  delay(1000);  // 1 segundo

  // Fase 3: Verde encendido
  digitalWrite(8, LOW);
  digitalWrite(9, LOW);
  digitalWrite(10, HIGH);
  delay(3000);  // 3 segundos
}
```

https://www.tinkercad.com/things/gPMp4r8ifQp-semaforo-con-tres-leds-2

📌 **Observaciones didácticas**
El código está simplificado para enfocarse en la estructura del programa, no en la abstracción de datos.
Los números de pines están escritos directamente, lo cual es útil al inicio, pero luego será recomendable usar constantes para mayor claridad y mantenimiento del código. Puede probarse fácilmente en Tinkercad Circuits.

🔄 **De salida a entrada: ¿Cómo usar una entrada digital?**

📘 **¿Qué es una entrada?**
Una entrada digital es un pin configurado para leer una señal externa, como la activación de un botón, un sensor o un contacto.
Si la señal es alta (HIGH) → el pin recibe 5 V.
Si la señal es baja (LOW) → el pin recibe 0 V (GND).

📌 **El sistema lee la entrada y luego decide qué hacer (encender un LED, activar un motor, etc.).**

Ejemplo: encender un LED al presionar un botón

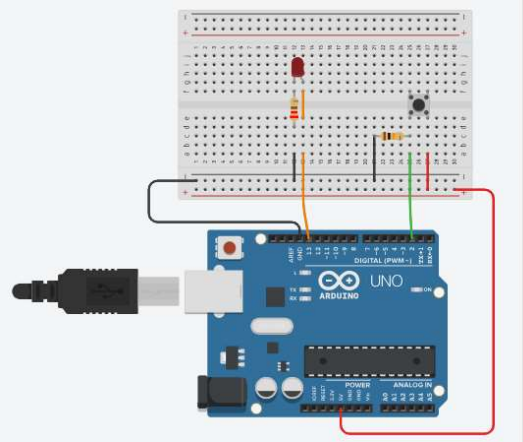
🎯 **Objetivo:**
Cuando el usuario presiona un botón, se enciende un LED.
Cuando suelta el botón, el LED se apaga.

Componentes:
Arduino Uno
1 botón (pulsador)
1 resistencia de 10 kΩ (pull-down)

1 LED
1 resistencia de 220 Ω (protección para el LED)
Cables + protoboard

🔌 Conexiones físicas:

Elemento	Conexión
Botón (pata 1)	Pin digital **2** de Arduino
Botón (pata 2)	**+5 V**
Resistencia de 10 kΩ	Entre **pin 2 y GND** (pull-down)
LED (ánodo, pata larga)	Pin digital **13**
LED (cátodo, pata corta)	Resistencia de **220 Ω** → GND



```
void setup() //Punto de partida del programa
{
  pinMode(2, INPUT);     // Configura el pin 2 como entrada digital (donde se conecta el botón)
  pinMode(13, OUTPUT);   // Configura el pin 13 como salida digital (donde se conecta el LED)
}

void loop() //Comportamiento dinámico del sistema
{
  // Lee el estado del botón (pin 2)
  // Si está presionado, el valor será HIGH (5 V)
  if (digitalRead(2) == HIGH) {
    digitalWrite(13, HIGH);  // Enciende el LED (pone el pin 13 en estado alto)
  } else {
    digitalWrite(13, LOW);   // Apaga el LED (pone el pin 13 en estado bajo)
  }
}
```

https://www.tinkercad.com/things/luqPgpAs5mc-de-salida-a-entrada-3

En este programa aparece por primera vez la estructura condicional if / else. Aunque aquí la usamos de forma simple para tomar una decisión (encender o apagar el LED), en las próximas lecciones profundizaremos en su lógica, estructura y uso en diferentes contextos, como parte fundamental de la programación de sistemas robóticos.

🔍 ¿Qué hace este programa?
Lee el estado del pin 2 (donde está conectado el botón).
Si detecta que el botón está presionado (recibe 5 V), enciende el LED.
Si el botón no está presionado, lo apaga.

Por qué se usa la resistencia de 10 kΩ?
Es una resistencia pull-down.
Evita que el pin 2 quede "flotando" cuando el botón no está presionado, asegurando que lea un valor LOW (0 V).
Sin ella, el comportamiento puede ser inestable.

Reflexión pedagógica
Este ejemplo permite introducir la interacción entre humano y sistema robótico, y construir la lógica de:
Entrada → Decisión → Acción

📖 Lectura reflexiva: Programar para construir sentido técnico

Enseñar programación en robótica no es solo enseñar a escribir líneas de código. Es proponer a los estudiantes un cambio en la forma de pensar los procesos técnicos: pasar de hacer "por costumbre" a diseñar "por lógica".

Cuando un sistema se comporta de manera predecible —enciende un LED, reacciona ante un botón, repite una secuencia de luces— no está simplemente ejecutando acciones. Está respondiendo a una estructura de control, a un modelo mental que el programador o programadora diseñó previamente. Ese modelo no surge de la intuición: se aprende, se prueba, se mejora.
La estructura básica de todo programa en Arduino (setup() y loop()) refleja dos ideas clave que vale la pena subrayar:

Preparar el sistema: Antes de actuar, hay que configurar. ¿Cuántas veces en el mundo técnico esto se omite?
Observar, decidir y actuar en ciclo: la esencia de cualquier automatismo real.

Desde esta perspectiva, enseñar programación robótica no es una cuestión de moda ni de digitalización. Es una estrategia pedagógica para que los y las estudiantes lean procesos complejos como sistemas organizados, con causas y consecuencias, y no como cajas negras que “funcionan solas”.

Cuando comprendemos qué significa escribir una instrucción, o decidir cuándo se enciende una salida, estamos más cerca de transformar el aula técnica en un laboratorio de pensamiento sistemático y creativo.

Programar no es solo decirle a una máquina qué hacer. Es aprender a pensar cómo hacer que algo funcione de manera confiable, lógica y abierta a ser mejorado.