SEMANA 1

📘 Lectura Informativa: Introducción a Arduino e Interrupciones

Tema 1: ¿Qué es Arduino?

Arduino es una plataforma de desarrollo de hardware y software de código abierto, orientada principalmente al diseño de prototipos electrónicos. Está pensada para ser accesible tanto para principiantes como para ingenieros experimentados, permitiendo fácilmente crear proyectos interactivos con sensores, motores, luces, pantallas y más.

La base de Arduino es un microcontrolador programable, que viene montado en una placa física (como el Arduino Uno, Nano o Mega) y se programa mediante un entorno gráfico sencillo llamado Arduino IDE.

Lo que hace especial a Arduino es su simplicidad, comunidad activa, bajo costo y compatibilidad con miles de sensores y módulos del mercado.

Tema 2: Componentes de Hardware de Arduino

Una placa Arduino típica incluye:

- **Microcontrolador**: El cerebro de la placa, como el ATmega328P en el Arduino Uno.
- **Memoria Flash**: Donde se almacena el programa cargado.
- **Puertos USB**: Para comunicación serial y carga del código desde la computadora.
- **Regulador de voltaje**: Permite alimentar la placa con voltajes externos seguros.
- **Temporizadores/Contadores**: Para medir tiempos o generar señales PWM.
- **Pines I/O**: Entradas/salidas digitales y analógicas para conectar dispositivos externos.

También pueden incluir elementos adicionales como convertidores analógico-digitales (ADC), comunicación inalámbrica (en modelos como el MKR WiFi 1010), y más.

Tema 3: Familia de Placas Arduino

Arduino ofrece una gama amplia de placas según las necesidades del proyecto:

- **Arduino Uno**: Ideal para principiantes. Tiene 14 pines digitales, 6 análogos, y usa el ATmega328P.
- **Arduino Nano**: Más pequeño que el Uno, muy útil en proyectos compactos.
- **Arduino Mega**: Con más pines digitales y memoria, ideal para proyectos complejos.
- **Arduino Leonardo**: Similar al Uno pero con controlador USB integrado.
- **Arduino Due**: Primer modelo con procesador ARM, soporta 32 bits.
- **Arduino MKR Zero / MKR WiFi 1010**: Placas modernas con funciones avanzadas como entrada SD, conexión Wi-Fi, etc.

Cada placa tiene ventajas específicas, por lo que elegir la adecuada dependerá del tipo de proyecto.

Tema 4: Arduino IDE

El **IDE de Arduino** (Entorno de Desarrollo Integrado) es un programa gratuito disponible para Windows, macOS y Linux. Permite escribir, compilar (verificar) y cargar programas en las placas Arduino.

Su interfaz es intuitiva, con botones claros para:

- **Verificar (✓)**: Comprobar errores de sintaxis.
- **Cargar (→)**: Enviar el código a la placa seleccionada.
- **Serial Monitor (🔍)**: Ver datos que envía el Arduino a través del puerto serie.

También permite instalar librerías adicionales desde la "Biblioteca del IDE" o gestionar tarjetas personalizadas desde el "Administrador de placas".

Tema 5: Puertos y Pines del Arduino

Los pines son las conexiones físicas que permiten a Arduino interactuar con otros componentes electrónicos. Se dividen en:

- **Pines Digitales**: Trabajan en modo entrada o salida con valores binarios (0 o 1).

- **Pines Analógicos**: Leen valores analógicos (0–1023 en Arduino Uno) y también pueden usarse como digitales.
- **Pines PWM**: Ofrecen salidas de onda modulada en anchura de pulso, útiles para controlar velocidad de motores o intensidad LED.
- **VCC y GND**: Proveen voltaje (5V o 3.3V) y tierra para alimentar circuitos externos.

Además, hay puertos internos para comunicación serial (UART), SPI (para comunicación rápida con periféricos), y I²C (comunicación en bus para múltiples dispositivos).

Tema 6: Sintaxis de Arduino

El lenguaje usado en Arduino está basado en C/C++. Algunas funciones básicas incluyen:

- `setup()` : Función que se ejecuta una vez al iniciar el programa. Se usa para inicializar configuraciones.
- `loop()` : Función principal que se repite indefinidamente. Aquí se coloca la lógica del programa.
- `pinMode(pin, modo)` : Define si un pin será entrada (`INPUT`) o salida (`OUTPUT`).
- `digitalWrite(pin, valor)` : Enviar un nivel alto (`HIGH`) o bajo (`LOW`) a un pin.
- `delay(ms)` : Detiene el programa durante los milisegundos indicados.
- `analogRead(pin)` : Lee el valor analógico de un pin (entre 0 y 1023).
- `analogWrite(pin, valor)` : Genera una señal PWM (valores entre 0 y 255).

Es importante tener en cuenta que el código debe estar bien estructurado y seguir buenas prácticas de programación.

Tema 7: Subida y Ejecución de Código

Para subir código a Arduino, debes:

1. Conectar la placa via USB.
2. Seleccionar el modelo correcto en Herramientas > Placa.
3. Seleccionar el puerto COM correspondiente.
4. Hacer clic en "Cargar".

Una vez cargado, el sketch se ejecuta automáticamente desde `setup()` y luego entra en ciclo en `loop()` hasta que se desconecte o se cargue otro programa. Si hay errores, el compilador

mostrará mensajes para corregirlos.

Tema 8: Concepto de Interrupciones

Las **interrupciones** son mecanismos que permiten al microcontrolador reaccionar ante eventos importantes sin necesidad de estar verificándolos constantemente. Cuando ocurre una interrupción, el programa principal se pausa y se ejecuta una rutina especial llamada ISR (Interrupt Service Routine).

Esto es vital en sistemas embebidos donde se requiere respuesta rápida y precisa frente a cambios en sensores, pulsadores o temporizadores.

Tema 9: Interrupciones de Hardware y Software

Hay dos tipos principales de interrupciones:

- **Interrupciones de Hardware**: Son generadas por componentes externos conectados a la placa, como un botón o sensor. Por ejemplo, cuando un usuario presiona un botón, puede enviar una señal que active una interrupción.
- **Interrupciones de Software**: Se generan dentro del propio código mediante comandos o condiciones lógicas. No dependen de eventos físicos, sino de estados internos del programa.

Ambas tienen aplicaciones diferentes, siendo las de hardware las más comunes en proyectos reales.

Tema 10: Interrupciones Internas y Externas

- **Interrupciones Internas**: Se originan dentro del microcontrolador, como errores o eventos de temporización (timer overflow). También pueden ser causadas por operaciones matemáticas inválidas o fallos del sistema.
- **Interrupciones Externas**: Se generan fuera del microcontrolador, generalmente mediante cambios en los pines definidos como fuentes de interrupción (por ejemplo, el pin 2 o 3 en el Arduino Uno).

Tema 11: Excepciones

En términos de programación de bajo nivel, las **excepciones** son eventos anormales que detienen la ejecución normal del programa. Aunque no son comunes en Arduino como en sistemas operativos, pueden surgir situaciones como:

- División entre cero
- Acceso ilegal a memoria
- Overflow de variables

Estas excepciones deben manejarse cuidadosamente para evitar reinicios no deseados o comportamientos erráticos.

Tema 12: Condiciones de Disparo

Las interrupciones pueden activarse por distintos tipos de estímulos eléctricos conocidos como **condiciones de disparo**, tales como:

- **Flanco ascendente (Rising Edge)**: Cambio de LOW a HIGH.
- **Flanco descendente (Falling Edge)**: Cambio de HIGH a LOW.
- **Cambio (Change)**: Cualquier cambio en el estado del pin.
- **Nivel bajo (Low Level)**: Mientras el pin esté en estado LOW.

Elegir el tipo de disparo adecuado depende del evento que se quiera detectar.

Tema 13: Rutinas de Servicio de Interrupción (ISR)

Las **ISR (Interrupt Service Routines)** son pequeñas funciones que se ejecutan cuando ocurre una interrupción. Estas deben cumplir ciertas reglas:

- **Ser cortas y rápidas**: Ya que detienen temporalmente el flujo principal del programa.
- **No usar funciones bloqueantes como delay() o Serial.println() dentro de ellas**.
- **No usar primitivas de tiempo prolongado**: Como esperas largas o bucles grandes.

Por eso, en lugar de hacer todo el trabajo dentro de la ISR, suele usarse para cambiar el estado de una variable o activar un flag que se maneje luego en el loop().

Tema 14: Interrupciones en Arduino

Arduino permite habilitar y configurar interrupciones usando funciones como:

```
attachInterrupt(digitalPinToInterrupt(pin), nombre_ISR, modo);
```

Donde:

- `pin` es el número del pin físico (ejemplo: 2 o 3 en Arduino Uno).
- `nombre_ISR` es el nombre de la función que actúa como ISR.
- `modo` define cómo se dispara: `RISING`, `FALLING`, `CHANGE`, o `LOW`.

Y para deshabilitar una interrupción:

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

Esto permite construir sistemas reactivos, como alarmas, contadores, o sistemas de control industrial básico.

✅ Conclusión

Arduino es una herramienta poderosa y versátil que combina hardware y software para desarrollar soluciones innovadoras. Dominar sus conceptos básicos, desde el uso del IDE hasta la gestión de interrupciones, permite crear sistemas inteligentes capaces de reaccionar a estímulos del entorno con precisión y eficiencia.

Este conjunto de habilidades te servirá como base para proyectos más complejos en electrónica y automatización, ya sea en el ámbito educativo, profesional o creativo.

🔗 Referencias

- Arduino Official Website
- Arduino Reference Documentation
- Interrupciones en Arduino – Artículo técnico
- Libro: *Beginning Arduino* – Michael McRoberts
- Documentación ATmega328P (Arduino Uno)
- Foros oficiales de Arduino: https://forum.arduino.cc/