MANUAL DE PRÁCTICAS DE LABORATORIO

Programación de Interfaces

Laboratorio de Programación

Programa Académico	**Ingeniero en Software**
Plan de Estudios	**2021**
Fecha de elaboración	**13/06/2025**
Versión del Documento	**1.0**

Universidad Estatal de Sonora
La Fuerza del Saber Estimulará mi Espíritu

Dra. Martha Patricia Patiño Fierro
Rectora

Mtra. Ana Lisette Valenzuela Molina
Encargada del Despacho de la Secretaría General Académica

Mtro. José Antonio Romero Montaño
Secretario General Administrativo

Lic. Jorge Omar Herrera Gutiérrez
Encargado de Despacho de Secretario General de Planeación

Tabla de contenido

INTRODUCCIÓN

Como parte de las herramientas esenciales para la formación académica de los estudiantes de la Universidad Estatal de Sonora, se definen manuales de práctica de laboratorio como elemento en el cual se define la estructura normativa de cada práctica y/o laboratorio, además de representar una guía para la aplicación práctica del conocimiento y el desarrollo de las competencias clave en su área de estudio. Su diseño se encuentra alineado con el modelo educativo institucional, el cual privilegia el aprendizaje basado en competencias, el aprendizaje activo y la conexión con escenarios reales.

Con el propósito de fortalecer la autonomía de los estudiantes, su pensamiento crítico y sus habilidades para la resolución de problemas, las prácticas de laboratorio integran estrategias didácticas como el aprendizaje basado en proyectos, el trabajo colaborativo, la experimentación guiada y el uso de tecnologías educativas. De esta manera, se promueve un proceso de enseñanza-aprendizaje dinámico, en el que los estudiantes no solo adquieren conocimientos teóricos, sino que también desarrollan habilidades prácticas y reflexivas para su desempeño profesional.

Señalar en este apartado brevemente los siguientes elementos según corresponda:

- Propósito del manual
- Justificación de su uso en el programa académico
- Competencias a desarrollar
    - **Competencias blandas:** Habilidades transversales que se refuerzan en las prácticas, como la comunicación, el trabajo en equipo, el uso de tecnologías, etc.
    - **Competencias disciplinares:** Conocimientos específicos del área del laboratorio, incluyendo fundamentos teóricos y habilidades técnicas.
    - **Competencias profesionales:** Aplicación de los conocimientos adquiridos en escenarios reales o simulados, en concordancia con el perfil de egreso del programa.

IDENTIFICACIÓN

Nombre de la Asignatura		PROGRAMACION DE INTERFACES	
Clave	061CP039	Créditos	5
Asignaturas Antecedentes	061CP003	Plan de Estudios	2021

Área de Competencia	Competencia del curso
Desarrollar software y servicios de soporte técnico y redes, con la finalidad de solucionar problemas y agilizar procesos en la toma de decisiones en empresas públicas y privadas, bajo estándares de calidad nacional e internacional, a través del análisis de problemas, comunicación, liderazgo e innovación.	Construir unidades de cómputo utilizando dispositivos embebidos para la automatización de procesos dentro de la organización, cumpliendo con los estándares indicados por los fabricantes, utilizando técnicas y herramientas de programación de sistemas que permitan mejorar la calidad y eficiencia de los procesos de una manera innovadora.

Carga Horaria de la asignatura

Horas Supervisadas			Horas Independientes	Total de Horas
Aula	Laboratorio	Plataforma		
2	1	1	2	6

Consignación del Documento

Unidad Académica	Unidad Académica Navojoa
Fecha de elaboración	13/06/2025
Responsables del diseño	MATI JESUS RAMON LOPEZ SANCHEZ
Validación	
Recepción	Coordinación de Procesos Educativos

MATRIZ DE CORRESPONDENCIA

Señalar la relación de cada práctica con las competencias del perfil de egreso

PRÁCTICA	PERFIL DE EGRESO
Solución de ejercicios de Arduino	Desarrollar software con la finalidad de agilizar los procesos y la toma de decisiones en empresas públicas y privadas, bajo estándares de calidad nacional e internacional con enfoque de liderazgo. Desarrollar soporte y asistencia técnica para la prevención y corrección de problemas en los sistemas de cómputo, atendiendo los requerimientos y políticas de la organización, garantizando la optimización y el uso responsable de los recursos. Aplicar soluciones e innovaciones tecnológicas con la finalidad de automatizar los procesos, atendiendo los principios de la organización y gestión efectiva de la información en los departamentos que así lo requieran, poniendo en práctica sus habilidades de trabajo en equipo y planeación.
Práctica de ejercicios adquisición de señales	Desarrollar soporte y asistencia técnica para la prevención y corrección de problemas en los sistemas de cómputo, atendiendo los requerimientos y políticas de la organización, garantizando la optimización y el uso responsable de los recursos. Aplicar soluciones e innovaciones tecnológicas con la finalidad de automatizar los procesos, atendiendo los principios de la organización y gestión efectiva de la información en los departamentos que así lo requieran, poniendo en práctica sus habilidades de trabajo en equipo y planeación.
Práctica de ejercicios actuadores con Arduino.	Desarrollar software con la finalidad de agilizar los procesos y la toma de decisiones en empresas públicas y privadas, bajo estándares de calidad nacional e internacional con enfoque de liderazgo. Desarrollar soporte y asistencia técnica para la prevención y corrección de problemas en los sistemas de cómputo, atendiendo los requerimientos y políticas de la organización, garantizando la optimización y el uso responsable de los recursos. Aplicar soluciones e innovaciones tecnológicas con la finalidad de automatizar los procesos, atendiendo los principios de la organización y gestión efectiva de la información en los departamentos que así lo requieran, poniendo en práctica sus habilidades de trabajo en equipo y planeación.

PRÁCTICA	PERFIL DE EGRESO
Practica integración de Arduino con otros entornos.	Desarrollar software con la finalidad de agilizar los procesos y la toma de decisiones en empresas públicas y privadas, bajo estándares de calidad nacional e internacional con enfoque de liderazgo. Desarrollar soporte y asistencia técnica para la prevención y corrección de problemas en los sistemas de cómputo, atendiendo los requerimientos y políticas de la organización, garantizando la optimización y el uso responsable de los recursos. Implementar redes de cómputo enlazando las diferentes áreas de la organización para compartir recursos, bajo los estándares de control de calidad nacional e internacional, garantizando la seguridad de la información para la toma de decisiones mediante su capacidad de análisis de problemas e innovación. Aplicar soluciones e innovaciones tecnológicas con la finalidad de automatizar los procesos, atendiendo los principios de la organización y gestión efectiva de la información en los departamentos que así lo requieran, poniendo en práctica sus habilidades de trabajo en equipo y planeación. Crear bases de datos para una gestión eficiente de la información, garantizando la integridad y seguridad de los datos, atendiendo los requerimientos de la organización con un sentido de liderazgo e innovación.

NORMAS DE SEGURIDAD Y BUENAS PRÁCTICAS

Reglamento general del laboratorio

Acceso a Laboratorio de Programación

- El acceso a las salas de trabajo será sólo a las horas que no haya clases.
- Deberá entrar sólo con sus documentos de trabajo, el resto de sus cosas deberá dejarlo a la entrada del laboratorio de cómputo.
- No deberá entrar a las salas de trabajo con alimentos, bebidas o fumando.
- Queda estrictamente prohibido introducir animales.

Permanencia

- Antes de trabajar se debe revisar el equipo de cómputo donde fue designado y reportar alguna anomalía al encargado de la sala; si posteriormente se detecta alguna falla no reportada, se le responsabilizará sobre ésta.
- No deberá mover el equipo de cómputo ni mobiliario de su lugar bajo ningún motivo; verifique en su apartado el equipo que va a necesitar para su trabajo.
- El usuario que dañe el equipo de cómputo, sus instalaciones o cambie su configuración del software, incluso protectores de pantalla o colores de ventanas quedará responsabilizado de pagar su costo de reparación o adquisición.
- El comportamiento de todo usuario no debe ir en contra de la moral y las buenas costumbres dentro de las diferentes salas de cómputo.
- El usuario deberá mantener limpia su área de trabajo.
- El uso del equipo de cómputo es exclusivamente para fines didácticos y de investigación, por lo que se prohíbe el uso de juegos, trabajos personales ó de terceros con fines de lucro.
- Solo se permitirá el trabajo individual por computadora, a excepciones de aquellos cursos ó clases impartidas por instructores ó profesores.
- Los usuarios no podrán ingresar ningún tipo de radio, grabadora ó elementos que produzcan ruido y/o elementos magnéticos.
- Los estudiantes deben guardar respeto mantener los canales de comunicación y demás normas establecidas con los monitores y responsables del centro de cómputo y estos para con ellos.
- El acceso a los laboratorios es para el usuario que haya solicitado el servicio, bajo ninguna circunstancia podrán estar dos personas trabajando conjuntamente en una computadora.

Uso adecuado del equipo y materiales

- El equipo periférico que se presta, es solo para uso durante la sesión de trabajo por lo que deberá de devolverlo al término de esta.
- Para asesorías ó dudas con respecto al uso del equipo, debe dirigirse al responsable en turno dentro de la sala donde se encuentre ó bien dirigirse al personal encargado del laboratorio
- El usuario no debe desconectar ni destapar los ratones de las computadoras, las impresoras ó cualquier equipo periférico que se tenga instalado.
- Es obligación del usuario saber operar el equipo de cómputo y periféricos; debe consultar con el personal del servicio social o leer los folletos sobre el uso de periféricos distribuidos para tal efecto; ya que el daño por el mal uso será responsabilidad del usuario.
- En caso de necesitar tóner la impresora notifique al responsable en turno ó bien al encargado del laboratorio, para que el la verifique y éste sea remplazado.

Procedimientos en caso de emergencia

- Evacuación ordenada por rutas señalizadas.
- Uso de extintores solo por personal capacitado.
- Reporte inmediato de accidentes al personal responsable.

RELACIÓN DE PRÁCTICAS DE LABORATORIO POR ELEMENTO DE COMPETENCIA

Elemento de Competencia al que pertenece la práctica	**EC II** Diseñar programas de cómputo mediante la placa de desarrollo Arduino utilizando sensores y actuadores conectados a un sistema de cómputo, cumpliendo con las especificaciones y estándares de la organización, demostrando capacidad de análisis, resolución de problemas y trabajo en equipo.

PRÁCTICA	NOMBRE	COMPETENCIA
Práctica No. 1	Solución de ejercicios de Arduino	Resolver de manera individual los ejercicios propuestos por el facilitador para comprender los conceptos básicos de programación de Arduino, utilizando la plataforma de desarrollo Arduino IDE, cumpliendo con los estándares de desarrollo de software vigentes y fomentando el trabajo colaborativo y la transferencia de conocimientos a situaciones prácticas.
Práctica No. 2	Práctica de ejercicios adquisición de señales	Resolver de manera individual los ejercicios propuestos por el facilitador para implementar las librerías para adquisición de señales que provee la plataforma de programación de Arduino, cumpliendo con los estándares de desarrollo de software vigentes y fomentando el trabajo colaborativo y la transferencia de conocimientos a situaciones prácticas.
Práctica No. 3	Práctica de ejercicios actuadores con Arduino.	Resolver de manera individual los ejercicios propuestos por el facilitador para implementar la programación de actuadores utilizando la plataforma de desarrollo Arduino IDE, cumpliendo con los estándares de desarrollo de software vigentes y fomentando el trabajo colaborativo y la transferencia de conocimientos a situaciones prácticas.

Elemento de Competencia al que pertenece la práctica	**EC III** Desarrollar un prototipo construido mediante sensores, actuadores y placa de desarrollo Arduino, que permita interactuar con un sistema de cómputo y su entorno, para lograr la automatización de algún proceso con el fin de mejorar su desempeño dentro de la organización, demostrando un alto sentido de responsabilidad, innovación y trabajo en equipo.

PRÁCTICA	NOMBRE	COMPETENCIA
Práctica No. 1	Practica integración de Arduino con otros entornos.	Realizar en equipo las prácticas propuestas por el facilitador para implementar la integración de Arduino con las plataformas de programación (Visual Studio, Gambas, JAVA) cumpliendo con los estándares de desarrollo de software vigentes y fomentando el trabajo colaborativo y la transferencia de conocimientos a situaciones prácticas.

UES
Universidad Estatal de Sonora
La Fuerza del Saber Estimulará mi Espíritu

PRÁCTICAS

NOMBRE DE LA PRÁCTICA	Solución de ejercicios de Arduino
COMPETENCIA DE LA PRÁCTICA	Resolver de manera individual los ejercicios propuestos por el facilitador para comprender los conceptos básicos de programación de Arduino, utilizando la plataforma de desarrollo Arduino IDE, cumpliendo con los estándares de desarrollo de software vigentes y fomentando el trabajo colaborativo y la transferencia de conocimientos a situaciones prácticas.

FUNDAMENTO TÉORICO

Arduino es una plataforma de hardware libre basada en microcontroladores, ampliamente utilizada en la enseñanza de la programación y la electrónica debido a su bajo costo, facilidad de uso y capacidad para materializar conceptos abstractos en aplicaciones tangibles. Su integración en la formación académica potencia el desarrollo de habilidades prácticas, creatividad, pensamiento crítico y resolución de problemas, aspectos esenciales en la carrera de Ingeniería de Software y otras áreas tecnológicas.

El aprendizaje de programación con Arduino permite a los estudiantes:

- Aplicar conceptos básicos de algoritmia (secuencias, condicionales, bucles, funciones) en un entorno físico, facilitando la comprensión y la motivación.
- Desarrollar prototipos funcionales que vinculan la teoría con la práctica, fortaleciendo la transferencia de conocimientos a contextos reales.
- Fomentar la colaboración y la socialización de soluciones, lo cual enriquece el proceso de aprendizaje y la adquisición de buenas prácticas de desarrollo de software.

Los estándares de desarrollo de software, como la documentación clara, modularidad y la gestión de versiones son aplicables a proyectos con Arduino y contribuyen a la formación de competencias profesionales alineadas con marcos reconocidos como COBIT, ITIL e ISO 27000.

MATERIALES, EQUIPAMIENTO Y/O REACTIVOS

- Placa Arduino (modelo UNO, Nano o Mega según disponibilidad)
- Cable USB para conexión y programación de la placa
- Protoboard (breadboard) para montaje de circuitos sin soldadura
- Cables jumper (macho-macho, macho-hembra) para conexiones en protoboard
- LEDs de varios colores para prácticas de salida digital
- Resistencias (valores comunes como 220Ω, 330Ω, 1kΩ, 10kΩ)
- Pulsadores (botones) para entradas digitales
- Potenciómetros para entradas analógicas

PROCEDIMIENTO O METODOLOGÍA

Manejar las instrucciones para interactuar con las entradas y salidas digitales de la placa de Arduino, concretamente utilizarás la instrucción de escritura, además, aprendas a manejar las instrucciones de gestión del tiempo de Arduino, montarás tu primer circuito electrónico y realizarás tu primer programa con el lenguaje de programación de Arduino.

La realización del montaje del circuito electrónico tienes que realizarla conforme a los siguientes pasos, recuerda tener desconectada de la computadora la placa cuando empieces a montarlo:

1. Conecta el LED a la protoboard.
2. Utiliza un cable de conexión macho-macho para conectar el pin GND de la placa de Arduino con el cátodo del LED. El cable lo tienes que conectar directamente al pin GND y mediante la protoboard al LED.
3. Conecta la resistencia con el ánodo del LED utilizando la protoboard.
4. Utiliza un cable de conexión macho-macho para conectar el pin digital número 2 de la placa de Arduino con el extremo libre de la resistencia. El cable lo tienes que conectar directamente al pin y mediante la protoboard a la resistencia.

Código fuente del programa

```
// Declaración de constantes
#define BLUE 2         // Pin al que está conectado el LED Azul
#define DELAYTIME 500  // Milisegundos de parada para la función
delay
  // Configuración de pines
  void
  setup() {
  pinMode(BLUE, OUTPUT);
  digitalWrite(BLUE, LOW);
}
void loop() {
  // Encender LED
  digitalWrite(BLUE, HIGH);
  // Esperar medio segundo
  delay(DELAYTIME);
  // Apagar LED
  digitalWrite(BLUE, LOW);
  // Esperar medio segundo
  delay(DELAYTIME);
}
```

RESULTADOS ESPERADOS

Una vez has montado el circuito y has escrito el código fuente del programa llega el momento de subirlo a la placa de Arduino. Por tanto, conecta la placa a la computadora y ejecuta la operación Subir desde la barra de accesos directos. Una vez haya acabado la subida del programa, este empezará su ejecución. La ejecución del programa es muy sencilla, el LED que has utilizado empezará a encenderse y a apagarse de forma continua.

ANÁLISIS DE RESULTADOS
• ¿Qué conceptos básicos de programación aplicaste en la solución de los ejercicios? • ¿El circuito y el programa funcionaron como esperabas? ¿Por qué? • ¿Qué errores encontraste y cómo los corregiste? • ¿Cómo documentaste tu código para facilitar su comprensión y reutilización? • ¿Qué aprendiste sobre el proceso de depuración y mejora de programas en Arduino?

CONCLUSIONES Y REFLEXIONES

ACTIVIDADES COMPLEMENTARIAS
Parpadeo de LED con Frecuencia Variable: Hacer que un LED parpadee, pero en lugar de usar delay(), utilizar la función millis() para controlar los tiempos y permitir que otras tareas se ejecuten simultáneamente. Requerimientos: El LED debe alternar entre encendido y apagado cada 500 ms sin bloquear el programa, Agregar un segundo LED que parpadee a una frecuencia diferente (ej. cada 200 ms) usando la misma técnica. Conceptos Reforzados: millis() para temporización no bloqueante, multitarea simple. **Contador de Pulsaciones de Botón:** Implementar un contador que incremente su valor cada vez que se presione un botón. El conteo debe mostrarse en el monitor serial. Requerimientos: Evitar rebotes del botón (debouncing) para asegurar un solo incremento por pulsación, Reiniciar el contador a 0 si se mantiene el botón presionado por más de 3 segundos. Conceptos Reforzados: Detección de flancos, debouncing, control de estados, if-else anidados. **Secuencia de LEDs con Múltiples Modos:** Controlar una secuencia de 4 LEDs. Al presionar un botón, la secuencia de parpadeo debe cambiar a un modo diferente (ej. modo 1: LEDs parpadean de izquierda a derecha; modo 2: LEDs parpadean alternadamente). Requerimientos: Implementar al menos dos modos de secuencia, Uso de variables de estado para cambiar entre modos, case o if-else if para seleccionar el modo activo. Conceptos Reforzados: Estructuras de control (switch-case o if-else if), estados de programa, modularización de código con funciones.

EVALUACIÓN Y EVIDENCIAS DE APRENDIZAJE	
Criterios de evaluación	comprensión conceptual, capacidad de programación, diseño de circuitos, resolución de problemas, creatividad y presentación.
Rúbricas o listas de cotejo para valorar desempeño	Rúbrica de Reporte de Prácticas
Formatos de reporte de prácticas	Reporte de Practicas

NOMBRE DE LA PRÁCTICA	Práctica de ejercicios adquisición de señales
COMPETENCIA DE LA PRÁCTICA	Resolver de manera individual los ejercicios propuestos por el facilitador para implementar las librerías para adquisición de señales que provee la plataforma de programación de Arduino, cumpliendo con los estándares de desarrollo de software vigentes y fomentando el trabajo colaborativo y la transferencia de conocimientos a situaciones prácticas.

FUNDAMENTO TÉORICO

La adquisición de señales es el proceso mediante el cual un sistema captura datos del entorno físico para su procesamiento digital. En sistemas embebidos, como Arduino, esto se realiza a través de sensores que convierten magnitudes físicas (temperatura, luz, presión) en señales eléctricas analógicas o digitales.

Arduino cuenta con entradas analógicas que permiten leer señales en un rango de voltajes (0-5V típicamente) mediante un convertidor analógico-digital (ADC) interno, usualmente de 10 bits, lo que significa que la señal analógica se digitaliza en valores entre 0 y 1023.

El uso de librerías estandarizadas y funciones como analogRead(pin) facilita la captura de datos, mientras que la correcta configuración y calibración del sensor es crucial para obtener mediciones precisas.

Aplicación Práctica en Arduino

Conexión del Sensor: Se conecta el sensor analógico al pin correspondiente de Arduino, asegurando alimentación y referencia de voltaje adecuadas.

Implementación del Código: Se utiliza analogRead() para capturar la señal, seguido de la conversión del valor digital a una magnitud física significativa (por ejemplo, grados Celsius para un sensor de temperatura).

Visualización y Análisis: Los datos pueden visualizarse en el monitor serial para su análisis en tiempo real o almacenarse para procesamiento posterior.

Cumplimiento de Estándares: Se recomienda seguir buenas prácticas de programación, como modularidad, documentación y validación de datos, para asegurar calidad y reproducibilidad.

La adquisición de señales en Arduino se basa en la digitalización de señales analógicas mediante ADC, permitiendo que el microcontrolador interprete y utilice información del entorno físico para aplicaciones prácticas en sistemas embebidos.

MATERIALES, EQUIPAMIENTO Y/O REACTIVOS

- Placa Arduino (Uno, Mega o similar)
- Sensores analógicos y digitales: Sensor de temperatura y humedad DHT11 o DHT22, Fotorresistencia (LDR), Sensor de pulso cardíaco (opcional)
- Resistencias (220Ω, 330Ω,10 kΩ para divisor de tensión con LDR)
- Potenciómetro (10kΩ, 47kΩ)
- Protoboard y cables de conexión
- Fuente de alimentación o cable USB para Arduino
- Computadora con Arduino IDE instalado

PROCEDIMIENTO O METODOLOGÍA

En esta práctica activaremos un diodo led mediante un potenciómetro, dependiendo del valor que

adquiera éste en un determinado momento.
Primero se deberá observar qué valores obtenemos al girar el potenciómetro a un lado y al otro del dial, recogiendo los valores en una variable y visualizándolos por el monitor serie.
Una vez que tengamos una idea de los valores, procedemos a encender el led según los siguientes valores:

- El led se activará cuando el potenciómetro marque un valor menor o igual a 100.
- El led se apagará cuando el potenciómetro marque un valor de entre 300 y 500.
- Deberá volver a encenderse cuando el potenciómetro adquiera un valor mayor de 900
-

CÓDIGO DE LA PRÁCTICA

```
/* Código de utilización de un potenciómetro */
int pot = A0;  //asignamos el pin A0 al terminal central del potenciómetro
int valor;     //en esta variable almacenaremos los valores recogidos por el
potenciómetro int led = 13;
void setup() {
  Serial.begin(9600);    //establecemos la comunicación serie
  pinMode(pot, INPUT);   //pin de entrada de datos
  pinMode(led, OUTPUT);  //pin de salida
}
void loop() {
  valor = analogRead(pot);
  Almacenamos los valores en la variable valor
    Serial.println(valor);  //imprimimos por pantalla los valores para
  determinar los rangos en las decisiones.Probamos accionando el potenciómetro y
    observamos el cambio en los valores registrados if (valor <= 100) {  //si valor
es menor de 100
    digitalWrite(led,
HIGH);                                          //activamos el led
  }
  if (valor > 300 && valor < 500) {  //si el valor está entre 300 y 500
    digitalWrite(led, LOW);          //desactivamos el led
  }
  if (valor > 900) {          //si el valor es mayor a 900
    digitalWrite(led, HIGH);  //activamos el led
  }
}
```

RESULTADOS ESPERADOS

- Lectura exitosa de señales analógicas y/o digitales desde sensores conectados a Arduino (por ejemplo, temperatura, luz, señales biológicas).
- Visualización en tiempo real de los datos adquiridos en el monitor serial o en una interfaz gráfica.

- Registro de datos en archivos para análisis posterior, por ejemplo en formato CSV, permitiendo su procesamiento en software externo como Octave o Excel.
- Interpretación de los valores adquiridos y conversión a unidades físicas (°C, %, etc.).
- Validación de la calidad de la señal (por ejemplo, aplicando media y varianza para caracterizar la señal, como en la adquisición EMG1).

ANÁLISIS DE RESULTADOS

1. ¿Qué tipo de señal adquiriste (analógica o digital) y cómo la identificaste?
2. ¿Cómo interpretaste el valor digital obtenido por el ADC? ¿Qué significa ese valor en términos de la magnitud física medida?
3. ¿Observaste variaciones o ruido en la señal? ¿A qué podrían deberse?
4. ¿El sensor respondió como esperabas ante cambios en el entorno? ¿Por qué sí o por qué no?
5. ¿Qué limitaciones identificas en la adquisición de señales con Arduino?
6. ¿Cómo podrías mejorar la precisión o estabilidad de la medición?

CONCLUSIONES Y REFLEXIONES

Relación con la teoría y aplicación en el campo profesional

ACTIVIDADES COMPLEMENTARIAS

Monitor de Temperatura y Humedad: Conectar un sensor DHT11 o DHT22 a Arduino. Adquirir lecturas de temperatura y humedad, y mostrarlas en el monitor serial.
Requerimientos: Lectura cada 2 segundos, Formatear la salida para indicar claramente la temperatura en °C y la humedad en %, Manejar posibles errores de lectura del sensor (ej. si la lectura es inválida, imprimir "Error de lectura").
Conceptos Reforzados: analogRead() (si el sensor es analógico), uso de librerías de sensores, manejo de datos y condicionales.

Medidor de Intensidad Lumínica: Utilizar una fotorresistencia (LDR) conectada a un pin analógico de Arduino para medir la intensidad lumínica del ambiente.
Requerimientos: Convertir la lectura analógica (0-1023) a un porcentaje de intensidad lumínica (0-100%), Encender un LED si la intensidad lumínica cae por debajo de un umbral predefinido (ej. 30%).
Conceptos Reforzados: Calibración de sensores, mapeo de valores, control de salida digital (digitalWrite).

Detección de Ritmo Cardíaco Básico: Solo si se dispone de un sensor de pulso cardíaco compatible con Arduino). Adquirir la señal de un sensor de pulso y visualizarla en el Serial Plotter de Arduino IDE para observar las variaciones.
Requerimientos: Investigar el funcionamiento básico del sensor de pulso, Comprender que la señal es una representación de la variación de volumen sanguíneo y no un electrocardiograma completo, Ajustar la velocidad de muestreo para una visualización adecuada.
Conceptos Reforzados: Adquisición de señales biológicas (a nivel introductorio), visualización de datos en tiempo real, importancia de la frecuencia de muestreo.

EVALUACIÓN Y EVIDENCIAS DE APRENDIZAJE	
Criterios de evaluación	comprensión conceptual, capacidad de programación, diseño de circuitos, resolución de problemas, creatividad y presentación.
Rúbricas o listas de cotejo para valorar desempeño	Rúbrica de Reporte de Prácticas
Formatos de reporte de prácticas	Reporte de Practicas

NOMBRE DE LA PRÁCTICA	Práctica de ejercicios actuadores con Arduino.
COMPETENCIA DE LA PRÁCTICA	Resolver de manera individual los ejercicios propuestos por el facilitador para implementar la programación de actuadores utilizando la plataforma de desarrollo Arduino IDE, cumpliendo con los estándares de desarrollo de software vigentes y fomentando el trabajo colaborativo y la transferencia de conocimientos a situaciones prácticas.

FUNDAMENTO TÉORICO

Los actuadores son dispositivos que convierten señales eléctricas en movimientos, cambios físicos o estados eléctricos, tales como encender un LED, mover un motor, activar un relé o modificar la posición de un servomotor. En sistemas basados en microcontroladores como Arduino, los actuadores reciben señales de control digitales o analógicas (PWM) que determinan su comportamiento.

Arduino controla actuadores mediante salidas digitales (encendido/apagado) o señales PWM que permiten variar la intensidad o posición (por ejemplo, brillo de un LED o ángulo de un servomotor). La modulación PWM emula una señal analógica variando el ciclo de trabajo de una señal digital.
El uso de actuadores está estrechamente ligado a la lectura de sensores y la lógica de control implementada en el código. Por ejemplo, un sensor de temperatura puede activar un ventilador (actuador) cuando se supera un umbral.

La programación de actuadores debe seguir principios de modularidad, manejo eficiente de recursos y documentación clara para garantizar la mantenibilidad y escalabilidad de los sistemas.
Los actuadores se conectan a pines digitales o PWM de Arduino, respetando las especificaciones eléctricas (voltaje, corriente). En algunos casos, se requieren circuitos de interfaz (transistores, drivers) para manejar cargas mayores. El uso de funciones como digitalWrite() para salidas digitales y analogWrite() para PWM. Librerías específicas, como Servo.h para servomotores, facilitan el control.

El programa debe leer entradas (sensores o botones), procesar la información y enviar señales adecuadas a los actuadores para lograr la acción deseada y verificar que el actuador responde correctamente y ajustar parámetros según sea necesario.

MATERIALES, EQUIPAMIENTO Y/O REACTIVOS

- Placa Arduino
- LEDs (diferentes colores para semáforo)
- Resistencias para LEDs (220 Ω típicamente)
- Potenciómetro (10 kΩ)
- Servomotor (ejemplo SG90)
- Botones pulsadores
- Protoboard y cables de conexión
- Fuente de alimentación o cable USB para Arduino
- Computadora con Arduino IDE instalado

PROCEDIMIENTO O METODOLOGÍA

Arduino posee librerías para manejar los motores paso a paso, pero en esta práctica vamos a explicar como ejecutar un paso de forma independiente, ya que es importante entender cómo funcionan estos motores.

La realización del montaje del circuito electrónico tienes que llevarla a cabo conforme a los siguientes pasos, recuerda tener desenchufada del ordenador la placa cuando empieces a montarlo:

1. Conecta el pulsador a la protoboard.
2. Utiliza un cable de conexión macho-macho para conectar el pin GND de la placa de Arduino con el uno de los extremos del pulsador. El cable lo tienes que conectar directamente al pin GND y mediante la protoboard al pulsador.
3. Utiliza un cable de conexión macho-macho para conectar el pin digital número 13 de la placa de Arduino con el extremo libre del pulsador. El cable lo tienes que conectar directamente al pin y mediante la protoboard al pulsador.
4. Conecta el motor paso a paso con el adaptador.
5. Utiliza un cable de conexión macho-hembra para conectar el otro pin GND de la placa de Arduino con la patilla etiquetada como - del adaptador del motor paso a paso. El cable tienes que conectarlo directamente al pin de la placa de Arduino y a la patilla del adaptador.
6. Utiliza un cable de conexión macho-hembra para conectar el pin de 5V de la placa de Arduino con la patilla etiquetada como + del adaptador del motor paso a paso. El cable tienes que conectarlo directamente al pin de la placa de Arduino y a la patilla del adaptador.
7. Utiliza un cable de conexión macho-hembra para conectar el pin digital número 8 de la placa de Arduino con la patilla etiquetada como 1N4 del adaptador del motor paso a paso. El cable tienes que conectarlo directamente al pin de la placa de Arduino y a la patilla del adaptador.
8. Utiliza un cable de conexión macho-hembra para conectar el pin digital número 9 de la placa de Arduino con la patilla etiquetada como 1N3 del adaptador del motor paso a paso. El cable tienes que conectarlo directamente al pin de la placa de Arduino y a la patilla del adaptador.
9. Utiliza un cable de conexión macho-hembra para conectar el pin digital número 10 de la placa de Arduino con la patilla etiquetada como 1N2 del adaptador del motor paso a paso. El cable tienes que conectarlo directamente al pin de la placa de Arduino y a la patilla del adaptador.
10. Utiliza un cable de conexión macho-hembra para conectar el pin digital número 11 de la placa de Arduino con la patilla etiquetada como 1N1 del adaptador del motor paso a paso. El cable tienes que conectarlo directamente al pin de la placa de Arduino y a la patilla del adaptador.

Código fuente

```
// Declaración de constantes
#define STEPMOTOR1 8  // Pin al que está conectado el primer
pin del motor paso a paso
#define STEPMOTOR2 9  // Pin al que está conectado el segundo
  pin del motor paso a paso
#define STEPMOTOR3 10  // Pin al que está conectado el tercer
    pin del motor paso a paso
#define STEPMOTOR4 11  // Pin al que está conectado el cuarto
      pin del motor paso a paso
#define BUTTON 13        // Pin al que se encuentra conectado el interruptor
#define DELAYTIME 10     // Tiempo de espera de los pasos
#define BUTTONDELAY 500  // Tiempo de espera tras pulsar el
```

```
        interruptor
  // Declaración de variables
  int changeState = 0;
// Configuración de pines
void setup() {
  pinMode(STEPMOTOR1, OUTPUT);
  pinMode(STEPMOTOR2, OUTPUT);
  pinMode(STEPMOTOR3, OUTPUT);
  pinMode(STEPMOTOR4, OUTPUT);
  pinMode(BUTTON, INPUT_PULLUP);
}
void loop() {
  // Comprueba si se ha presionado el interruptor
  if (digitalRead(BUTTON) == LOW) {
    delay(BUTTONDELAY);
    // Cambia el estado en el que se encuentra el sentido de
    giro del motor if (changeState == 0)
      changeState = 1;
    else changeState = 0;
  }
  // Da un paso hacia adelante o hacia atrás dependiendo del
  valor del sentido de giro if (changeState == 1)
    GoAhead();
  else GoBack();
  delay(DELAYTIME);
}
// Función que realizará un paso en el motor hacia atrás
void GoBack() {
  digitalWrite(STEPMOTOR1, HIGH);
  digitalWrite(STEPMOTOR2, HIGH);
  digitalWrite(STEPMOTOR3, LOW);
  digitalWrite(STEPMOTOR4, LOW);
  delay(DELAYTIME);
  digitalWrite(STEPMOTOR1, LOW);
  digitalWrite(STEPMOTOR2, HIGH);
  digitalWrite(STEPMOTOR3, HIGH);
  digitalWrite(STEPMOTOR4, LOW);
  delay(DELAYTIME);
  digitalWrite(STEPMOTOR1, LOW);
  digitalWrite(STEPMOTOR2, LOW);
  digitalWrite(STEPMOTOR3, HIGH);
  digitalWrite(STEPMOTOR4, HIGH);
  delay(DELAYTIME);
```

```
  digitalWrite(STEPMOTOR1, HIGH);
  digitalWrite(STEPMOTOR2, LOW);
  digitalWrite(STEPMOTOR3, LOW);
  digitalWrite(STEPMOTOR4, HIGH);
  delay(DELAYTIME);
}
// Función que realizará un paso en el motor hacia adelante
void GoAhead() {
  digitalWrite(STEPMOTOR1, HIGH);
  digitalWrite(STEPMOTOR2, LOW);
  digitalWrite(STEPMOTOR3, LOW);
  digitalWrite(STEPMOTOR4, HIGH);
  delay(DELAYTIME);
  digitalWrite(STEPMOTOR1, LOW);
  digitalWrite(STEPMOTOR2, LOW);
  digitalWrite(STEPMOTOR3, HIGH);
  digitalWrite(STEPMOTOR4, HIGH);
  delay(DELAYTIME);
  digitalWrite(STEPMOTOR1, LOW);
  digitalWrite(STEPMOTOR2, HIGH);
  digitalWrite(STEPMOTOR3, HIGH);
  digitalWrite(STEPMOTOR4, LOW);
  delay(DELAYTIME);
  digitalWrite(STEPMOTOR1, HIGH);
  digitalWrite(STEPMOTOR2, HIGH);
  digitalWrite(STEPMOTOR3, LOW);
  digitalWrite(STEPMOTOR4, LOW);
  delay(DELAYTIME);
}
```

RESULTADOS ESPERADOS

- Control preciso de actuadores (LEDs, servomotores, relés) mediante señales generadas por Arduino.
- Respuesta física observable: encendido/apagado de LEDs, movimiento de servomotores, etc.
- Interacción con entradas: por ejemplo, cambiar el estado de un actuador en respuesta a la lectura de un sensor o un botón.
- Implementación de secuencias lógicas (ejemplo: simulación de semáforo, control proporcional de brillo, movimiento de un servomotor según señal de entrada)

ANÁLISIS DE RESULTADOS

- ¿Cómo se comportó el actuador ante las señales de control enviadas desde Arduino?

- ¿El actuador respondió de manera inmediata y precisa? Si hubo retardo o error, ¿a qué lo atribuyes?
- ¿Qué relación existe entre el valor de entrada (sensor, potenciómetro, botón) y la respuesta del actuador?
- ¿Cómo modularías el sistema para controlar más de un actuador simultáneamente?
- ¿Qué aspectos del código o del circuito podrían optimizarse?

CONCLUSIONES Y REFLEXIONES

ACTIVIDADES COMPLEMENTARIAS

Semáforo Sencillo: Descripción: Simular el funcionamiento de un semáforo utilizando tres LEDs (rojo, amarillo, verde) conectados a pines digitales de Arduino.
Requerimientos: Secuencia: Verde (5s) -> Amarillo (2s) -> Rojo (5s) -> Rojo y Amarillo (1s) -> Verde, Uso de la función delay().
Conceptos Reforzados: digitalWrite(), control de tiempos, secuencias lógicas.

Control de Luminosidad con Potenciómetro: Conectar un potenciómetro a un pin analógico y un LED a un pin con capacidad PWM. Variar la intensidad lumínica del LED moviendo el potenciómetro.
Requerimientos: Mapear la lectura del potenciómetro (0-1023) al rango de PWM (0-255), Utilizar analogWrite() para controlar el brillo del LED.
Conceptos Reforzados: analogWrite(), PWM, mapeo de rangos, control analógico de salidas.

Servomotor Controlado por Botón: Conectar un servomotor a un pin digital de Arduino y un botón a otro. Cada vez que se presione el botón, el servomotor debe alternar entre dos posiciones predefinidas (ej. 0° y 90°).
Requerimientos: Uso de la librería Servo.h, Detección de flanco (cambio de estado) del botón para evitar múltiples activaciones con una sola pulsación.
Conceptos Reforzados: Servo.h, detección de eventos (interrupciones o lectura de estado), control de posición.

EVALUACIÓN Y EVIDENCIAS DE APRENDIZAJE	
Criterios de evaluación	Comprensión conceptual, capacidad de programación, diseño de circuitos, resolución de problemas, creatividad y presentación.
Rúbricas o listas de cotejo para valorar desempeño	Rúbrica de Reporte de Prácticas
Formatos de reporte de prácticas	Reporte de Practicas

NOMBRE DE LA PRÁCTICA	Practica integración de Arduino con otros entornos.
COMPETENCIA DE LA PRÁCTICA	Realizar en equipo las prácticas propuestas por el facilitador para implementar la integración de Arduino con las plataformas de programación (Visual Studio, Gambas, JAVA) cumpliendo con los estándares de desarrollo de software vigentes y fomentando el trabajo colaborativo y la transferencia de conocimientos a situaciones prácticas.

FUNDAMENTO TÉORICO

Arduino se comunica con otros entornos (Visual Studio, Java, Gambas, etc.) principalmente a través del puerto serial (USB), que utiliza un protocolo estándar para la transmisión de datos en serie. Este método permite la transferencia bidireccional de información en forma de bytes, que deben ser codificados y decodificados correctamente por ambos extremos para asegurar la integridad y sincronización de los datos.

La integración implica abstraer el hardware de Arduino mediante APIs y librerías que facilitan la lectura y escritura de datos, mientras que el entorno externo gestiona la interfaz de usuario, procesamiento avanzado o almacenamiento. Esta separación modular permite que cada componente se especialice en su función, facilitando el desarrollo, mantenimiento y escalabilidad del sistema.

La comunicación efectiva requiere el manejo adecuado de eventos (recepción/envío de datos) y sincronización para evitar pérdidas o corrupción de información. Esto incluye técnicas como buffers, control de flujo y protocolos simples de confirmación. La integración de Arduino con entornos de programación externos potencia el aprendizaje colaborativo y la transferencia de conocimientos prácticos, al permitir que los estudiantes desarrollen aplicaciones más complejas y realicen análisis en tiempo real, combinando hardware y software de manera efectiva

MATERIALES, EQUIPAMIENTO Y/O REACTIVOS

1. Placa Arduino
2. Sensores para adquisición (por ejemplo, DHT11/DHT22 o LDR)
3. Actuadores (LED, servomotor)
4. Cables y protoboard
5. Computadora con: Arduino IDE instalado, Entorno de desarrollo para integración (Visual Studio para C#, Gambas o Java IDE)
6. Cable USB para conexión serial Arduino-PC

PROCEDIMIENTO O METODOLOGÍA

Crear una aplicación en Visual Basic con dos botones para encender y apagar un LED conectado al Arduino.

Paso 1: Montaje del Circuito

1. Conecta el cátodo (pata corta) del LED a un pin **GND** del Arduino a través de la resistencia de 220Ω.
2. Conecta el ánodo (pata larga) del LED directamente al pin digital **13** del Arduino.

Paso 2: Codificación del Arduino

Abre el Arduino IDE, pega este código y súbelo a tu placa.

```
// Práctica 1: Recibir comandos desde el PC para controlar un LED

// Definimos el pin donde está conectado el LED
const int ledPin = 13;

void setup() {
  // Inicializamos la comunicación serial a 9600 baudios
  // Esta velocidad DEBE COINCIDIR con la de Visual Basic
  Serial.begin(9600);

  // Configuramos el pin del LED como salida
  pinMode(ledPin, OUTPUT);
}

void loop() {
  // Verificamos si hay datos disponibles para leer en el buffer serial
  if (Serial.available() > 0) {
    // Leemos el primer byte (carácter) que llegó
    char datoRecibido = Serial.read();

    // Comparamos el dato recibido para tomar una acción
    if (datoRecibido == '1') {
      // Si recibimos un '1', encendemos el LED
      digitalWrite(ledPin, HIGH);
    } else if (datoRecibido == '0') {
      // Si recibimos un '0', apagamos el LED
      digitalWrite(ledPin, LOW);
    }
  }
}
```

Paso 3: Creación de la Interfaz en Visual Basic

1. Abre Visual Studio y crea un nuevo proyecto de tipo **"Aplicación de Windows Forms (.NET Framework)"** en Visual Basic.

2. Desde el Cuadro de Herramientas, arrastra los siguientes controles al formulario:
    - Un ComboBox (lo llamaremos cmbPuertos).
    - Un Button (texto: "Conectar", nombre: btnConectar).
    - Otro Button (texto: "Encender LED", nombre: btnEncender).
    - Otro Button (texto: "Apagar LED", nombre: btnApagar).
    - Un Label para mostrar el estado (nombre: lblEstado).
    - Desde el Cuadro de Herramientas, busca y arrastra un componente SerialPort al formulario. Aparecerá en la bandeja inferior (lo dejaremos con su nombre por defecto, SerialPort1).

Paso 4: Codificación en Visual Basic

Haz doble clic en el formulario para abrir la vista de código y pega lo siguiente.

```
' Importamos la librería para manejar los puertos seriales
Imports System.IO.Ports

Public Class Form1

    ' Evento que se ejecuta cuando el formulario se carga
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        ' Deshabilitamos los botones de control hasta que nos conectemos
        btnEncender.Enabled = False
        btnApagar.Enabled = False
        btnConectar.Text = "Conectar"

        ' Buscamos y añadimos todos los puertos COM disponibles al ComboBox
        cmbPuertos.Items.Clear()
        For Each port As String In SerialPort.GetPortNames()
            cmbPuertos.Items.Add(port)
        Next
        If cmbPuertos.Items.Count > 0 Then
            cmbPuertos.SelectedIndex = 0 ' Seleccionamos el primero por defecto
        Else
            lblEstado.Text = "No se encontraron puertos COM."
            cmbPuertos.Enabled = False
            btnConectar.Enabled = False
        End If
    End Sub

    ' Evento del botón Conectar/Desconectar
```

```
    Private Sub btnConectar_Click(sender As Object, e As EventArgs) Handles
btnConectar.Click
        Try
            If btnConectar.Text = "Conectar" Then
                ' Configuramos el puerto serial
                SerialPort1.PortName = cmbPuertos.Text
                SerialPort1.BaudRate = 9600 ' ¡Debe ser la misma que en Arduino!
                SerialPort1.Open() ' Abrimos el puerto

                ' Actualizamos la interfaz
                btnConectar.Text = "Desconectar"
                lblEstado.Text = "Conectado a " & SerialPort1.PortName
                cmbPuertos.Enabled = False
                btnEncender.Enabled = True
                btnApagar.Enabled = True
            Else
                ' Si ya está conectado, lo cerramos
                SerialPort1.Close()

                ' Actualizamos la interfaz
                btnConectar.Text = "Conectar"
                lblEstado.Text = "Desconectado"
                cmbPuertos.Enabled = True
                btnEncender.Enabled = False
                btnApagar.Enabled = False
            End If
        Catch ex As Exception
            ' En caso de error (ej: puerto en uso), mostramos un mensaje
            MessageBox.Show("Error: " & ex.Message, "Error de Conexión")
        End Try
    End Sub

    ' Evento del botón para encender el LED
    Private Sub btnEncender_Click(sender As Object, e As EventArgs) Handles
btnEncender.Click
        If SerialPort1.IsOpen Then
            SerialPort1.Write("1") ' Enviamos el carácter '1'
        End If
    End Sub

    ' Evento del botón para apagar el LED
    Private Sub btnApagar_Click(sender As Object, e As EventArgs) Handles
btnApagar.Click
```

```
        If SerialPort1.IsOpen Then
            SerialPort1.Write("0") ' Enviamos el carácter '0'
        End If
    End Sub

    ' Importante: Asegurarse de cerrar el puerto al cerrar la aplicación
    Private Sub Form1_FormClosing(sender As Object, e As FormClosingEventArgs)
Handles MyBase.FormClosing
        If SerialPort1.IsOpen Then
            SerialPort1.Close()
        End If
    End Sub
End Class
```

Ejecución:

1. Asegúrate de que el Arduino esté conectado y programado.
2. Ejecuta la aplicación de Visual Basic (F5).
3. Selecciona el puerto COM correcto de tu Arduino en el ComboBox.
4. Haz clic en "Conectar". El estado debería cambiar.
5. Usa los botones "Encender LED" y "Apagar LED" para controlar el LED en tiempo real

Leer el valor de un potenciómetro conectado al Arduino y mostrarlo en la aplicación de Visual Basic.

Paso 1: Montaje del Circuito

1. Deja el circuito del LED como está.
2. Conecta el potenciómetro:
    - El pin central (wiper) al pin analógico **A0** del Arduino.
    - Uno de los pines exteriores a **5V**.
    - El otro pin exterior a **GND**.

Paso 2: Codificación del Arduino

Modifica el código del Arduino para que ahora lea el sensor y envíe los datos.

```
// Práctica 2: Enviar datos de un sensor al PC

const int potenciometroPin = A0;  // Pin analógico para el potenciómetro
```

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

void loop() {
  // Leemos el valor del potenciómetro (devuelve un valor entre 0 y 1023)
  int valorSensor = analogRead(potenciometroPin);

  // Enviamos el valor leído al PC.
  // Usamos println para añadir un salto de línea al final.
  // Esto facilita la lectura en Visual Basic.
  Serial.println(valorSensor);

  // Esperamos un poco para no saturar el puerto serial
  delay(100);  // Envía datos 10 veces por segundo
}
```

Nota: *Este código no incluye la lógica para recibir datos y controlar el LED, se enfoca solo en el envío. Puedes combinar ambos códigos si lo deseas.*

Paso 3: Modificación de la Interfaz en Visual Basic

Añade los siguientes controles al formulario existente:

- Un Label con el texto "Valor del Sensor:".
- Un TextBox (nombre: txtValorSensor, propiedad ReadOnly: True).
- Un ProgressBar (nombre: pbValorSensor).

Paso 4: Codificación en Visual Basic

Modifica el código de Visual Basic para manejar los datos que llegan desde el Arduino

```
' Añade esto en la parte superior
Imports System.IO.Ports

Public Class Form1
    ' ... (El código anterior de la Práctica 1 se mantiene aquí) ...

    ' Declaramos un delegado. Es necesario para actualizar la UI desde otro hilo.
    Private Delegate Sub DelegadoActualizarUI(texto As String)

    ' Este es el evento que se dispara CADA VEZ que llegan datos al puerto serial
    Private Sub SerialPort1_DataReceived(sender As Object, e As
```

```
SerialDataReceivedEventArgs) Handles SerialPort1.DataReceived
        Try
            ' Leemos la línea completa que envía el Arduino (gracias a println)
            Dim datosRecibidos As String = SerialPort1.ReadLine()

            ' ¡IMPORTANTE! No se puede actualizar la UI directamente desde este
hilo.
            ' Usamos Invoke para llamar a un método que sí lo haga de forma segura.
            Me.Invoke(New DelegadoActualizarUI(AddressOf ActualizarControles),
datosRecibidos)

        Catch ex As Exception
            ' Ignorar errores que pueden ocurrir al cerrar el puerto mientras se
reciben datos
        End Try
    End Sub

    ' Este método es llamado por el delegado y es seguro para actualizar la UI
    Private Sub ActualizarControles(texto As String)
        ' Limpiamos el texto de posibles caracteres no deseados
        texto = texto.Trim()

        ' Actualizamos el TextBox
        txtValorSensor.Text = texto

        ' Intentamos convertir el texto a un número para la ProgressBar
        Dim valorNumerico As Integer
        If Integer.TryParse(texto, valorNumerico) Then
            ' La ProgressBar va de 0 a 100, pero el Arduino envía de 0 a 1023.
            ' Ajustamos el rango de la ProgressBar para que coincida.
            pbValorSensor.Maximum = 1023
            pbValorSensor.Value = valorNumerico
        End If
    End Sub

    ' ... (El resto del código de la Práctica 1 se mantiene aquí) ...
End Class
```

Ejecución:

1. Sube el nuevo código al Arduino.
2. Ejecuta la aplicación de Visual Basic.

3. Conéctate al puerto COM.
4. Gira la perilla del potenciómetro. Verás cómo el valor en el TextBox y la ProgressBar se actualizan en tiempo real.

RESULTADOS ESPERADOS

- Comunicación exitosa entre Arduino y una aplicación externa (Visual Studio, Gambas, Java).
- Visualización y/o control remoto de sensores y actuadores desde la aplicación de escritorio.
- Sincronización de datos y comandos: envío de datos de sensores a la PC y recepción de comandos para controlar actuadores desde la PC.
- Registro y análisis de señales en la aplicación externa (por ejemplo, graficar datos en tiempo real o almacenar registros para análisis posterior

ANÁLISIS DE RESULTADOS

- ¿Fue exitosa la comunicación entre Arduino y la aplicación externa? ¿Qué evidencias tienes de ello?
- ¿Qué dificultades técnicas enfrentaste durante la integración y cómo las resolviste?
- ¿Cómo se reflejaron los datos adquiridos por Arduino en la aplicación externa?
- ¿Qué ventajas aporta la integración de Arduino con otros entornos de programación?
- ¿Qué aplicaciones prácticas identificas para este tipo de integración en el mundo real?

CONCLUSIONES Y REFLEXIONES

ACTIVIDADES COMPLEMENTARIAS

Monitor de Sensor Remoto (Visual Studio / Gambas / JAVA): Enviar lecturas de un sensor de temperatura (DHT11/DHT22) desde Arduino a una aplicación de escritorio desarrollada en Visual Studio (C#), Gambas o JAVA, y mostrar los datos en una interfaz gráfica.
Requerimientos: Aplicación de escritorio que se conecte al puerto serial de Arduino, Arduino envía la temperatura y humedad cada cierto intervalo, Interfaz gráfica que muestre los valores actualizados.
Conceptos Reforzados: Comunicación serial (Serial.print() en Arduino, librerías de puerto serial en el entorno de desarrollo), parsing de datos, diseño de interfaz de usuario.

Control de LED desde Aplicación de Escritorio: Desarrollar una aplicación de escritorio que contenga un botón para encender y apagar un LED conectado a Arduino.
Requerimientos: La aplicación envía un carácter o cadena de texto específica a Arduino para "encender" o "apagar" el LED, Arduino recibe el comando y controla el LED consecuentemente.
Conceptos Reforzados: Comunicación serial bidireccional, envío de comandos, manejo de eventos en la aplicación de escritorio.

Control de Servomotor Bidireccional: Crear una aplicación de escritorio con un control deslizante (slider) o entrada numérica para establecer la posición de un servomotor conectado a Arduino. Arduino

debe responder a los comandos y enviar su posición actual de vuelta a la aplicación. Requerimientos: Comunicación continua y fluida, Manejo de errores en la comunicación (ej. datos corruptos), Feedback visual en la aplicación sobre la posición actual del servomotor. Conceptos Reforzados: Comunicación serial avanzada, sincronización de datos, control en tiempo real.

EVALUACIÓN Y EVIDENCIAS DE APRENDIZAJE	
Criterios de evaluación	comprensión conceptual, capacidad de programación, diseño de circuitos, resolución de problemas, diseño de interfaz, creatividad y presentación.
Rúbricas o listas de cotejo para valorar desempeño	Rúbrica de Reporte de Prácticas
Formatos de reporte de prácticas	Reporte de Practicas

FUENTES DE INFORMACIÓN

1. Aliverti, P. (2016). *El manual de Arduino:* (1 ed.). Marcombo. https://elibro.net/es/ereader/ues/280056

2. Schmidt, D. (2022). *Arduino: curso completo:* (2 ed.). RA-MA Editorial. https://elibro.net/es/ereader/ues/222675

3. Lajara Vizcaíno, J. R. & Pelegrí Sebastià, J. (2014). *Sistemas integrados con Arduino:* (1 ed.). Marcombo. https://elibro.net/es/ereader/ues/280053

4. *Programming | Arduino Documentation*. (n.d.). https://docs.arduino.cc/programming/

5. *Getting Started with Arduino | Arduino Documentation*. (n.d.). https://docs.arduino.cc/learn/starting-guide/getting-started-arduino/

6. Oliva Ramos, R. (2018). *Monitoreo, control y adquisición de datos con Arduino y Visual Basic.net:* (1 ed.). Marcombo. https://elibro.net/es/ereader/ues/280321

NORMAS TÉCNICAS APLICABLES

COBIT, ITIL e ISO 27000

UES
Universidad Estatal de Sonora
La Fuerza del Saber Estimulará mi Espíritu

ANEXOS

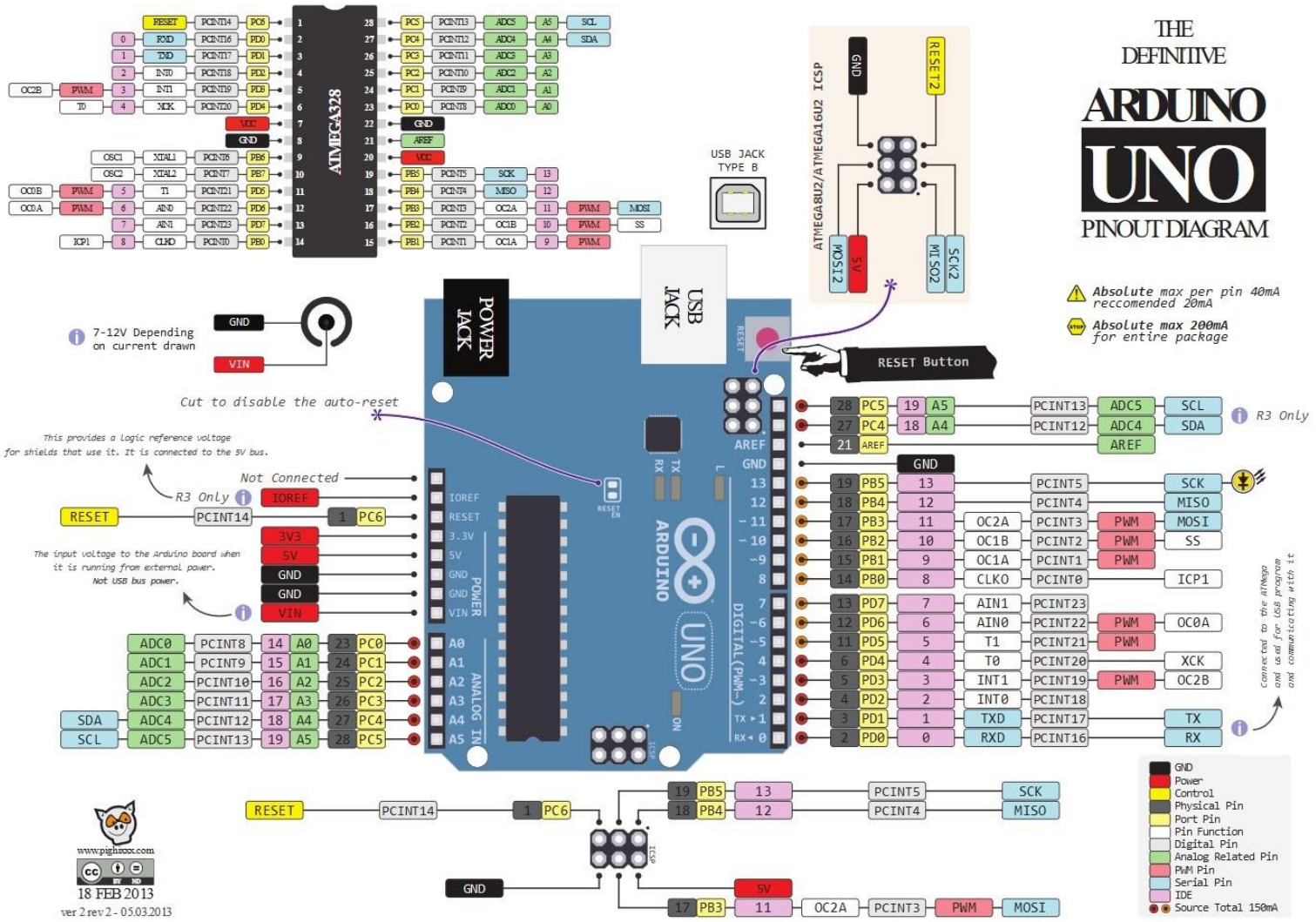
1.- Diagramas, tablas, ejemplos de reportes
2.- Formatos de seguridad y protocolos adicionales
3.- Problemas o ejercicios de apoyo

Descripción de la Rubrica
Rúbrica de evaluación para proyectos basados en la plataforma Arduino. Define criterios para la evaluación de la funcionalidad, código, diseño, etc.
Importancia de la Rubrica
Proporciona una herramienta objetiva y consistente para evaluar proyectos de Arduino, facilitando la retroalimentación y la mejora del aprendizaje.

Criterio	Excelente	Bueno	Regular	Malo	Deficiente
Funcionalidad	El proyecto cumple todas las funciones especificadas de manera eficiente y robusta.	El proyecto cumple la mayoría de las funciones especificadas, con algunas pequeñas deficiencias.	El proyecto cumple algunas de las funciones especificadas, pero con varias deficiencias.	El proyecto cumple pocas funciones especificadas, con muchas deficiencias.	El proyecto no cumple con las funciones especificadas.
Código	El código es limpio, bien documentado, eficiente y fácil de entender. Utiliza buenas prácticas de programación.	El código es legible y funciona correctamente, aunque podría mejorarse la documentación o la eficiencia.	El código funciona, pero es difícil de leer, entender o mantener. Falta documentación.	El código funciona con dificultad y presenta errores significativos. La documentación es inexistente o incompleta.	El código no funciona o es incomprensible.
Diseño del Circuito	El diseño del circuito es eficiente, limpio y bien organizado. Se utilizan componentes adecuados.	El diseño del circuito es funcional, pero podría mejorarse en términos de eficiencia o organización.	El diseño del circuito es funcional, pero presenta deficiencias en organización y/o elección de componentes.	El diseño del circuito presenta errores significativos que afectan su funcionalidad.	El diseño del circuito es incorrecto o no funcional.
Integración de Hardware/Software	La integración entre el hardware y el software es perfecta y eficiente.	La integración entre hardware y software funciona correctamente, pero podría optimizarse.	La integración entre hardware y software presenta algunas dificultades o ineficiencias.	La integración entre hardware y software presenta problemas significativos que afectan la funcionalidad.	La integración entre hardware y software es incorrecta o inexistente.
Innovación	El proyecto presenta una solución innovadora y creativa al problema planteado.	El proyecto presenta una solución original con algunas mejoras sobre soluciones existentes.	El proyecto presenta una solución funcional, pero no especialmente innovadora.	El proyecto presenta una solución poco original y con poca innovación.	El proyecto no presenta ninguna innovación.
Pruebas y Depuración	El proyecto ha sido probado exhaustivamente y se han corregido todos los errores.	El proyecto ha sido probado y la mayoría de los errores se han corregido.	El proyecto ha sido probado parcialmente y presenta algunos errores residuales.	El proyecto presenta numerosos errores sin corregir.	El proyecto no ha sido probado.
Documentación	La documentación es completa, clara y	La documentación es clara y	La documentación es incompleta o	La documentación es mínima o	No se presenta documentación.

Criterio	Excelente	Bueno	Regular	Malo	Deficiente
	concisa. Incluye diagramas de circuitos y explicaciones detalladas.	completa, aunque podría mejorarse en algunos aspectos.	poco clara.	inexistente.	
Presentación	La presentación del proyecto es clara, organizada y profesional.	La presentación del proyecto es comprensible, aunque podría mejorarse en organización o claridad.	La presentación del proyecto es confusa o incompleta.	La presentación del proyecto es difícil de entender.	No se presenta el proyecto.

Diagrama Arduino Uno



Fundamentos programación Arduino

Arduino - Guía Rápida

Variables, Vectores y Datos

Tipos de datos

```
void      vacío
boolean   (0, 1, true, false)
char      (ej. 'a'  -128 a 127)
int       (-32768 a 32767)
long      (-2147483648 a 2147483647)
unsigned char  (0 a 255)
byte           (0 a 255)
unsigned int   (0 a 65535)
word           (0 a 65535)
unsigned long  (0 a 4294967295)
float     (-3.4028e+38 a 3.4028e+38)
double    (igual que los flotantes)
```

Constantes

```
HIGH | LOW
INPUT | OUTPUT
true | false
143        //Decimal
0173       //Octal (comenzando en 0)
0b11011111 //Binario
0x7B       //Hex (hexadecimal)
7U         //forzar unsigned
10L        //forzar long
15UL       //forzar long unsigned
10.0       //forzar floating point
2.4e5      //240000
```

Calificadores

```
static   //persiste entre llamadas
volatile //usa la RAM
const    //sólo lectura
PROGMEM  //usar la flash
```

Vectores y matrices

```
int myInts[6]; //vector de 6 enteros
int myPins[]={2, 4, 8, 3, 6};
int mySensVals[6]={2, 4, -8, 3, 2};
myInts[0]=42;  //asigna al primero
               //en el índice
myInts[6]=12;  //ERROR! El índice va
               //de 0 a 5
```

Punteros

```
&  (referencia: obtener puntero)
*  (valor: seguir puntero)
```

Cadenas

```
char S1[8] =
 {'A','r','d','u','i','n','o'};
 //cadena sin terminación
 //puede producir error
char S2[8] =
 {'A','r','d','u','i','n','o','\0'};
 //incluye terminación nula \0
char S3[]="arduino";
char S4[8]="arduino";
```

Operadores

Operadores generales

```
=  (operador de asignación)
+  (adición)     -  (sustracción)
*  (multiplicación)
/  (división)    %  (módulo)
== (igual a)     != (desigual a)
<  (menor que)   >  (mayor que)
<= (igual o menor que)
>= (mayor o igual que)
&& (y)    || (ó)    ! (negación)
```

Operadores compuestos

```
++ (incremento)
-- (decremento)
+= (suma compuesta)
-= (resta compuesta)
*= (multiplicación compuesta)
/= (división compuesta)
&= (AND binario compuesto)
|= (OR binario compuesto)
```

Operadores a nivel de bit

```
& (AND binario)    | (OR binario)
^ (XOR binario)    ~ (NOT binario)
<< (desplazamiento a la izquierda)
>> (desplazamiento a la derecha)
```

Estructura y flujo

Estructura básica del programa

```
void setup() {
  // Corre una vez cuando el
  // programa inicia
}
void loop() {
  // Se ejecuta repetidamente
```

Estructuras de control

```
if (x < 5) { ... } else { ... }
while (x < 5) { ... }
do { ... } while ( x < 5);
for (int i = 0; i < 10; i++) { ... }
break;  //sale del bucle inmediatamente
continue; //va a la siguiente iteración
switch (miVariable) {
   case 1:
     ...
     break;
   case 2:
     ...
     break;
   default:
     ...
}
return x; // o "return;" para vacíos
```

Funciones incluídas

E/S Digital

```
pinMode(pin,[INPUT, OUTPUT])
digitalWrite(pin, valor)
int digitalread(pin)
  //Escribe HIGH en entradas para
  //usar los pull-ups
```

E/S Analógicas

```
analogReference([DEFAULT,
    INTERNAL, EXTERNAL])
int analogRead(pin)
analogWrite(pin, valor)  //PWM
```

Advanced I/O

```
tone(pin, freqhz)
tone(pin, freqhz, duracion_ms)
noTone(pin)
shiftOut(pinDatos, pinReloj,
  [MSBFIRST,LSBFIRST], valor)
unsigned long pulseIn(pin,
  [HIGH,LOW])
```

Tiempo

```
unsigned long millis()
  //desbordamiento en 50 dias
unsigned long micros()
  //desbordamiento en 70 minutos
delay(ms)
delayMicroseconds(us)
```

Matemáticas

```
min(x, y)   max(x, y)   abs(x)
sin(rad)    cos(rad)    tan(rad)
sqrt(x)     pow(base, exponente)
constrain(x, valMin, valMax)
map(val, deBAJO, deALTO,
aBAJO,aAlto)
```

Números aleatorios

```
randomSeed(semilla) //long ó int
long random(max)
long random(min, max)
```

Bits y Bytes

```
lowByte(x)    highByte(x)
bitRead(x, bitn)
bitWrite(x, bitn, bit)
bitSet(x, bitn)
bitClear(x, bitn)
bit(bitn)  // bitn: 0=LSB 7=MSB
```

Conversiones

```
char()        byte()
int()         word()
long()        float()
```

Interrupciones Externas

```
attachInterrupt(interrup, func,
 [LOW, CHANGE, RISING, FALLING])
detachInterrupt(interrupción)
interrupts()
noInterrupts()
```

Librerías

Serie

```
begin([300, 1200, 2400, 4800,
      9600, 14400, 19200, 28800,
      38400, 57600, 115200])
     //Puede ser cualquier número
end()
int available()
byte read()
byte peek()
flush()
print(misDatos)
println(misDatos)
write(misBytes)
```

EEPROM (#include <EEPROM.h>)

```
byte read(dirInterna)
write(dirInterna, miByte)
```

Servo (#include <Servo.h>)

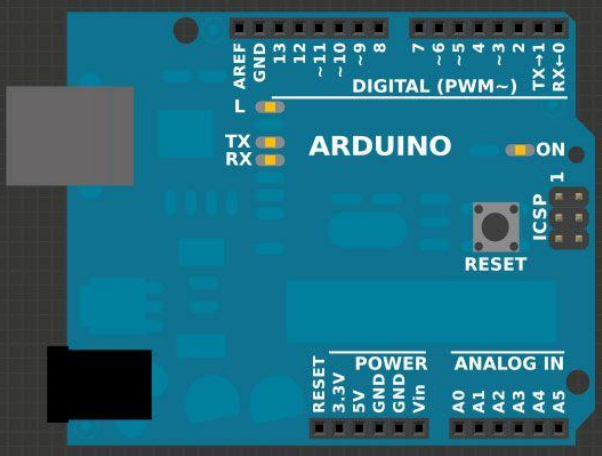
```
attach(pin, [min_uS, max_uS])
write(ángulo) // 0, 180
writeMicroseconds(uS)
    //1000-2000; 1500 es en medio
read()       //0 - 180
attached()   //regresa boleano
detach()
```

SoftwareSerial(RxPin, TxPin)

```
  (#include <softwareSerial.h>)
begin(long velocidad) //hasta 9600
char read()     //espera los datos
print(misDatos)
println(misDatos)
```

Wire (#include <Wire.h>) //para I²C

```
begin()       //se une a maestro
begin(addr)   //se une a esclavo @dir
requestFrom(dirección, cuenta)
beginTransmission(dir)  // Paso 1
send(miByte)            // Paso 2
send(char * miCadena)
send(byte * datos, tamaño)
endTransmission()       // Paso 3
byte available()   // Num de bytes
byte receive() //Regresa el sig byte
onReceive(manejador)
onRequest(manejador)
```



aladuino
electrónica especializada

www.aladuino.com.mx

Basada principalmente de la Referencia de Lenguaje Arduino: http://arduino.cc/reference/es/

Visual Basic Quick Reference

Operators

+, -, *, /	Addition, subtraction, multiplication, division
****	Integer Division
Mod	Remainder
^	Exponent
&	String concatenation
=, >, <, >=, <=	Comparison
NOT, AND, OR	Boolean operators

Data Types

Variant, Integer (%), Long(&), Single (!), Double (#), Byte, Boolean, Date, Currency (@), String ($)

CBool(*expr*), **CByte**(*expr*), **CCur**(*expr*), **CDate**(*expr*), **CDbl**(*expr*), **CDec**(*expr*), **CInt**(*expr*), **CLng**(*expr*), **CSng**(*expr*), **CStr**(*expr*), **CVar**(*expr*)

[**Public** | **Private**] **Const** *constname* [**As** *type*] = expression

Dim [**WithEvents**] *varname* [([*subscripts*])] [**As** [**New**] *type*]

ReDim [**Preserve**] *varname* (*subscripts*) [**As** *type*]

[**Public** | **Private**] **Enum** *name*

[**Private** | **Public**] **Type** *varname*
elementname [([*subscripts*])] **As** *type*
[elementname [([*subscripts*])] **As** *type*]
. . .
End Type

Set *objectvar* = {[**New**] *objectexpression* | **Nothing**}

Static *varname*[([*subscripts*])] [**As** [**New**] *type*]

Math Functions

Abs(*num*), **Atn**(*num*), **Cos**(*num*), **Log**(*num*), **Rnd**[(*num*)], **Randomize**, **Sin**(*num*), **Sqr**(*num*), **Tan**(*num*)

FV(*rate, nper, pmt*[, *pv*[, *type*]])
NPV(*rate, values()*)
PV(*rate, nper, pmt*[, *fv*[, *type*]])
Pmt(*rate, nper, pv*[, *fv*[, *type*]])
PPmt(*rate, per, nper, pv*[, *fv*[, *type*]])

String Functions

Left(*string, length*), **Right**(*string, length*), **Mid**(*string, start*[, *length*])
UCase(*string*), **LCase**(*string*), **Len**(*string*)
LTrim(*string*), **RTrim**(*string*), **Trim**(*string*)
Asc(*string*), **Val**(*string*), **Oct**(*number*), **Hex**(*number*)

Split(*expression*[, *delimiter*[, *count*[, *compare*]]])
Join(*list*[, *delimiter*])

Replace(*expression, find, replacewith*[, *start*[, *count*[, *compare*]]])
StrComp(*string1, string2*[, *compare*])
Filter(*InputStrings, Value*[, *Include*[, *Compare*]])
StrReverse(*string1*)

InStr([*start*,]*string1, string2*[, *compare*])
InstrRev(*string1, string2*[, *start*[, *compare*]])

Program Flow

For *counter* = start **To** end [**Step** step]
[statements]
[**Exit For**]
[statements]
Next [*counter*]

For Each *element* **In** *group*
[statements]
[**Exit For**]
[statements]
Next [*element*]

If *condition* **Then** [statements] [**Else** elsestatements]

Or, you can use the block form syntax:
If *condition* **Then**
[statements]
[**ElseIf** *condition-n* **Then**
[elseifstatements]
[Else
[elsestatements]]
End If

Do [{**While** | **Until**} *condition*]
[statements]
[**Exit Do**]
[statements]
Loop

Do
[statements]
[**Exit Do**]
[statements]
Loop [{**While** | **Until**} *condition*]

Select Case *testexpression*
[**Case** *expressionlist-n*
[statements-n]] . . .
[**Case Else**
[elsestatements]]
End Select

While *condition*
[statements]
Wend

With *object*
[statements]
End With

On Error GoTo *line*
On Error Resume [0|`line`**|Next]**
On Error GoTo 0

File Operation

Open *pathname* **For** *mode* [**Access** *access*] [lock] **As** [**#**] *filenumber* [**Len**=*reclength*]

Input #*filenumber, varlist*
Print #*filenumber,* [*outputlist*]

Line Input #*filenumber, varname*
Write #*filenumber,* [*outputlist*]

Get [**#**]*filenumber,* [*recnumber*], *varname*
Put [**#**]*filenumber,* [*recnumber*], *varname*

Loc(*filenumber*)
Seek [**#**]*filenumber, position*

Eof(*filenumber*)
Lof(*filenumber*)

Reset
Close [*filenumberlist*]

Lock [**#**]*filenumber*[, *recordrange*]
Unlock [**#**]*filenumber*[, *recordrange*]

Function and Procedure

[**Public** | **Private** | **Friend**] [**Static**] **Function** *name*[(*arglist*)] [**As** *type*]
[*statements*]
[**Exit Function**]
[*statements*]
End Function

[**Private** | **Public** | **Friend**] [**Static**] **Sub** *name* [(*arglist*)]
[statements]
[**Exit Sub**]
[statements]
End Sub

[**Public** | **Private**] **Declare Sub** *name* **Lib** "*libname*" [**Alias** "*aliasname*"] [([*arglist*])]

[**Public** | **Private**] **Declare Function** *name* **Lib** "*libname*" [**Alias** "*aliasname*"] [([*arglist*])] [**As** *type*]

[**Call**] *name* [[ByVal] *argumentlist*]

Property Procedure

[**Public** | **Private** | **Friend**] [**Static**] **Property Get** *name* [(*arglist*)] [**As** *type*]
[statements]
[**Exit Property**]
[statements]
End Property

[Public | Private | Friend] [Static] Property Let *name* ([*arglist*,] *value*)
[statements]
[Exit Property]
[statements]
End Property

[Public | Private | Friend] [Static] Property Set *name* ([*arglist*,] *reference*)
[statements]
[Exit Property]
[statements]
End Property

System and Miscellaneous

AppActivate *title*[, *wait*]	Make a windows focus
SendKeys *string*[, *wait*]	Send key stroks to current app
Shell(*pathname*[, *windowstyle*])	Run an external program
Timer	seconds elapsed since midnight
Command	Command line
Beep	Beep on internal speaker
Option Base {0 | 1}	Array base
Option Explicit	Force explicit declaration
ChDir *path*	Change current directory
MkDir *path*	Make a directory
RmDir *path*	Remove a directory
ChDrive *drive*	Change current drive
Kill *filename*	Delete a file
FileCopy *source*, *destination*	Copy a file
Name *old* As *new*	Rename a file
FileLen(*pathname*)	File size
FileDateTime(*pathname*)	File creation date
Date	Get/Set system date
Time	Get/Set system time
Environ({*envstring* | *number*})	Environment string
Error[(*errornumber*)]	Error description string
' comment	Rem comment
space_	Continue line

InputBox(*prompt*[, *title*] [, *default*] [, *xpos*] [, *ypos*] [, *helpfile*, *context*])
MsgBox(*prompt*[, *buttons*] [, *title*] [, *helpfile*, *context*])

GetSetting(*appname*, *section*, *key*[, *default*])
SaveSetting *appname*, *section*, *key*, *setting*
GetAllSettings(*appname*, *section*)
DeleteSetting *appname*, *section*[, *key*]

CreateObject(*class*, [*servername*])
GetObject([*pathname*] [, *class*])
Load object
Unload *object*

LoadPicture([*filename*], [*size*], [*colordepth*],[*x,y*])
SavePicture *picture*, *stringexpression*

Objects

App

Properties	Comments, CompanyName, EXEName, FileDescription, HelpFile, LegalCopyright, LegalTrademarks, LogMode, LogPath, Major, Minor, NonModalAllowed, OLERequestPendingMsgText, OLERequestPendingMsgTitle, OLERequestPendingTimeout, OLEServerBusyMsgText, OLEServerBusyMsgTitle, OLEServerBusyRaiseError, OLEServerBusyTimeout, Path, PrevInstance, ProductName, RetainedProject, Revision, StartMode, TaskVisible, ThreadID, Title, UnattendedApp, hInstance
Methods	LogEvent, StartLogging

Printer

Properties	ColorMode, Copies, Count, CurrentX, CurrentY, DeviceName, DrawMode, DrawStyle, DrawWidth, DriverName, Duplex, FillColor, FillStyle, Font, FontBold, FontItalic, FontStrikethru, FontUnderline, FontCount, FontName, FontSize, FontTransparent, Fonts, Height, Width, Orientation, Page, PaperBin, PaperSize, Port, PrintQuality, RightToLeft, ScaleHeight, ScaleWidth, ScaleLeft, ScaleTop, ScaleMode, TrackDefault, TwipsPerPixelX, TwipsPerPixelY, Zoom, hDC
Methods	Circle, EndDoc, KillDoc, Line, NewPage, PSet, PaintPicture, Scale, ScaleX, ScaleY, TextHeight, TextWidth

Timer

Properties	Enabled, Index, Interval, Left, Top, Name, Parent, Tag

Standard Controls

CheckBox

Properties	Alignment, Appearance, BackColor, ForeColor, Caption, Container, DataChanged, DataField, DataFormat, DataMember, DisabledPicture, DownPicture, DragIcon, DragMode, Enabled, Font, FontBold, FontItalic, FontStrikethru, FontUnderline, FontName, FontSize, Height, Width, HelpContextID, Index, Left, Top, MaskColor, MouseIcon, MousePointer, Name, OLEDropMode, Parent, Picture, RightToLeft, RightToLeft, Style, TabIndex, TabStop, Tag, ToolTipText, UseMaskColor, Value, Visible, WhatsThisHelpID, hWnd
Methods	Drag, Move, OLEDrag, Refresh, SetFocus, ShowWhatsThis, ZOrder
Events	Click, DragDrop, DragOver, GotFocus, KeyDown, KeyUp, KeyPress, LostFocus, MouseDown, MouseUp, MouseMove, OLECompleteDrag, OLEDragDrop, OLEDragOver, OLEGiveFeedback, OLESetData, OLEStartDrag, Validate

ComboBox

Properties	Appearance, BackColor, ForeColor, Container, DataChanged, DataField, DataFormat, DataMember, DragIcon, DragMode, Enabled, Font, FontBold, FontItalic, FontStrikethru, FontUnderline, FontName, FontSize, Height, Width, HelpContextID, Index, IntegralHeight, ItemData, Left, Top, List, ListCount, ListIndex, Locked, MouseIcon, MousePointer, Name, NewIndex, OLEDragMode, OLEDropMode, Parent, RightToLeft, RightToLeft, SelLength, SelStart, SelText, SelLength, SelStart, SelText, Sorted, Style, TabIndex, TabStop, Tag, Text, ToolTipText, TopIndex, Visible, WhatsThisHelpID, hWnd
Methods	AddItem, Clear, Drag, Move, OLEDrag, Refresh, RemoveItem, SetFocus, ShowWhatsThis, ZOrder
Events	Change, Click, DblClick, DragDrop, DragOver, DropDown, GotFocus, KeyDown, KeyUp, KeyPress, LostFocus, OLECompleteDrag, OLEDragDrop, OLEDragOver, OLEGiveFeedback, OLESetData, OLEStartDrag, Scroll, Validate

CommandButton

Properties	Appearance, BackColor, ForeColor, Cancel, Caption, Container, Default, DisabledPicture, DownPicture, DragIcon, DragMode, Enabled, Font, FontBold, FontItalic, FontStrikethru, FontUnderline, FontName, FontSize, Height, Width, HelpContextID, Index, Left, Top, MaskColor, MouseIcon, MousePointer, Name, OLEDropMode, Parent, Picture, RightToLeft, Style, TabIndex, TabStop, Tag, ToolTipText, UseMaskColor, Value, Visible, WhatsThisHelpID, hWnd
Methods	Drag, Move, OLEDrag, Refresh, ShowWhatsThis, UpdateControls, UpdateRecord, ZOrder
Events	Click, DragDrop, DragOver, GotFocus, KeyDown, KeyUp, KeyPress, LostFocus, MouseDown, MouseUp, MouseMove, OLECompleteDrag, OLEDragDrop, OLEDragOver, OLEGiveFeedback, OLESetData, OLEStartDra

Data

Properties	Align, Appearance, BOFAction, EOFAction, BackColor, ForeColor, Caption, Connect, Database, DatabaseName, DefaultCursorType,

	DefaultType, DragIcon, DragMode, EditMode, Enabled, Exclusive, Font, FontBold, FontItalic, FontStrikethru, FontUnderline, FontName, FontSize, Height, Width, Index, Left, Top, MouseIcon, MousePointer, Name, OLEDropMode, Options, Parent, ReadOnly, RecordSource, Recordset, RecordsetType, RightToLeft, RightToLeft, Tag, ToolTipText, Visible, WhatsThisHelpID
Methods	Drag, Move, OLEDrag, Refresh, ShowWhatsThis, UpdateControls, UpdateRecord, ZOrder
Events	DragDrop, DragOver, Error, MouseDown, MouseUp, MouseMove, OLECompleteDrag, OLEDragDrop, OLEDragOver, OLEGiveFeedback, OLESetData, OLEStartDrag, Reposition, Resize, Validate

DirListBox

Properties	Appearance, BackColor, ForeColor, Container, DragIcon, DragMode, Enabled, Font, FontBold, FontItalic, FontStrikethru, FontUnderline, FontName, FontSize, Height, Width, HelpContextID, Index, Left, Top, List, ListCount, ListIndex, MouseIcon, MousePointer, Name, OLEDragMode, OLEDropMode, Parent, Path, TabIndex, TabStop, Tag, ToolTipText, TopIndex, Visible, WhatsThisHelpID, hWnd
Methods	Drag, Move, OLEDrag, Refresh, SetFocus, ShowWhatsThis, ZOrder
Events	Change, Click, DragDrop, DragOver, GotFocus, KeyDown, KeyUp, KeyPress, LostFocus, MouseDown, MouseUp, MouseMove, OLECompleteDrag, OLEDragDrop, OLEDragOver, OLEGiveFeedback, OLESetData, OLEStartDrag, Scroll, Validate

DriveListBox

Properties	Appearance, BackColor, ForeColor, Container, DragIcon, DragMode, Drive, Enabled, Font, FontBold, FontItalic, FontStrikethru, FontUnderline, FontName, FontSize, Height, Width, HelpContextID, Index, Left, Top, List, ListCount, ListIndex, MouseIcon, MousePointer, Name, OLEDropMode, Parent, TabIndex, TabStop, Tag, ToolTipText, TopIndex, Visible, WhatsThisHelpID, hWnd
Methods	Drag, Move, OLEDrag, Refresh, SetFocus, ShowWhatsThis, ZOrder
Events	Change, DragDrop, DragOver, GotFocus, KeyDown, KeyUp, KeyPress, LostFocus, OLECompleteDrag, OLEDragDrop, OLEDragOver, OLEGiveFeedback, OLESetData, OLEStartDrag, Scroll, Validate

FileListBox

Properties	Appearance, Archive, Hidden, Normal, System, BackColor, ForeColor, Container, DragIcon, DragMode, Enabled, FileName, Font, FontBold, FontItalic, FontStrikethru, FontUnderline, FontName, FontSize, Height, Width, HelpContextID, Index, Left, Top, List, ListCount, ListIndex, Locked, MouseIcon, MousePointer, MultiSelect, Name, OLEDragMode, OLEDropMode, Parent, Path, Pattern, ReadOnly, Selected, TabIndex, TabStop, Tag, ToolTipText, TopIndex, Visible, WhatsThisHelpID, hWnd
Methods	Drag, Move, OLEDrag, Refresh, SetFocus, ShowWhatsThis, ZOrder
Events	Click, DblClick, DragDrop, DragOver, GotFocus, KeyDown, KeyUp, KeyPress, LostFocus, MouseDown, MouseUp, MouseMove, OLECompleteDrag, OLEDragDrop, OLEDragOver, OLEGiveFeedback, OLESetData, OLEStartDrag, PathChange, PatternChange, Scroll, Validate

Frame

Properties	Appearance, BackColor, ForeColor, BorderStyle, Caption, ClipControls, Container, DragIcon, DragMode, Enabled, Font, FontBold, FontItalic, FontStrikethru, FontUnderline, FontName, FontSize, Height, Width, HelpContextID, Index, Left, Top, MouseIcon, MousePointer, Name, OLEDropMode, Parent, RightToLeft, TabIndex, Tag, ToolTipText, Visible, WhatsThisHelpID, hWnd
Methods	Drag, Move, OLEDrag, Refresh, ShowWhatsThis, ZOrder
Events	Click, DblClick, DragDrop, DragOver, MouseDown, MouseUp, MouseMove, OLECompleteDrag, OLEDragDrop, OLEDragOver, OLEGiveFeedback, OLESetData, OLEStartDrag

HScrollBar/VScrollBar

Properties	Container, DragIcon, DragMode, Enabled, Height, Width, HelpContextID, Index, LargeChange, SmallChange, Left, Top, Max, Min, MouseIcon, MousePointer, Name, Parent, RightToLeft, TabIndex, TabStop, Tag, Value, Visible, WhatsThisHelpID, hWnd
Methods	Drag, Move, Refresh, SetFocus, ShowWhatsThis, ZOrder
Events	Change, DragDrop, DragOver, GotFocus, KeyDown, KeyUp, KeyPress, LostFocus, Scroll, Validate

Image

Properties	Appearance, BorderStyle, Container, DataChanged, DataField, DataFormat, DataMember, DataSource, DragIcon, DragMode, Enabled, Height, Width, Index, Left, Top, MouseIcon, MousePointer, Name, OLEDragMode, OLEDropMode, Parent, Picture, Stretch, Tag, ToolTipText, Visible, WhatsThisHelpID
Methods	Drag, Move, OLEDrag, Refresh, ShowWhatsThis, ZOrder
Events	Click, DblClick, DragDrop, DragOver, MouseDown, MouseUp, MouseMove, OLECompleteDrag, OLEDragDrop, OLEDragOver, OLEGiveFeedback, OLESetData, OLEStartDrag

Label

Properties	Alignment, Appearance, AutoSize, BackColor, ForeColor, BackStyle, BorderStyle, Caption, Container, DataChanged, DataField, DataFormat, DataMember, DataSource, DragIcon, DragMode, Enabled, Font, FontBold, FontItalic, FontStrikethru, FontUnderline, FontName, FontSize, Height, Width, Index, Left, Top, LinkItem, LinkMode, LinkTimeout, LinkTopic, MouseIcon, MousePointer, Name, OLEDrag Method, OLEDropMode, Parent, RightToLeft, TabIndex, Tag, ToolTipText, UseMnemonic, Visible, WhatsThisHelpID, WordWrap
Methods	Drag, LinkExecute, LinkPoke, LinkRequest, LinkSend, Move, OLEDrag, Refresh, ShowWhatsThis, ZOrder
Events	Change, Click, DblClick, DragDrop, DragOver, LinkClose, LinkError, LinkNotify, LinkOpen, MouseDown, MouseUp, MouseMove, OLECompleteDrag, OLEDragDrop, OLEDragOver, OLEGiveFeedback, OLESetData, OLEStartDrag

Line

Properties	BorderColor, BorderStyle, BorderWidth, Container, DrawMode, Index, Name, Parent, Tag, Visible, X1, Y1, X2, Y2
Methods	Refresh, ZOrder

ListBox

Properties	Appearance, BackColor, ForeColor, Columns, Container, DataChanged, DataField, DataFormat,

	DataMember, DataSource, DragIcon, DragMode, Enabled, Font, FontBold, FontItalic, FontStrikethru, FontUnderline, FontName, FontSize, Height, Width, HelpContextID, Index, IntegralHeight, ItemData, Left, Top, List, ListCount, ListIndex, MouseIcon, MousePointer, MultiSelect, Name, NewIndex, OLEDragMode, OLEDropMode, Parent, RightToLeft, SelCount, Selected, Sorted, Style, TabIndex, TabStop, Tag, Text, ToolTipText, TopIndex, Visible, WhatsThisHelpID, hWnd
Methods	AddItem, Clear, Drag, Move, OLEDrag, Refresh, RemoveItem, SetFocus, ShowWhatsThis, ZOrder
Events	Click, DblClick, DragDrop, DragOver, GotFocus, ItemCheck, KeyDown, KeyUp, KeyPress, LostFocus, MouseDown, MouseUp, MouseMove, OLECompleteDrag, OLEDragDrop, OLEDragOver, OLEGiveFeedback, OLESetData, OLEStartDrag, Scroll, Validate

Menu

Properties	Caption, Checked, Enabled, HelpContextID, Index, Name, NegotiatePosition, Parent, Shortcut, Tag, Visible, WindowList
Events	Click

OptionButton

Properties	Alignment, Appearance, BackColor, ForeColor, Caption, Container, DataFormat, DisabledPi`cture, DownPicture, DragIcon, DragMode, Enabled, Font, FontBold, FontItalic, FontStrikethru, FontUnderline, FontName, FontSize, Height, Width, HelpContextID, Index, Left, Top, MaskColor, MouseIcon, MousePointer, Name, OLEDropMode, Parent, Picture, RightToLeft, Style, TabIndex, TabStop, Tag, ToolTipText, UseMaskColor, Value, Visible, WhatsThisHelpID, hWnd
Methods	Drag, Move, OLEDrag, Refresh, SetFocus, ShowWhatsThis, ZOrder
Events	Click, DblClick, DragDrop, DragOver, GotFocus, KeyDown, KeyUp, KeyPress, LostFocus, MouseDown, MouseUp, MouseMove, OLECompleteDrag, OLEDragDrop, OLEDragOver, OLEGiveFeedback, OLESetData, OLEStartDrag, Validate

PictureBox

Properties	Align, Appearance, AutoRedraw, AutoSize, BackColor, ForeColor, BorderStyle, ClipControls, Container, CurrentX, CurrentY, DataChanged, DataField, DataFormat, DataMember, DataSource, DragIcon, DragMode, DrawMode, DrawStyle, DrawWidth, Enabled, FillColor, FillStyle, Font, FontBold, FontItalic, FontStrikethru, FontUnderline, FontName, FontSize, FontTransparent, Height, Width, HelpContextID, Image, Index, Left, Top, LinkItem, LinkMode, LinkTimeout, LinkTopic, MouseIcon, MousePointer, Name, Negotiate, OLEDragMode, OLEDropMode, Parent, Picture, RightToLeft, ScaleHeight, ScaleWidth, ScaleLeft, ScaleTop, ScaleMode, TabIndex, TabStop, Tag, ToolTipText, Visible, WhatsThisHelpID, hDC, hWnd
Methods	Circle, Cls, Drag, Line, LinkExecute, LinkPoke, LinkRequest, LinkSend, Move, OLEDrag, PSet, PaintPicture, Point, Refresh, Scale, ScaleX, ScaleY, SetFocus, ShowWhatsThis, TextHeight, TextWidth, ZOrder
Events	Change, Click, DblClick, DragDrop, DragOver, GotFocus, KeyDown, KeyUp, KeyPress, LinkClose, LinkError, LinkNotify, LinkOpen, LostFocus, MouseDown, MouseUp, MouseMove, OLECompleteDrag, OLEDragDrop, OLEDragOver, OLEGiveFeedback, OLESetData, OLEStartDrag, Paint, Resize, Validate

Shape

Properties	BackColor, ForeColor, BackStyle, BorderColor, BorderStyle, BorderWidth, Container, DrawMode, FillColor, FillStyle, Height, Width, Index, Left, Top, Name, Parent, Shape, Tag, Visible
Methods	Move, Refresh, Zorder

TextBox

Properties	Alignment, Appearance, BackColor, ForeColor, BorderStyle, Container, DataChanged, DataField, DataFormat, DataMember, DataSource, DragIcon, DragMode, Enabled, Font, FontBold, FontItalic, FontStrikethru, FontUnderline, FontName, FontSize, Height, Width, HelpContextID, HideSelection, Index, Left, Top, LinkItem, LinkMode, LinkTimeout, LinkTopic, Locked, MaxLength, MouseIcon, MousePointer, MultiLine, Name, OLEDragMode, OLEDropMode, Parent, PasswordChar, RightToLeft, ScrollBars, SelLength, SelStart, SelText, TabIndex, TabStop, Tag, Text, ToolTipText, Visible, WhatsThisHelpID, hWnd
Methods	Drag, LinkExecute, LinkPoke, LinkRequest, LinkSend, Move, OLEDrag, Refresh, SetFocus, ShowWhatsThis, ZOrder
Events	Change, Click, DblClick, DragDrop, DragOver, GotFocus, KeyDown, KeyUp, KeyPress, LinkClose, LinkError, LinkNotify, LinkOpen, LostFocus, MouseDown, MouseUp, MouseMove, OLECompleteDrag, OLEDragDrop, OLEDragOver, OLEGiveFeedback, OLESetData, OLEStartDrag, Validate

OLE Container

Properties	Action, AppIsRunning, Appearance, AutoActivate, AutoVerbMenu, BackColor, ForeColor, BackStyle, BorderStyle, Class, Container, Data, DataChanged, DataField, DataText, DisplayType, DragIcon, DragMode, Enabled, FileNumber, Format, Height, Width, HelpContextID, HostName, Index, Left, Top, MiscFlags, MouseIcon, MousePointer, Name, OLEDropAllowed, OLEType, OLETypeAllowed, Object, ObjectAcceptFormats, ObjectAcceptFormatsCount, ObjectGetFormats, ObjectGetFormatsCount, ObjectVerbFlags, ObjectVerbs, ObjectVerbsCount, Parent, PasteOK, Picture, SizeMode, SourceDoc, SourceItem, TabIndex, TabStop, Tag, UpdateOptions, Verb, Visible, WhatsThisHelpID, hWnd, lpOleObject
Methods	Close, Copy, CreateEmbed, CreateLink, Delete, DoVerb, Drag, FetchVerbs, InsertObjDlg, Move, Paste, PasteSpecialDlg, ReadFromFile, Refresh, SaveToFile, SaveToOle1File, SetFocus, ShowWhatsThis, Update, ZOrder
Events	Click, DblClick, DragDrop, DragOver, GotFocus, KeyDown, KeyUp, KeyPress, LostFocus, MouseDown, MouseUp, MouseMove, ObjectMove, Resize, Updated

Common Dialog

Properties	Action, CancelError, Color, Copies, DefaultExt, DialogTitle, FileName, FileTitle, Filter, FilterIndex, Flags, FontBold, FontItalic, FontStrikethru, FontName, FontSize, FromPage, ToPage, hDC, HelpCommand, HelpContext, HelpFile, HelpKey, Index, InitDir, Left, Top, Max, Min, MaxFileSize, Name, Object, Orientation, Parent, PrinterDefault
Methods	AboutBox, ShowColor, ShowFont, ShowHelp, ShowOpen, ShowPrinter, ShowSave


UES
Universidad Estatal de Sonora
La Fuerza del Saber Estimulará mi Espíritu