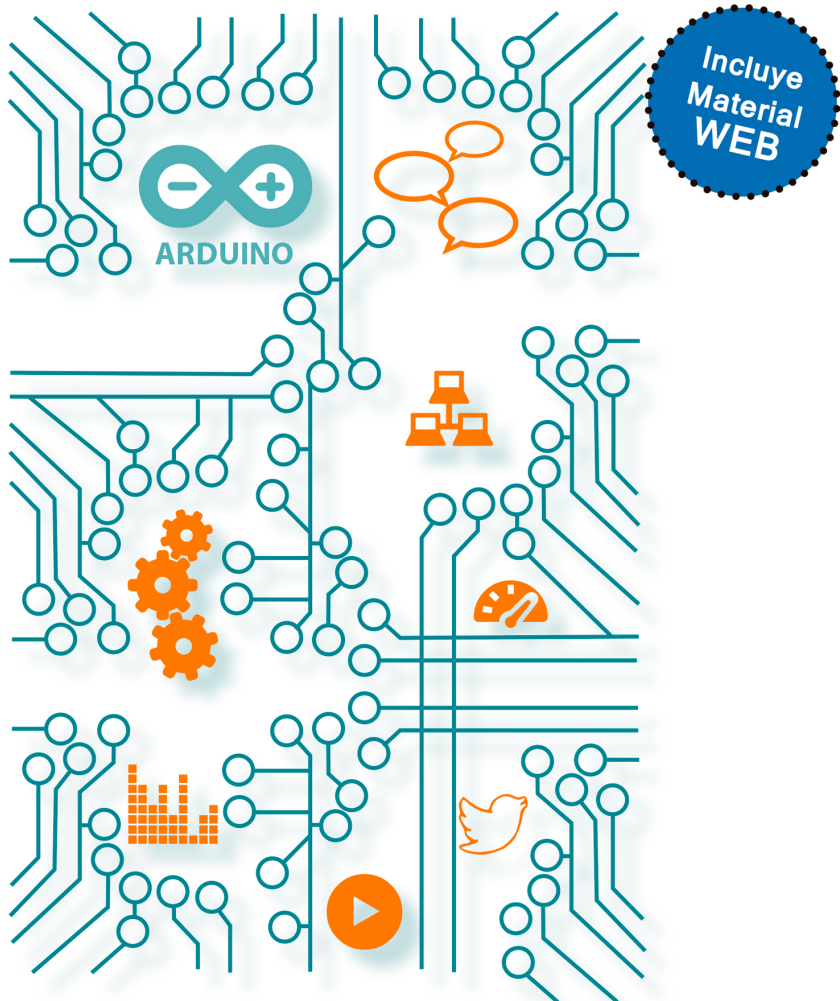


MONITOREO, CONTROL Y ADQUISICIÓN DE DATOS CON ARDUINO Y VISUAL BASIC .NET

Rubén Oliva Ramos



Monitoreo, control y adquisición de datos con Arduino y Visual Basic .NET

Rubén Oliva Ramos

Acceda a www.marcombo.info
para descargar gratis
el contenido adicional
complemento imprescindible de este libro

Código: ARDUINO2

Monitoreo, control y adquisición de datos con Arduino y Visual Basic .NET

Rubén Oliva Ramos



Monitoreo, control y adquisición de datos con arduino y Visual Basic .NET
Rubén Oliva Ramos

Derechos reservados © Alfaomega Grupo Editor, S.A. de C.V., México
Primera edición: febrero 2017
ISBN: 978-607-622-757-2

Primera edición: MARCOMBO, S.A. 2018

© 2018 MARCOMBO, S.A.
www.marcombo.com

«Cualquier forma de reproducción, distribución, comunicación pública o transformación de esta obra sólo puede ser realizada con la autorización de sus titulares, salvo excepción prevista por la ley. Diríjase a CEDRO (Centro Español de Derechos Reprográficos, www.cedro.org) si necesita fotocopiar o escanear algún fragmento de esta obra».

ISBN: 978-84-267-2567-7
D.L.: B-20651-2017

Impreso en Prodigitalk
Printed in Spain

Acerca del autor

Rubén Oliva Ramos



Especialista en desarrollo de software para Arduino y Raspberry Pi con .NET; así como en plataformas de uso libre, desarrollo de aplicaciones para control y monitoreo de procesos a distancia, aplicaciones móviles para monitoreo y control, Sistemas SCADA, tecnologías Android, iOS, Windows Phone, Sitios web y Servicios web aplicados a las plataformas de desarrollo de Arduino y Raspberry Pi con diferentes Frameworks y aplicaciones para la nube (cloud services).

- Ingeniero en Sistemas Computacionales por el Instituto Tecnológico de León.
- Maestro en Ingeniería de Sistemas Electrónicos y Computacionales por la Universidad de la Salle Bajío en León, Guanajuato.
- Especialista en teleinformática y redes por la Universidad de la Salle Bajío en León, Guanajuato.
- Docente a nivel bachillerato en el Centro de Bachillerato Tecnológico Industrial y de Servicios No. 225 (CBTis 225) en León, Guanajuato, en la especialidad de Mecatrónica y Programación, dando cursos de electrónica, automatización, microcontroladores, robótica y control, desarrollo de aplicaciones con Android, bases de datos y plataformas web para monitoreo y control; lo cual ha llevado a cabo desde el año 2011 hasta la actualidad.
- Desde el 2008 hasta la actualidad ha sido docente en la Universidad de La Salle Bajío a nivel posgrado en la Especialidad en Mecatrónica y en la maestría en Diseño e Ingeniería de Sistemas Mecatrónicos, donde también participó en el diseño curricular y en la actualización del plan de estudios; en esta misma institución, además, imparte clases a nivel licenciatura.
- Participación en proyectos de investigación y de vinculación con empresas a través del sistema conacyt en colaboración con la Universidad de la Salle Bajío.
- Desarrollo de múltiples prototipos de sistemas mecatrónicos y robóticos.
- Fundador de Edu-training, donde actualmente brinda capacitación y consultoría en cursos de formación en las áreas de automatización y electrónica, sistemas de monitoreo a distancia y tecnología aplicada.
- En el 2014 fue coordinador de cursos de educación continua en las áreas de mecatrónica, dispositivos de control y automatización en la Universidad de la Salle Bajío.
- Del 2009 al 2011 llevó a cabo la tarea de director de la Facultad de Ingeniería Mecatrónica en la Universidad de León.

Dedicatoria

A Dios por darme la sabiduría y el ejemplo para seguir y cumplir mis sueños, sin Él no los podría llevar a cabo; gracias por darme todo lo que tengo y soy.

A mi esposa Mayte y a mis hijos Rubén y Darío; gracias por sus vidas, por su comprensión y amor. Los amo mucho.

A mis papás Rubén y Rosalía por su gran ejemplo y dedicación a lo largo de mi vida; muchas gracias por ser mis papás. Los quiero mucho.

A mis hermanos Tomás y Rosalía; gracias por su comprensión y apoyo.

Contenido

Capítulo 1

Bienvenido a Arduino y Visual Basic .NET	1
1.1 Introducción	2
1.2 Entorno de programación de Arduino IDE	2
1.2.1 Funciones básicas iniciales	2
1.2.2 Agregar comentarios	3
1.3 Conceptos básicos de programación	3
1.3.1 Declaración de variables y constantes	3
1.3.2 Arreglos con variables	5
1.3.3 Operaciones aritméticas	6
1.3.4 Asignaciones compuestas	6
1.3.5 Operadores de comparación	7
1.3.6 Operadores lógicos	7
1.3.7 Constantes	8
1.3.8 Estructuras de control	9
1.4 Tipos de comunicación	15
1.4.1 Comunicación Serial UART	15
1.4.2 Comunicación Serial por software	19
1.4.3 Librería SoftwareSerial	19
1.5 Tarjetas Arduino y los puertos de comunicación	19
1.5.1 Arduino UNO	19
1.5.2 Arduino MEGA	19
1.5.3 Arduino Due	19
1.5.4 Arduino YUN	20
1.6 Aspectos de comunicación serial con Visual Basic .NET	20
1.6.1 Clase SerialPort	20
1.6.2 Funciones de comunicación serial de Clase SerialPort	22
1.6.3 Puertos de comunicación y sus propiedades	23
1.6.4 Configuraciones iniciales y métodos para ejecutarse	25
1.6.5 Comandos de lectura y escritura	27
1.7 Resumen	27
1.8 Problemas	28

Capítulo 2

Aspectos generales de la programación en Visual Basic .NET	31
2.1 Introducción	32
2.2 Requerimientos de software y hardware	32
2.3 Configuración de hardware	32
2.4 Escritura de un programa en Visual Basic .NET	36
2.5 Manejo de controles en pantalla	37
2.6 Prueba de comunicación con Arduino	39
2.6.1 Comandos del control SerialPort	42

2.7	Resumen	46
2.8	Problemas	46
Capítulo 3		
Estación meteorológica de monitoreo con Arduino y Visual Basic .NET		49
3.1	Introducción	50
3.2	Requerimientos de software y hardware	50
3.3	Cómo conectar los diferentes componentes	51
3.3.1	Conexión de la tarjeta Arduino al protoboard	51
3.3.2	Conexión del sensor DHT11 al protoboard	51
3.3.3	Conexión de la fotoresistencia al protoboard	52
3.3.4	Conexión de la pantalla LCD	52
3.4	Prueba de los sensores	53
3.5	Despliegado de datos en la pantalla LCD	55
3.6	Pantalla de monitoreo del sistema	57
3.6.1	Pasos para crear la interfaz de monitoreo	58
3.6.2	Código para abrir el puerto y lectura de los valores enviados desde la Arduino	63
3.6.3	Resultados del sistema de los sensores en el sistema de monitoreo en tiempo real	63
3.7	Resumen	64
3.8	Problemas	64
Capítulo 4		
Detección de presencia inalámbrica con módulos XBee (sensores inalámbricos)		67
4.1	Introducción	68
4.2	Requerimientos de software y hardware	68
4.3	Configuración del hardware	69
4.4	Establecer la interfaz del el sensor PIR con Arduino	71
4.5	Programación del módulo XBee	72
4.5.1	Código del detector de presencia inalámbrico	73
4.6	Creación de la interfaz gráfica del detector de presencia	73
4.7	Otros ejemplos con esta misma aplicación	75
4.8	Resumen	78
4.9	Problemas	78
Capítulo 5		
Control de las luces desde una interfaz HMI		79
5.1	Introducción	80
5.2	Requerimientos de software y hardware	80
5.3	Configuración del hardware	80
5.3.1	Conexión de la tarjeta Arduino con los relevadores	81
5.3.2	Conexión del foco al relevador	81
5.4	Prueba de los relevadores	82
5.5	Creación de la interfaz gráfica para control de los relevadores	85
5.6	Prueba de la interfaz de comunicación	86
5.6.1	Aplicación web ASP.NET para control mediante comunicación serial	87
5.6.2	Creación del sitio web en Visual Basic.NET	92
5.7	Resumen	103
5.8	Problemas	103

Capítulo 6	
Control de un motor de corriente directa	105
6.1 Introducción	106
6.2 Requerimientos de software y hardware	106
6.3 Configuración del hardware	106
6.4 Prueba del motor	108
6.5 Control del giro y la velocidad de un servomotor	109
6.6 Creación de la pantalla de control	111
6.7 Resumen	113
6.8 Problemas	114
Capítulo 7	
Sistema de alarma inalámbrica	115
7.1 Introducción	116
7.2 Requerimientos de software y hardware	116
7.3 Configuración del hardware	116
7.3.1 Conexiones de los módulos transmisor y receptor	117
7.4 Comunicación serial inalámbrica	119
7.5 Prueba de los módulos de comunicación transmisor-receptor	121
7.6 Interfaz gráfica de monitoreo	122
7.7 Resumen	124
7.8 Problemas	124
Capítulo 8	
Estación de registro de datos	125
8.1 Introducción	126
8.2 Requerimientos de software y hardware	127
8.3 Configuración del hardware	127
8.4 Guardar los datos localmente mediante el módulo SD	128
8.5 Servidor de la base de datos	129
8.6 Inserción de los datos desde la aplicación	133
8.6.1 Control para insertar los datos	134
8.7 Pantalla de registro de datos	136
8.7.1 Mostrando los datos insertados	137
8.8 Envío de los datos a Excel para graficar los valores registrados	138
8.9 Resumen	143
8.10 Problemas	144
Capítulo 9	
Desarrollo de proyectos del Internet de las cosas basados en el Shield Ethernet de Arduino	145
9.1 Introducción	146
9.2 Requerimientos de software y hardware	146
9.3 Cómo se aprovecha la interacción entre servicios web y Arduino	146
9.3.1 El internet de las cosas	147
9.3.2 Usos de los servicios web	147
9.3.3 Estándares empleados en servicios web	149
9.4 Servicios web aplicados a Arduino	150

9.4.1	Ejemplo de servicio web con SOAP	150
9.4.2	Ejemplo de conversión de temperatura a través de un servicio web	154
9.4.3	Comandos de servicios RESTful con Arduino	172
9.4.4	Control del Ethernet Shield con ASP.NET	173
9.4.5	Monitoreo de un sensor de flujo de agua desde una página web en ASP.NET	176
9.4.6	Registro de datos en tiempo real de un panel solar a través de servicios web en la nube	190
9.4.7	Control de un módulo GSM/GPRS	200
9.4.8	Abrir una chapa al enviar un mensaje de texto SMS	210
9.4.9	Solicitud de temperatura y humedad con el sensor DHT11 a través de un mensaje SMS	213
9.4.10	Permitir un acceso mediante la huella digital	215
9.4.11	Monitoreo remoto con cámara web conectada a la nube	230
9.5	Resumen	245
9.6	Problemas	245

Capítulo 10

Prototipo de un Sistema de Control Supervisorio y Adquisición de datos a distancia (SCADA)

		247
10.1	Introducción	248
10.2	Requerimientos de software y hardware	248
10.3	Redes Industriales	248
10.4	Protocolos de comunicación industrial	251
10.4.1	Protocolo Modbus	252
10.5	Comunicación Modbus TCP/IP con Arduino y el Ethernet Shield	253
10.6	Configuración del hardware	254
10.7	Sistemas SCADA	258
10.7.1	Elementos que conforman un sistema SCADA	258
10.8	Servidores OPC	259
10.8.1	Clientes OPC	259
10.8.2	Servidor OPC de National Instruments	259
10.8.3	Configuración de las tags en el Servidor OPC	259
10.8.4	Clientes OPC	273
10.9	Módulo DSC de National Instruments	275
10.10	Cliente OPC en .NET	276
10.11	Implementación del prototipo de la aplicación del sistema de monitoreo y control	281
10.11.1	Enlazar las direcciones de los registros Modbus	282
10.11.2	Enlazar los controles de la aplicación con los tags	284
10.12	Control y monitoreo desde una página web	287
10.13	Resumen	294
10.14	Problemas	294

Capítulo 11

Rastreador móvil por medio de GSM/GPRS y GPS

11.1	Introducción	298
11.2	Requerimientos de software y hardware	298
11.3	Configuración del hardware	298
11.4	Módulo GPS para recibir coordenadas	299
11.5	Comunicación entre el módulo GSM/GPRS y el módulo GPS	302
11.6	Monitoreo remoto y rastreador móvil	304

11.7	Rastreador remoto	307
11.8	Resumen	315
11.9	Problemas	316
Capítulo 12		
Robot controlado inalámbricamente		317
12.1	Introducción	318
12.2	Construcción del robot móvil	318
12.3	Requerimientos de software y hardware	319
12.4	Configuración del hardware	321
12.5	Comunicación inalámbrica	325
12.6	Programación de los módulos	326
12.6.1	Código para la tarjeta Arduino UNO. Módulo 1.	326
12.6.2	Código para la tarjeta Arduino UNO. Módulo 2.	328
12.7	Prueba de los comandos desde el navegador web	331
12.8	Interfaz hombre-máquina	332
12.8.1	Código de la aplicación de los botones en la página web	333
12.8.2	Actualización de datos desde el Page Load	334
12.8.3	Autorefresh con Ajax Script Manager y Timer	334
12.8.4	Envío de comandos desde el sitio web	336
12.9	Control mediante la voz	337
12.9.1	Configuración del módulo de reconocimiento de voz	338
12.9.2	Grabar comandos de texto	339
12.9.3	Movimiento con base en los mensajes grabados	342
12.10	Integración de ambas tecnologías	345
12.11	Resumen	346
12.12	Problemas	346
Índice analítico		351

Plataforma de contenidos interactivos

Para tener acceso al material de la plataforma de contenidos interactivos del libro: *Acb]rcfeoz Vtbfcc` m UXei Jg]VCE XY XUrcg Vtb` 5fXi Jbc` m J]gi U` 6Ug]WB9H*, 1a. edición, siga los pasos de la primera página del libro.

NOTA: Se recomienda respaldar los archivos descargados en un soporte físico.

Introducción

Capítulo 1. Bienvenido a Arduino y Visual Basic .NET. A lo largo de este capítulo se hace una introducción al manejo de la plataforma de Arduino, a las instrucciones básicas de programación, a las configuraciones para preparar el sistema y a la comunicación para poder llevar a cabo los proyectos que se presentan en los siguientes capítulos.

Capítulo 2. Aspectos generales de la programación en Visual Basic .NET. En este apartado se explica la programación en Visual Basic .NET requerida para realizar las interfaces gráficas hombre-máquina; además, se realizará un programa con el que se comunicará Arduino con una pantalla hecha en .NET.

Capítulo 3. Estación meteorológica de monitoreo con Arduino y Visual Basic .NET. Bajo la guía de esta sección se conectará un sensor de temperatura y humedad DTH11 a una fotorresistencia, esto con el objetivo de realizar una estación meteorológica y enviar los datos capturados a una pantalla conectada a una PC a través del puerto serial. Además, se enlazará una pantalla LCD 20x4 y el módulo I2C para ejecutar la interfaz de la pantalla con la placa de Arduino, con lo cual se visualizarán constantemente los datos.

Capítulo 4. Detección de presencia inalámbrica con módulos XBEE. A lo largo de este capítulo se explica el uso de los módulos de comunicación XBEE para realizar una alarma inalámbrica integrando las tecnologías de Arduino y Visual Basic .NET; con ello, se buscará conectar un sensor de movimiento PIR y efectuar la conexión de los módulos de comunicación con el software de la interfaz.

Capítulo 5. Control de las luces desde una interfaz HMI. Esta sección abarca el control de dispositivos desde la interfaz, gracias a lo cual se enlazará un módulo de relevador al Arduino que se podrá controlar desde la interfaz HMI; por otro lado, se podrán también controlar dispositivos de potencia, tal como la conexión de una lámpara .

Capítulo 6. Control de un motor de corriente directa. En este capítulo se conectará un motor de corriente directa al Arduino a través de un puente H para controlar el giro y la velocidad; con ello, se realizará la interfaz requerida para el control del dispositivo.

Capítulo 7. Sistema de alarma inalámbrica. Este capítulo explica cómo realizar una alarma a través de comunicación inalámbrica; asimismo se llevará a cabo el diseño de la interfaz web para monitoreo del sistema.

Capítulo 8. Estación de Registro de datos. En este capítulo se integran los conocimientos de los proyectos realizados previamente, aquello se efectúa a través de un registro de datos de forma local. Por otro lado, se desarrollará como proyecto integrador un registro de información (*data logger*) en una base de datos y se realizará un reporte para consultarlo en Excel.

Capítulo 9. Desarrollo de proyectos del Internet de las cosas basados en el Shield Ethernet de Arduino. En éste se desarrollarán y aplicarán proyectos adaptados al Internet de las cosas para alcanzar la comprensión de las tecnologías enfocadas en determinadas áreas.

Capítulo 10. Prototipo de un Sistema de Control Supervisorio y Adquisición de datos a distancia (SCADA). Apartado en el cual se aplicarán los conceptos de los sistemas de monitoreo y control, desarrollando una aplicación de un sistema SCADA utilizando la tecnología .NET y la plataforma de Arduino.

Capítulo 11. Rastreador móvil por medio de GSM/GPRS y GPS. Gracias a este capítulo se desarrollará un proyecto de tecnologías aplicadas para sistemas web en la nube con el uso de la placa de GSM/GPRS y GPS; igualmente, se podrá elaborar un prototipo de un tracker móvil con tecnología de programación y Arduino con el objetivo de aplicarlo en un ejemplo real y práctico.

Capítulo 12. Robot controlado inalámbricamente. En esta sección se aplicará todo lo aprendido para desarrollar un prototipo de robot controlado inalámbricamente con las tecnologías de Arduino y diferentes aplicaciones de robótica; ello con el objetivo de que se puedan ejecutar posteriormente proyectos mecatrónicos propios.

Capítulo 1

Bienvenido a Arduino y Visual Basic .NET

- 1.1 Introducción
- 1.2 Entorno de programación de Arduino IDE
- 1.3 Conceptos básicos de programación
- 1.4 Tipos de comunicación
- 1.5 Tarjetas Arduino y los puertos de comunicación
- 1.6 Aspectos de comunicación serial con Visual Basic .NET
- 1.7 Resumen
- 1.8 Problemas

Objetivos

En este capítulo se asentarán las bases para utilizar la plataforma Arduino, se describirán las instrucciones básicas de programación y se mencionarán las configuraciones tanto para preparar el sistema como para establecer la comunicación, esto con la finalidad de que se sienten los cimientos para realizar los proyectos que se presentan en los siguientes capítulos.

1.1 Introducción

Este capítulo es una introducción a Arduino y a la programación entorno a él necesaria e importante para establecer una comunicación serial con los diferentes dispositivos externos para el control y la adquisición de datos. Dicha comunicación serial será fundamental en los proyectos de los siguientes capítulos. A continuación se explicará cómo se lleva a cabo la comunicación serial entre Visual Basic .NET y la tarjeta Arduino UNO, para enlazar los controles de los formularios y el aspecto gráfico con el hardware.

1.2 Entorno de programación de Arduino IDE

Arduino es un sistema de desarrollo para microcontroladores de la firma ATMEL. Fue desarrollado en Italia y está compuesto por un software editor-compiler (basado en Processing) en donde se escribe un programa en lenguaje C (basado en Wiring), así como un hardware que consiste en un microcontrolador ATMEL, el cual tiene precargado un sistema operativo (Bootstrap) que permite su programación directa *in-circuit* a través de señales seriales de comunicación.

1.2.1 Funciones básicas iniciales

Función Setup. Contiene todas las configuraciones iniciales del programa. Aquí se incluyen las condiciones iniciales para la operación de algunas instrucciones o librerías que se agregarán en el programa.

Función Loop. En ella se colocan todas las instrucciones que van a realizarse en forma repetitiva (*loop* significa lazo o bucle).

Además de estas funciones, se pueden agregar otras creadas por el usuario, aunque las anteriores no pueden omitirse en el programa. La estructura general de una función se describe en el ejemplo 1.1:



Ejemplo 1.1 Estructura general de una función.

```
void setup( )
{
  Instrucciones;
}
void loop( )
{
  Instrucciones;
}

void usuario( )
{
  Instrucciones;
}
```

Aquí se puede observar que cada función inicia con `void`, seguida del nombre de la función. En el caso de `setup` y `loop`, no es posible cambiar su nombre. En el caso de una función de usuario, es posible colocar cualquier nombre, empleando letras, números, guiones medio y bajo, aunque no acepta signos ni caracteres especiales. Seguido del nombre de la función se colocan los paréntesis (`()`). Entre las llaves (`{ }`) se colocan los comandos o instrucciones; al final de cada una, se debe agregar el punto y coma (`;`).

Es importante que en cada función e instrucción se respeten las mayúsculas y minúsculas, así como los espacios y símbolos que deben emplearse; de lo contrario, aparecerá un código de error.

1.2.2 Agregar comentarios

Es posible agregar comentarios a nuestro programa. Para ello, se colocan dos diagonales (`//`) seguidas del texto con el comentario que se desea mostrar. Pueden colocarse en cualquier parte del programa. (Ver ejemplo 1.2.)



Ejemplo 1.2 Agregar comentarios al programa.

```
Void setup( ){
pinMode(buttonPin, INPUT);    // la variable buttonPin es
configurada como entrada.
pinMode(ledPin, OUTPUT); // la variable ledPin es configura-
da como salida.
}
```

También es posible escribir un bloque de texto, colocando diagonal y asterisco (`/*`) al inicio del bloque y asterisco diagonal (`*/`) al final del bloque. (Ver ejemplo 1.3.)



Ejemplo 1.3 Escribir bloques de texto.

```
/*
Programa de Ejemplo para activar un led
En intervalos de 0.s segundos Empleando el led integrado
a la tarjeta Arduino Duemilanove (pin 13)
*/
```



1.3 Conceptos básicos de programación

En esta sección se revisarán algunos conceptos básicos de programación.

1.3.1 Declaración de variables y constantes

Una regla básica para emplear variables y constantes es que siempre deben declararse para que puedan utilizarse. También es importante indicar el tipo de variable según su

formato numérico. En función de su capacidad numérica, las variables y constantes pueden ser de tipo:

Byte. Valores numéricos con capacidad de 8 bits (0-255). (Ver ejemplo 1.4.)



Ejemplo 1.4 Declaración de variables tipo byte.

```
// Asigna a la variable tiempo una longitud de byte (0-255) sin valor inicial.  
byte tiempo;  
// Asigna a la variable temperatura una longitud de byte (0-255) iniciando con el valor de 0.  
byte temperatura = 0;
```

Int. Entero. Valores numéricos con capacidad de 16 bits con signo (-32768 y 32767). (Ver ejemplo 1.5.)



Ejemplo 1.5 Declaración de variables tipo int.

```
// Asigna a la variable como_se_llame, una longitud de int, sin valor inicial.  
int como_se_llame;  
  
// Asigna a la variable como_se_llame, una longitud de int, iniciando con el valor de 0.  
int como_se_llame = 1765;
```

Long. Extendido. Valores numéricos enteros con capacidad de 32 bits (-2147483648 a 2147483647). (Ver ejemplo 1.6.)



Ejemplo 1.6 Declaración de variables tipo long.

```
// Asigna a la variable cualquier_nombre, una longitud de long, sin valor inicial.  
long cualquier_nombre;  
  
// Asigna a variable cualquier_nombre, una longitud de long, iniciando con el valor de 0.  
long cualquier_nombre = 150000;
```

Float. Flotante. Valores numéricos con fracción decimal con capacidad de 32 bits (3.4028235E +38 y 3.4028235E -38). Es importante destacar que los resultados de las operaciones matemáticas sólo muestran dos decimales con redondeo. (Ver ejemplo 1.7.)

**Ejemplo 1.7 Declaración de variables tipo float.**

```
// Asigna a la variable ponle_un_nombre, una longitud de
int, sin valor inicial.
float ponle_un_nombre;

// Asigna a variable ponle_un_nombre, una longitud de
int, iniciando con el valor de 0.
float ponle_un_nombre = 3.14;
```

1.3.2 Arreglos con variables

También es posible hacer arreglos con variables. De esta forma, se pueden asignar variables con el mismo nombre pero asignando una posición; o bien, realizar matrices de valores indicando el tipo de dato, el tamaño y asignar valores a una posición específica. (Ver ejemplo 1.8.)

**Ejemplo 1.8 Arreglos con variables.**

```
/*
En este arreglo, el primer valor se encuentra en la posi-
ción 0 de mi variable, el segundo valor, en la posición
1, el tercer valor, en la posición 2, etcétera.
*/

int mi_arreglo[] = {10, 50, otra_variable}

En donde:

mi_arreglo[0] = 10
mi_arreglo[1] = 50
mi_arreglo[2] = lo que vale otra variable
```

De igual forma, es posible declarar una matriz de valores indicando el tipo de datos y el tamaño, y, posteriormente, asignar valores a una posición específica. (Ver ejemplo 1.9.)

**Ejemplo 1.9 Declarar matrices de valores.**

```
int mi_arreglo[5]; // declara un arreglo de ente-
ros de 6 posiciones (0 - 5).
mi_arreglo[3] = 10; // asigna el valor 10 a la posi-
ción 4 del arreglo.
```

Recordemos que el valor que se coloca entre corchetes ([]) representa el máximo valor de elementos en el arreglo, comenzando desde el cero. El cero cuenta como la posición uno.

Es posible asignar a una variable una determinada terminal de la tarjeta, definida como *entrada* o *salida*, tanto analógica como digital. (Ver ejemplo 1.10.)



Ejemplo 1.10 Asignar una determinada terminal de la tarjeta a una variable.

```
Int ledPin = 13; // asigna
la variable ledPin a la terminal (pin) 13 de la tarjeta
Arduino.
Int Valor_Analógico = 0; // asigna a la variable
Valor_Analógico al pin 0 de la tarjeta.
```

1.3.3 Operaciones aritméticas

Arduino puede manejar las cuatro operaciones aritméticas básicas: suma, resta, multiplicación y división. Para efectuar cualquiera de estas operaciones, es posible emplear números directos o variables. Cuando se emplean variables, es importante declarar cada variable con el formato numérico deseado (int, long, float, etc.). Hay que recordar que el único formato que puede realizar operaciones con valores decimales es el float. (Ver ejemplo 1.11.)



Ejemplo 1.11 Operaciones aritméticas.

```
X = y + 5;
Z = x - 30;
P = valor * 50;
R = valor1/valor2;
```

Si se requiere realizar alguna ecuación, se escribe la operación entre paréntesis (()). (Ver ejemplo 1.12.)



Ejemplo 1.12 Ecuaciones.

```
Temperatura = (valor_Analógico * 500) / 1023
```

1.3.4 Asignaciones compuestas

Las asignaciones compuestas combinan una operación aritmética con una variable asignada. Éstas comúnmente se utilizan en las condiciones de ciclos, tal como se describe más adelante (ver ejemplo 1.13). Estas asignaciones compuestas pueden ser las siguientes:

```

x ++    // igual que x = x + 1, o incrementar x en + 1
x --    // igual que x = x - 1, o decrementar x en -1
x += y  // igual que x = x + y, o incrementar x en +y
x -= y  // igual que x = x - y, o decrementar x en -y
x *= y  // igual que x = x * y, o multiplicar x por y
x /= y  // igual que x = x / y, o dividir x por y

```



Ejemplo 1.13 Asignaciones compuestas.

`x * = 3` hace que `x` se convierta en el triple del antiguo valor `x`, por lo tanto, `x` se reasigna al nuevo valor.

1.3.5 Operadores de comparación

Las comparaciones entre variables y constantes se utilizan con frecuencia en las estructuras condicionales `if` para establecer si una condición es verdadera, como se mostrará más adelante. Los operadores de comparación en Arduino son los siguientes:

```

x == y          // x es igual a y
x != y          // x no es igual a y
x < y           // x es menor que y
x > y           // x es mayor que y
x <= y         // x es menor o igual que y
x >= y         // x es mayor o igual que y

```

1.3.6 Operadores lógicos

Comúnmente, los operadores lógicos son una forma de comparar dos expresiones y dar como respuesta un estado *verdadero* o *falso*, dependiendo del operador lógico utilizado. Existen tres operadores lógicos: AND (`&&`), OR (`||`) y NOT (`!`), los cuales se utilizan generalmente en condicionales de tipo `if`. A continuación se muestran algunos ejemplos. (Ver ejemplo 1.14.)



Ejemplo 1.14 Ejemplos de operadores lógicos AND, OR y NOT.

Lógica AND:

```
if (x > 0 && x < 5) // condición es verdadera sólo si las dos expresiones
son ciertas
```

Lógica OR:

```
if (x > 0 || y > 0) // condición es verdadera si cualquiera de las expresiones
es cierta
```

Lógica NOT:

```
if (!x > 0) // condición es verdadera solo si la expresión (x > 0) es falsa
```


1.3.7 Constantes

El lenguaje de programación de Arduino contiene valores predeterminados, que son llamados “constantes”. Estos valores, realizan funciones específicas que pueden ser empleadas, principalmente, por valores digitales, o bien, por valores numéricos.

En el caso de una constante numérica, se antepone la palabra *const*, seguida de su formato numérico (int, float, byte, etc.). Finalmente, se le asigna un valor. (Ver ejemplo 1.15.)



Ejemplo 1.15 Ejemplos de constantes.

```
const int nombre_variable = 10;    // asigna el valor de
10 a la constante declarada
const float b = 3.14; // asigna el valor de 3.14 a la cons-
tante llamada b
const int ledPin = 13; // asigna el valor de 13 a la cons-
tante ledPin
```

Cuando se emplean valores digitales —también llamados valores booleanos—, se pueden utilizar las siguientes constantes asignadas, que la programación Arduino establece directamente.

TRUE/FALSE

FALSE se define como un valor de cero. TRUE se asocia con un valor de 1; sin embargo, cualquier entero que es no-cero es true, en un sentido booleano. Es decir, -1, 2 y -200 son todos true. Es importante considerar que las constantes TRUE y FALSE se escriben en mayúsculas. (Ver ejemplo 1.16.)



Ejemplo 1.16 Constantes TRUE y FALSE.

```
if (x == TRUE);
{
  ejecutar estas instrucciones;
}
```

INPUT/OUTPUT

Estas constantes se emplean al inicio del programa —dentro de la función void *_setup*—, para definir el sentido de una señal digital, si es de entrada —INPUT— o si es de salida —OUTPUT—. Estas constantes deberán escribirse siempre en mayúsculas. Para que puedan operar, se requiere anteponer la instrucción *pinMode*, seguida del valor de la terminal *pin*— que se desea direccionar separado éste por una coma (,) del sentido de la señal; todo esto, entre paréntesis (()). (Ver ejemplo 1.17.)

**Ejemplo 1.17 Constantes INPUT y OUTPUT.**

```
pinMode (13,OUTPUT); // asigna el pin 13 del Arduino
como una salida
```

También es posible que el pin sea asignado a través de una variable previamente definida. (Ver ejemplo 1.18.)

**Ejemplo 1.18 Pin asignado a través de una variable previamente definida.**

```
Int ledPin = 13; //
asigna a la variable ledPin el valor de 13
pinMode(ledPin, OUTPUT); // asigna el valor de la va-
riable ledPin como una salida
```

HIGH/LOW

Estas constantes también se escriben en mayúscula y establecen el estado lógico directamente de un pin de entrada/salida. HIGH establece que el nivel lógico es de 1; LOW establece que el nivel lógico es de 0. Estas constantes forman parte de las funciones de lectura –digitalRead– o escritura –digitalWrite– de valores digitales en los pines de entrada/salida de Arduino. (Ver ejemplo 1.19.)

**Ejemplo 1.19 Constantes HIGH y LOW.**

```
digitalWrite(ledPin, HIGH); // envía a un estado lógico 1
al pin establecido en la variable
digitalWrite(ledPin, LOW); // envía a un es-
tado lógico 0 al pin establecido en la variable
```

1.3.8 Estructuras de control

A continuación se revisarán algunas estructuras de control y sus formatos de instrucción.

Condiciona l if

If es una instrucción que se utiliza para determinar si una condición se ha cumplido; por ejemplo, averiguar si un valor analógico está por encima de un cierto número. Si la condición se cumple, deberá ejecutar una serie de instrucciones que se escriben dentro de llaves ({ }). Si no se cumple, el flujo del programa salta esta condición y no ejecuta las operaciones que están dentro de las llaves.

El formato de instrucción es el siguiente:

```
if (Variable Condición Valor)
{
  Instrucciones a ejecutar;
```

En donde:

Variable es un valor o dato que se desea condicionar. Puede estar dado en cualquier formato para valores numéricos.

Condición es cualquiera de los operadores de comparación (==, !=, >, <, >=, =<).

Valor es el dato numérico (valor directo o variable) que se desea comparar. (Ver ejemplo 1.20.)



Ejemplo 1.20 Ejemplos de condicional if.

```
If (temperatura >= 30)      // si la variable temperatura
es mayor o igual a 30:
{
  digitalWrite(ledPin, HIGH); // activa el led conectado
en el pin designado (ledpin).
}
```

```
If (valor_actual == valor_establecido) // si la varia-
ble valor_actual es igual a variable
{
  // valor_establecido
  Serial.print("Valor Alcanzado"); // envía por el serial
del Arduino el texto.
} // "Valor Alcanzado".
```

```
If (x != 50) // si la variable x no es igual a s0
{
  digitalWrite(10, LOW) // apaga el pin
10.
}
```

Condicional if else

If else es una instrucción que se emplea para definir lo que deberá realizarse en el programa en caso de que una determinada condición no se cumpla. Esto equivale a decir: "Si se cumple esta condición, realiza estas operaciones; de lo contrario (else), realiza éstas". (Ver ejemplo 1.21.)

El formato de instrucción es el siguiente:

```
if (variable condición valor)
{
  Instrucciones a ejecutar;
```

```

}
else
{
Instrucciones a ejecutar;
}

```



Ejemplo 1.21 Condición if else.

```

if (boton == HIGH) { // si el estado contenido en la
variable botón esta en alto
digitalWrite(13, HIGH); //entonces, activa el pin 13.
}
else { // de lo contrario,
digitalWrite(13, LOW); // apaga el pin 13.
}

```

Ciclo for

Este comando realiza un cierto número de veces un ciclo repetitivo de las operaciones que se encuentran dentro de él. Una vez que termina el ciclo, el programa continúa ejecutando las instrucciones fuera de éste. Si se quiere repetir el ciclo, se puede recomenzar.

El formato de instrucción es el siguiente:

```

for (Inicialización; Condición; Expresión)
{
Instrucciones a ejecutar;
}

```

En donde:

Inicialización es el valor inicial dado a una variable. Éste es el valor de inicio del ciclo.

Condición es el operador de comparación que indica hasta dónde puede llegar el valor de la variable (valor final).

Expresión indica si el conteo de los ciclos dentro del valor inicial y final, se incrementa o decrementa. Aquí se emplea una asignación compuesta (++ , --). (Ver ejemplo 1.22.)



Ejemplo 1.22 Ciclo for.

```

for (int x = 0; x < 12; x++) // x vale desde 0
hasta 12 y se incrementa de 1 en 1.
{

```

```

Serial.println(x); // escribe al serial el valor que
toma x
delay(500); // espera 0.s segundos antes de enviar
otro valor
}

```

Es importante observar en el ejemplo 1.22 que la variable *x* se puede declarar dentro de la instrucción como un valor *int* (entero), o bien, puede ser previamente declarada al inicio del programa, con cualquier formato numérico.

El ejemplo 1.23 muestra cómo podemos realizar ciclos en los que se incrementen de 2 en 2 los valores declarados en la función.



Ejemplo 1.23 Ciclos en los que se incrementan de 2 en 2 los valores declarados en la función.

```

for (int x = 0; x < 12; x+=2) // x vale
desde 0 hasta 12 y se incrementa de 2 en 2.
{
Serial.println(x); // escribe al serial el valor que
toma x.
delay(500); // espera 0.s segundos antes de enviar
otro valor.
}

```

En el ejemplo 1.24 se observa cómo realizar un ciclo negativo de valores –decremento– de 1 en 1.



Ejemplo 1.24 Ciclo negativo de valores.

```

for (int x = 10; x > -1; x--) // x vale
desde 10 hasta 0 y se incrementa de 1 en 1.
{
Serial.println(x); // escribe al serial el valor que
toma x.
delay(s00); // espera 0.s segundos antes de enviar otro
valor.
}

```

Comparador múltiple Switch Case

Esta instrucción relaciona los valores de una variable con diferentes condiciones; las compara y realiza las instrucciones indicadas para cada caso (*case*). Al finalizar las instrucciones que deberán ejecutarse en cada uno, se coloca una instrucción *break*, necesaria para que el programa pueda seguir su flujo normal. Al final del último *case*, se coloca el comando

default, para indicar que, si no se cumplió ninguno, el programa continúa en la siguiente línea. (Ver ejemplo 1.25.)

El formato de instrucción es el siguiente:

```
switch (var) {
  case etiqueta: instrucciones; break;
  case etiqueta: instrucciones;
  break;
  default:
  instrucciones;
}
```



Ejemplo 1.25 Comparador múltiple Switch Case.

```
switch (var) {
  case 1: // cuando valor
  de var sea igual a 1:
  digitalWrite(13, HI H);
  break;
  case 2: // cuando valor de
  var sea igual a 2:
  digitalWrite(13, LOW);
  break;
}
```

Ciclo while

Los ciclos while se ejecutan continuamente hasta que la expresión dentro del paréntesis (condición de comparación) deja de cumplirse. Algo debe modificar la variable comprobada; de lo contrario, el ciclo while nunca terminará. Lo que modifique la variable puede estar dentro del código, como una variable que se incrementa, o ser una condición externa; por ejemplo, el valor de un sensor.

El formato de instrucción es el siguiente. (Ver ejemplo 1.26):

```
while(expresión)
{
  instrucciones;
}
```



Ejemplo 1.26 Ciclo while.

```
while(var < 200) { // mientras que var sea menor que 200:
  Serial.println(var); // envía por el serial, el valor de
  var
  var++; // incrementa en 1 el valor de var
}
```