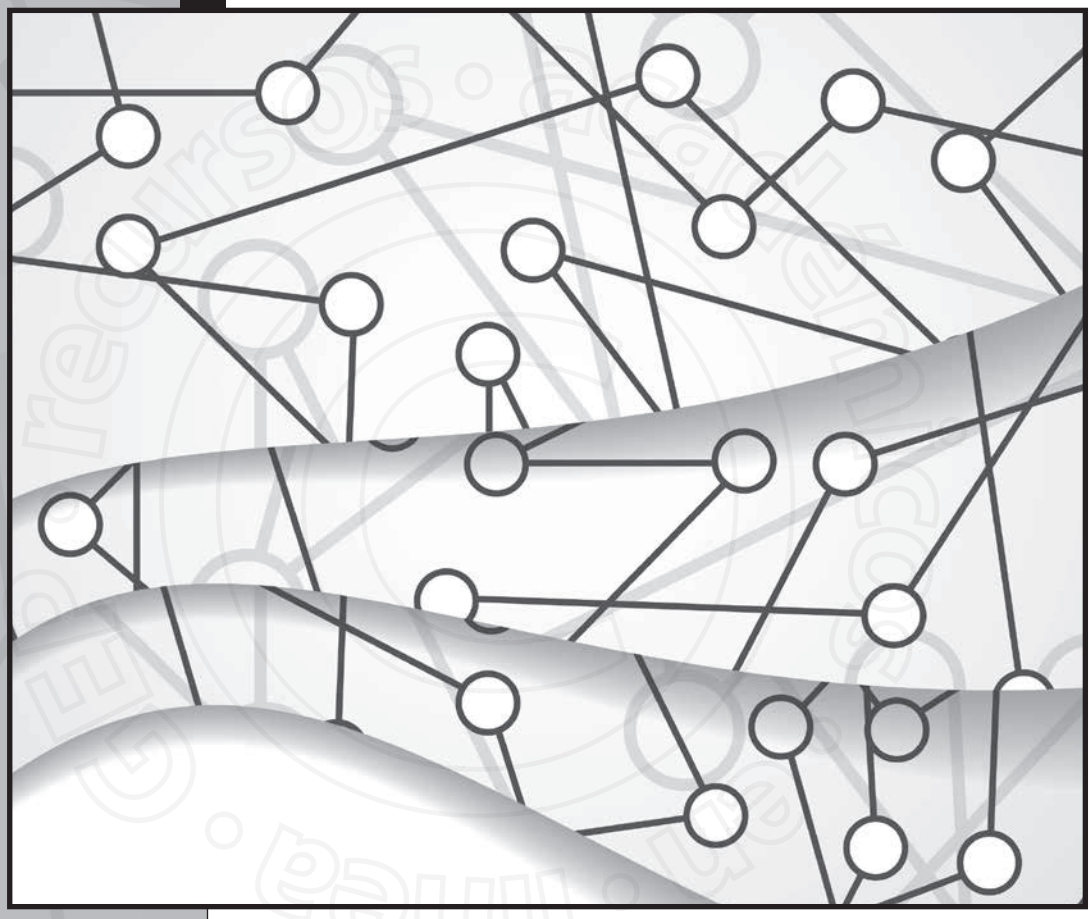
Introducción a Arduino®



¿Qué debo saber para la lectura de este capítulo?

Para el estudio de este apéndice es necesario tener conocimientos elementales de sistemas digitales y electrónica, además de contar con experiencia en el desarrollo de programas de cómputo, de preferencia relacionados con el manejo de variables, datos y rutinas de control en lenguaje C.

Objetivo general

Conocer y familiarizarse con el ambiente de trabajo de Arduino®, así como con su lenguaje de programación, alcances y limitaciones.

Objetivos particulares

- Conocer las características físicas, las capacidades y las especificaciones generales y particulares de la tarjeta Arduino®.
- Aprender las características del ambiente de trabajo y entorno de programación del software para la tarjeta Arduino®.
- Conocer e interpretar los tipos de datos, librerías, operadores, estructuras de control y demás esquemas de programación de la programación con Arduino®.

Generalidades

Arduino® es una plataforma electrónica integrada principalmente por un microcontrolador ATMEGA, de la compañía ATMEL, un lenguaje de programación y un conjunto de terminales de entradas y salidas, para el desarrollo de prototipos basada en software de código abierto y hardware flexible; está concebida para ser utilizada por profesionistas enfocados y dedicados a generar entornos u objetos interactivos, dado que esta puede tomar información de las variables y magnitudes físicas de una amplia gama de sensores mediante sus terminales de entrada, para, con base en esta información, modificar los parámetros de la red de control a través de los actuadores propios del sistema, y así interactuar con su entorno físico.

Debido a que Arduino® utiliza diferentes modelos de microcontroladores, existen múltiples versiones de estas tarjetas; no obstante, es importante hacer notar que todas se componen fundamentalmente por los siguientes elementos:

- Microcontrolador ATMEGA
- Regulador
- Cristal de reloj
- Interfaz USB
- Interfaz de programación SPI
- Puerto de programación
- Terminales de entradas y salidas
- Terminales analógicas
- Terminales digitales
- Terminales para comunicación con otros dispositivos

En la tabla A.1 se listan las características y especificaciones generales de algunos microcontroladores ATMEGA usados en las tarjetas Arduino®.

	ATMEGA168	ATMEGA328	ATMEGA1280
Voltaje operativo	5 V	5 V	5 V
Voltaje de entrada recomendado	7-12 V	7-12 V	7-12 V
Voltaje de entrada límite	6-20 V	6-20 V	6-20 V
Terminales de entrada y salida digital	14 (6 proporcionan PWM)	14 (6 proporcionan PWM)	54 (14 proporcionan PWM)
Terminales de entrada analógica	6	6	16
Intensidad de corriente	40 mA	40 mA	40 mA
Memoria flash	16 KB (2KB reservados para el sector de arranque)	32 KB (2KB reservados para el sector de arranque)	128 KB (4KB reservados para el sector de arranque)
SRAM	1 KB	2 KB	8 KB
EEPROM	512 bytes	1 KB	4 KB
Frecuencia de reloj	16 MHz	16 MHz	16 MHz

Fuente: ATMEL.COM/AVR.

Tabla A.1 Especificaciones generales de algunos microcontroladores ATMEGA.

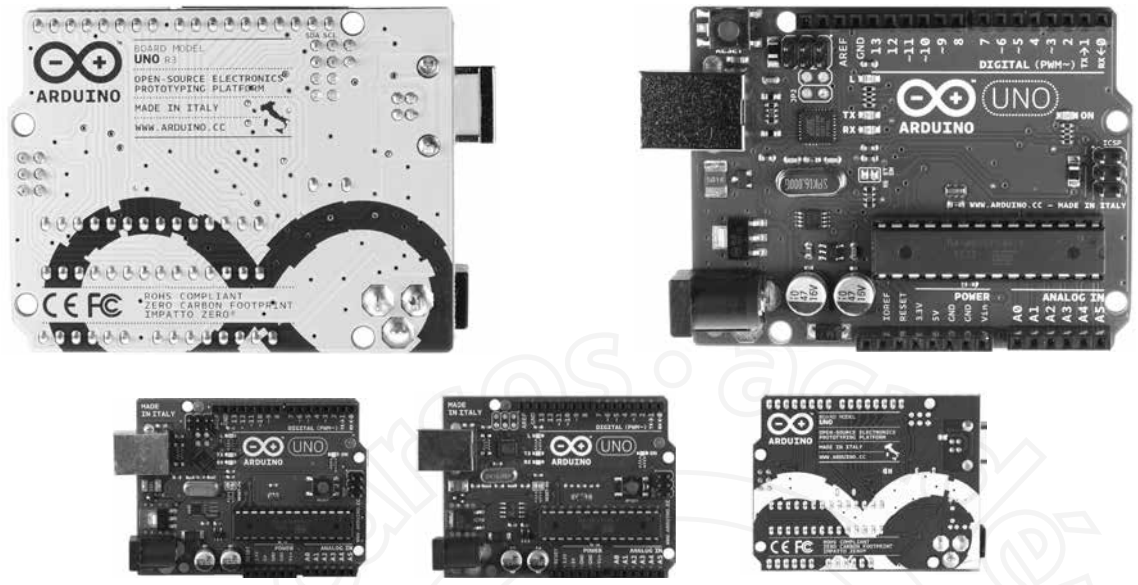


Figura A.1 Diferentes versiones de tarjetas Arduino Uno® (fuente: www.arduino.cc).

Con el fin de que no haya confusiones y la información acerca de este tema a lo largo de todo el libro sea lo más clara posible, es importante resaltar que cuando se hace mención a la tarjeta Arduino® en realidad se hace referencia de manera exclusiva a la versión Arduino Uno R3®; pues, en la actualidad existen diferentes categorías de tarjetas Arduino Uno®, entre las que destacan: Arduino Uno SMD®, Arduino Uno R2® y Arduino Uno R3®. En la figura A.1 se muestran todas estas tarjetas y la disposición física de los elementos que las constituyen.

Descripción del hardware (Arduino Uno R3®)

Características generales de la tarjeta

Esta tarjeta se fundamenta en el microprocesador ATMEGA328. Contiene 14 terminales digitales de entrada/salida; de estas, las terminales 3, 5, 6, 8, 10 y 11 se puede utilizar como salidas PWM. Además, también tiene 6 entradas analógicas con una resolución de 10 bits, que proporcionan un número entero de 0 a 1 023, un resonador cerámico de 16 MHz, una conexión USB, un conector de alimentación, un header ICSP y un botón de reinicio. La tabla A.2 resume las características fundamentales de esta tarjeta.

Microcontrolador	ATMEGA328
Voltaje de trabajo	5 V
Voltaje de entrada (recomendado)	7-12 V
Pines digitales de E/S	14 (6 salidas de PWM)
Pines de entrada analógica	6
Corriente DC por Pin de E/S	40 mA
Memoria flash	32 KB (0.5 KB usada en bootloader)
SRAM	2 KB
EEPROM	1 KB
FOSC	16 MHz

Fuente: ATMEL.COM/AVR.

Tabla A.2 Características generales de la tarjeta Arduino Uno®.

Alimentación

La tarjeta Arduino Uno® puede ser alimentada mediante la conexión USB o a través de una fuente de alimentación externa (la fuente de alimentación se selecciona de manera automática). Así, cualquier adaptador AC-DC o batería dentro del voltaje de entrada permitido puede ser utilizado como fuente de alimentación externa (no USB). Para el caso del adaptador, este es acoplado mediante el uso de un conector plug de 2.1 mm, con polaridad positiva en el centro, a la terminal de alimentación de la tarjeta Arduino®. Por otro lado, si se requiere usar una batería o juego de baterías, las terminales de estas deben ser conectadas a las terminales Gnd y Vin del conector de alimentación.

La tarjeta puede funcionar con un voltaje de 6 a 20 V; sin embargo, si se suministra un voltaje menor a 7 V, existe la posibilidad de que la terminal de 5 V del Arduino® pueda suministrar un voltaje menor; por otra parte, si se utiliza un voltaje de más de 12 V, el regulador de voltaje se puede sobrecalentar y dañar la tarjeta. Por lo anterior, el rango recomendado es de 7 a 12 V.

Las terminales de alimentación de la tarjeta son:

- **VIN.** Terminal de entrada de voltaje a la tarjeta Arduino® cuando se utiliza una fuente de alimentación externa y no la conexión USB u otra fuente de alimentación regulada. También es posible usar esta terminal para proveer voltaje cuando el Arduino® está energizado a través del conector de alimentación (power jack).
- **5V.** Terminal de salida de suministro de alimentación regulada utilizada para alimentar el microcontrolador y otros componentes de la tarjeta. Esta terminal puede provenir de VIN, a través del regulador en la tarjeta, o ser proporcionada por USB u otra fuente regulada de 5 V.
- **3V3.** Terminal que otorga un suministro de 3.3 V generados por el regulador en la tarjeta. En este caso, la corriente máxima es de 50 mA.
- **GND.** Corresponde a las terminales de referencia a tierra (ground).
- **IOREF.** Terminal que se utiliza para proporcionar el voltaje de referencia con el que opera el microcontrolador.

Memoria

El microcontrolador ATMEGA328 cuenta con la siguiente distribución de espacio de memoria:

- **Flash 32KB** (de estos, 0.5 KB son utilizados por el gestor de arranque del microcontrolador)
- **SRAM 2KB**
- **EEPROM 1KB**
- **Memoria Flash.** Espacio del programa donde Arduino® almacena el *sketch*[1].
- **SRAM** (Static Random Access Memory). Memoria donde los sketches almacenan y manipulan variables al ejecutarse.
- **EEPROM** (Electrically Erasable Programmable Read-Only Memory). Un espacio de memoria que suele ser utilizado por los programadores para almacenar información de largo plazo.

Es importante destacar que las memorias Flash y EEPROM son no-volátiles; esto significa que la información se mantiene en estas aún después de cortar la alimentación. En cambio, la memoria SRAM sí es volátil, por lo que la información almacenada se pierde cada vez que se reinicia la unidad.

Terminales (Input y Output)

Terminales digitales

Cada una de las terminales digitales de la tarjeta Arduino® puede ser configurada como entrada o salida de propósito general a través de las funciones pinMode(), digitalRead() y digitalWrite(), las cuales se abordan con detalle más adelante en este apéndice. Estas terminales operan a 5V; además, cada terminal provee o recibe un máximo de 40 mA y tiene una resistencia interna pull-up de 20 a 50 KΩ,

[1] Código del programa considerado un boceto; en inglés: "sketch".

que se puede activar o desactivar mediante el uso de la función DigitalWrite( ), con un valor de HIGH o LOW, respectivamente, aunque, de manera general, esta resistencia se encuentra desactivada. Es importante aclarar que algunas terminales tienen funciones especializadas.

Terminales digitales con funciones especializadas

- **Serial: 0 (RX) y 1 (TX).** Se utilizan para recibir (RX) y transmitir (TX) datos serie del tipo TTL. Estas terminales están conectadas a las correspondientes terminales del circuito integrado ATMEGA8U2 USB- to- TTL Serial.
- **Interrupciones externas: 2 y 3.** Estas terminales pueden ser configuradas para disparar una interrupción con un valor bajo, un pulso de subida o de bajada, o un cambio de valor. Para más detalles revisar la función attachInterrupt( ).
- **PWM: 3, 5, 6, 9, 10, y 11.** Estas terminales proporcionan salidas PWM de 8 bits mediante el uso de la función analogWrite( ).
- **SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK).** Estas soportan el protocolo SPI (Serial Peripheral Interface) usado por los microcontroladores para lograr una comunicación rápida y a distancias cortas, con uno o más dispositivos periféricos. En el caso de estas terminales siempre existe un dispositivo maestro que, en general, es el microcontrolador conectado con otros elementos periféricos esclavos. De manera típica, existen tres líneas comunes para todos los dispositivos: *MISO*, línea por la cual el esclavo envía datos al dispositivo maestro; *MOSI*, línea por la cual el maestro envía datos a los periféricos; *SCK*, línea para la sincronización con los datos de transmisión generados por el maestro.

  Además, existe una línea específica llamada *SS*, que constituye la línea selectora con la cual el maestro puede habilitar o deshabilitar los dispositivos periféricos.

  Para mayor detalle al respecto, se sugiere revisar la librería SPI y la documentación del fabricante.
- **LED: 13.** En esta terminal hay un led conectado a la terminal 13, así que cuando un valor HIGH está presente en la terminal, el led está activado; por el contario, cuando existe un valor LOW, el led permanece apagado.

Terminales analógicas

Además de las terminales digitales, la tarjeta Arduino® tiene 6 entradas analógicas, etiquetadas desde A0 hasta A5, cada una de las cuales se lee mediante el convertidor analógico digital con 10 bits de resolución; es decir, tiene 1 024 valores diferentes. De fábrica, el rango de medición es desde la referencia a tierra hasta 5 V, sin embargo, es posible cambiar el límite superior y su rango usando la terminal AREF y la función analogRead( ).

De igual manera que en el caso de las terminales digitales, en las analógicas también se tienen algunas terminales especializadas.

Terminales analógicas con funciones especializadas

- **TWI: SDA y SCL.** El microcontrolador ATMEGA328 también soporta el protocolo de comunicación I^2C/TWI, la interfaz SDA, o línea de datos que se encuentra en la terminal 4, así como la interfaz SCL, o línea de reloj que se encuentra en la terminal 5, para simplificar el uso del bus I^2C. El fabricante proporciona la librería Wire; se recomienda leerla para una referencia e información más completas.
- **AREF.** Esta terminal se utiliza para referenciar el voltaje necesario en las entradas analógicas, lo que se logra con la ayuda de la función analogReference( ).
- **Reset.** Al situar el estado en LOW de esta terminal es posible restablecer el microcontrolador. Esta se suele usar para adicionar un botón de reset extra, el cual bloquea el reset principal de la tarjeta.

Instalación de la tarjeta Arduino®

Instalación de los Drivers en Window

En el caso de los equipos de cómputo con sistema operativo Windows Vista, Windows 7 o Windows 8, al conectar la tarjeta Arduino®, la descarga e instalación de los drivers se inicia de manera automática.

Sin embargo, en el caso de las computadoras con sistema operativo Windows XP, esto se realiza al poner en práctica los siguientes pasos:

1. Abrir el diálogo de instalación de Nuevo Hardware, como se observa en la figura A.2.
2. Seleccionar la opción para que el sistema no utilice Windows Update en la búsqueda del software.

Figura A.2 Diálogo de instalación de nuevo hardware.

3. De manera inmediata se despliega el cuadro de diálogo que se observa en la figura A.3. Seleccionar de este cuadro la opción inferior: *"Instalar desde una ubicación específica"*.

Figura A.3 Instalación desde una ubicación específica.

4. Enseguida aparece el cuadro de diálogo que se observa en la figura A.4. Marcar la opción superior: *"Buscar el controlador más adecuado"* y la casilla *"Incluir esta ubicación en la búsqueda"*, esta es la dirección de la carpeta: FTDI USB Drivers, contenida en la carpeta original de Arduino®.

Figura A.4 Búsqueda del controlador.

5. Cuando se ha localizado e instalado el driver, el sistema muestra el cuadro de diálogo que se observa en la figura A.5. Dar clic en finalizar.

Figura A.5 Finalización de la instalación.

Ambiente de programación

Para la programación del microcontrolador de la tarjeta se utiliza un lenguaje de programación que el fabricante nombró como "Arduino®", que se basa en C/C++; por tanto, este soporta las construcciones de C estándar y algunas funcionalidades de C++, además de que está orientado en

Wiring[2] y vincula la librería AVRLibc, lo que permite el uso de todas sus funciones. Los programas elaborados con Arduino® pueden dividirse en las partes básicas siguientes: estructura, funciones, variables y constantes.

Además, mediante este programa, los proyectos pueden ejecutarse sin necesidad de estar conectados a una computadora, si bien tienen la posibilidad de hacerlo y comunicarse con diferentes tipos de software (por ejemplo, Flash, Processing, MaxMSP). Por otra parte, el entorno de desarrollo Arduino® se basa en Processing[3].

Entorno de desarrollo para Arduino®

El entorno de desarrollo para Arduino® admite la conexión con el hardware de la tarjeta, con el fin de cargar los programas y comunicarse con estos (véase figura A.6); dicho entorno está constituido de manera principal por:

1. Editor de textos.
2. Área de mensajes.
3. Barra de funciones rápidas.
4. Barra de menús.

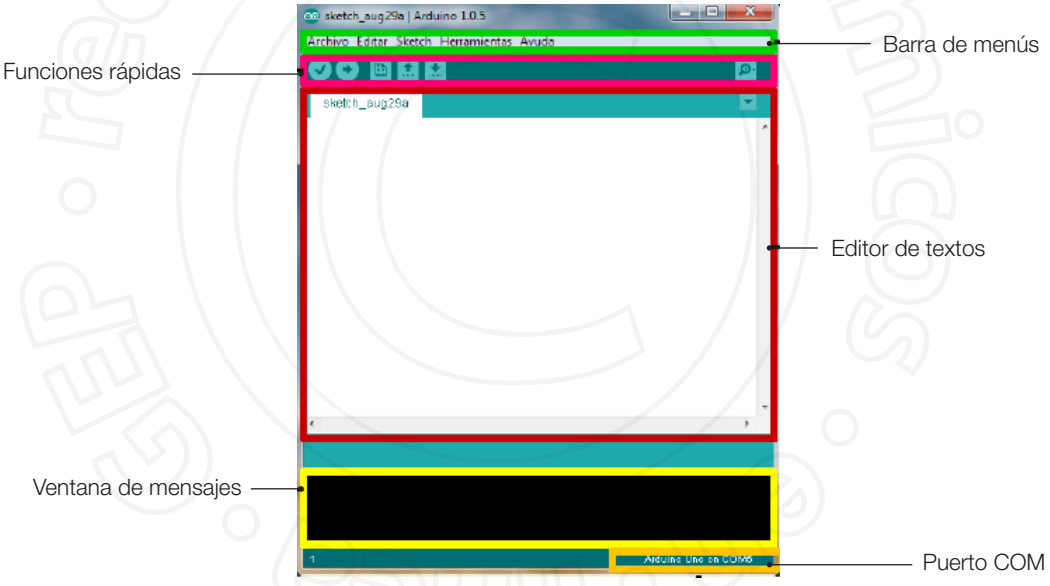


Figura A.6 Entorno de desarrollo para Arduino®.

Editor de textos

Para escribir el código en Arduino® se suele utilizar lo que se denomina "sketch" (programa). Con este componente es posible cortar, pegar, buscar y reemplazar texto.

Área de mensajes

Área donde se exhibe la información mientras se "cargan" los programas y donde también se señalan los posibles errores.

[2] Plataforma de programación de software libre que permite escribir un código para controlar un amplio rango de dispositivos dispuestos a partir de diferentes tarjetas de microcontroladores.

[3] Ambiente de lenguaje de programación orientado al software libre.

La barra de funciones rápidas que se observa en la figura A.7 permite verificar el proceso de carga, la creación, la apertura y el guardado de programas, así como el monitoreo serial de manera simple.

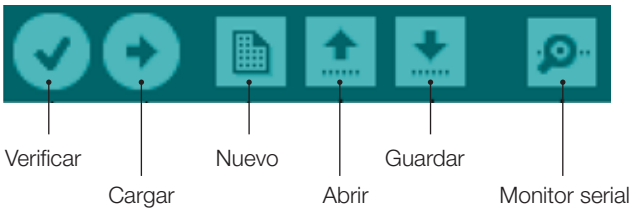


Figura A.7 Barra de funciones rápidas.

La barra de menús está constituida por los menús: archivo, editar, scketch y herramientas, los cuales se muestran y describen en las siguientes tablas.

	Menú archivo
Nuevo	Crea un nuevo sketch
Abrir	Abre un sketch ya existente
Sketchbook	Muestra el sketchbook de Arduino®
Ejemplos	Muestra el menú de ejemplos que contiene el IDE
Cerrar	Cierra el sketch
Guardar	Guarda el sketch
Guardar como	Guarda el sketch con otro nombre
Cargar	Carga el programa a la tarjeta Arduino®
Cargar usando programador	Carga el programa a la tarjeta con ayuda de otro programador (ICSP)
Configuración de página	Configura el tamaño de página para impresión
Imprimir	Envía a imprimir el sketch
Preferencias	Permite modificar algunas preferencias del IDE
Salir	Cierra el IDE

Tabla A.3

	Menú editar
Deshacer	Deshace la acción anterior
Rehacer	Rehace la acción anterior
Cortar	Corta un texto seleccionado al portapapeles
Copiar	Copia un texto seleccionado al portapapeles
Copiar para el foro	Copia el código en un formato para el foro de Arduino®
Copiar como HTML	Copia el código en un formato para HTML
Pegar	Pega algún texto del portapapeles
Seleccionar todo	Selecciona todo en el editor de texto

Comentar/Quitar comentario	Convierte en comentario algún texto seleccionado
Incrementar margen	Incrementa el margen del editor de texto
Reducir margen	Reduce el margen en el editor de texto
Buscar	Busca algún texto específico dentro del código

Tabla A.4

	Menú sketch
Verificar/Compilar	Compila el código para corroborar que no haya errores
Mostrar la carpeta de sketch	Muestra la carpeta donde se encuentra el sketch actual
Agregar libro	Agrega un archivo más al sketch
Importar librería	Incluye una librería a utilizar dentro del código

Tabla A.5

	Menú herramientas
Formato automático	Da formato automático al texto, lo ordena
Archivar el sketch	Crea una carpeta comprimida con el sketch actual
Monitor serial	Abre la ventana del monitor serial
Tarjeta	Elige la tarjeta Arduino® a utilizar
Puerto serial	Elige el puerto COM asignado a la tarjeta Arduino®
Programador	Elige el programador a utilizar

Tabla A.6

Tipo de datos

Como se dijo antes, la programación en Arduino® se basa en C/C++; por tanto, los tipos de datos utilizados son semejantes. A continuación se presentan los diferentes tipos de datos, además de que se anexa un ejemplo de código.

Datos tipo booleano

Un dato de tipo **booleano** solo puede tomar dos valores: **verdadero** o **falso**. Cada booleano ocupa un único byte en la memoria.

Ejemplo

```
boolean bool = false; // Se crea la variable booleana bool y le asigna el
         // valor de falso.
```

Datos tipo char

Es un tipo de dato que ocupa un byte de memoria y almacena un valor de caracter. Los caracteres, como las literales, se escriben con comillas simples, por ejemplo: 'a'; en tanto, para varios caracteres, como los strings, se utilizan comillas dobles, por ejemplo: "abc".

Los caracteres son almacenados como números, ya que están codificados según el código ASCII; por tanto, es posible realizar cálculos aritméticos con estos, por ejemplo: 'a'+1 tiene un valor de 98, ya que el valor ASCII de la letra minúscula a es 97.

Ejemplo

```
char valorc = 'a';
char valorc = 97; // En los dos casos se tiene el mismo valor.
```

Datos tipo byte

Para este tipo de datos, un byte almacena un número sin signo en un registro de 8 bits; esto es, desde 0 hasta 255.

Ejemplo

```
byte b = B110; // Se crea la variable b, donde "B" indica
.              // que el dato estara dado en formato  binario.
```

Datos tipo int

Los datos int son el tipo de datos más usado; estos almacenan valores de **2 bytes**. Esto produce un rango entre -32 768 o -2 ^ 15 hasta 32 767 o (2 ^ 15)- 1. En el caso de las variables, se pueden almacenar números negativos con la técnica denominada **complemento a dos**. El bit más alto, en ocasiones llamado "**sign bit**", indica que el número es negativo. Para esto se invierte el valor de cada uno de los **bits**; es decir, se realiza el complemento a uno y se suma 1 al número obtenido.

Ejemplo

```
int indicador = 13;
```

Sintaxis

```
  int variable = valor;

variable - Nombre de la variable int.
valor - Valor asignado a dicha variable.
```

```
   int variable
   variable = -32,768;
   variable = variable - 1;   // Ahora variable contiene 32,767.

   variable = 32,767;
   variable = variable + 1; // Ahora variable contiene -32,768.
```

Nota

Cuando las variables exceden su límite, estas vuelven a su capacidad mínima, esto sucede en ambas direcciones (roll over):

Datos tipo unsigned int

Los enteros sin signo (unsigned int) también trabajan con un valor de dos bytes. Sin embargo, solo almacenan valores positivos, generando un rango útil desde 0 a 65535 o 2 ^ 16 - 1.

Ejemplo

```
    unsigned int indicador = 13;
```

Sintaxis

```
     unsigned int variable = valor;

variable - El nombre de la variable unsigned int.
valor - El valor que  se asigna a  esa variable.
```

```
unsigned int variable
  variable = 0;
  variable = variable - 1; // Ahora  variable contiene 65535.
  variable = variable + 1; // Ahora variable contiene 0.
```

Datos tipo long

Las variables tipo long constituyen variables de tamaño extendido para almacenamiento de números en 32 bits (4 bytes), desde el rango: 2147483648 hasta 2147483647.

Ejemplo

```
long variable = 186000L;   // Observe la 'L' al final del numero indicando
       // que el dato es de tipo long
```

Sintaxis

```
    long variable = valor;
variable - Nombre de la variable tipo Long.
valor - Valor asignado a la variable.
```

Datos tipo unsigned long

Las variables de este tipo son variables extendidas para almacenar números en 32 bits (4 bytes). Al contrario de las variables long estándar, las unsigned long no almacenan números negativos, haciendo que su rango sea desde 0 a 4294967295; es decir 2^32 - 1.

Ejemplo

```
unsigned long variable;
```

Sintaxis

```
unsigned long variable = valor;

variable - El nombre de la variable long
valor - El valor que se asigna a la variable long
```

Datos tipo float

Son el tipo de datos para números de punto flotante, es decir aquellos que tienen punto decimal. Se suelen emplear para aproximar valores analógicos y continuos, debido a que ofrecen una mayor resolución que en el caso de los enteros. El rango de los datos tipo float varía desde 3.4028235E+38 hasta 3.4028235E+38; ocupan 4 bytes (32 bits) de información.

Estos datos tienen una precisión de 6 o 7 dígitos decimales, que es el número total de dígitos, mas no el número de dígitos a la derecha del punto decimal.

Para la realización de operaciones matemáticas con punto flotante es necesario más tiempo de procesamiento, comparado con el tiempo requerido para las operaciones realizadas con números enteros, por lo que cuando el tiempo es un factor crítico y el manejo de datos de punto flotante en realidad no es necesario, hay que tratar de evitar este tipo de operaciones; por ejemplo, en el caso de un programa para aplicaciones que tiene que ejecutarse en tiempo real, por lo regular se asignan equivalencias, a fin de convertir las operaciones de punto flotante a operaciones con enteros, con lo que se aumenta la precisión y velocidad del procesamiento.

Ejemplo

```
int variable1;
   int variable2;
   float variable3;

variable1 = 1;
   variable2 = variable1 / 2;        // Ahora variable2 tiene 0,
   variable3 = (float)variable1/2.0;  // Ahora variable3 es 0.5.
```

Sintaxis

float variable = valor;

```
variable - El nombre de la variable tipo float.
valor - El valor que se asigna a esa variable.
```

Datos tipo double

Son datos de punto flotante de doble precisión, los cuales ocupan 4 bytes. Sin embargo, al contrario de otras plataformas, a través de las cuales es posible obtener mayor precisión usando una variable tipo double (por ejemplo, por encima de 15 dígitos), con el uso de Arduino®, los datos tipo double tienen la misma capacidad que los float y, en consecuencia, no existe mayor precisión.

Datos tipo string

Los datos de este tipo se representan como arreglos de caracteres tipo char, que terminan con el caracter NULL; por ejemplo, las siguientes son declaraciones válidas de strings.

```
char Str1[15];
  char Str2[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o'};
  char Str3[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o', '\0'};
char Str4[ ] = "Arduino";
  char Str5[8] = "Arduino";
  char Str6[15] = "Arduino";
```

Observaciones de la declaración de strings

- Declarar un arreglo de caracteres sin inicializarlo, como en Str1.
- Declarar un arreglo de caracteres (con un caracter extra) y el compilador añadirá el caracter NULL requerido, como en STR2.
- Explicitar el caracter NULL, como en Str3.
- Inicializar con un string constante entre comillas dobles; el compilador medirá el tamaño del arreglo para ajustar el string constante y el caracter NULL para finalizar, como en Str4.
- Inicializar el arreglo con un tamaño explícito y un string constante, como en Str5.
- Inicializar el arreglo dejando un espacio extra para un string más largo, como en Str6.

Terminación NULL

En general, los strings se finalizan con un caracter NULL (código ASCII 0), esto permite a funciones como Serial.print() establecer dónde está el final del string. De otra forma, seguiría leyendo los siguientes bytes de la memoria que no forman parte del string.

Comillas simples o dobles

Los string siempre se definen entre comillas dobles ("Abc") y los caracteres entre comillas simples ('A').

Arreglos de strings

Al trabajar con grandes cantidades de texto es conveniente configurar un arreglo de strings. Como los strings son en sí mismos arreglos, estos constituyen un ejemplo real de un arreglo bidimensional. Un asterisco después del tipo de dato char, "char*", indica que este es un arreglo de "punteros"; como todos los nombres de arreglos son punteros, para crear un arreglo de arreglos es necesario hacer esto.

Ejemplo

```
char* cadenas[]={"Primera", "Segunda", "Tercera"};
```

Datos tipo Arrays

Una matriz o "arreglo" es una colección de variables a las que se accede mediante un número de índice.

Declarando una matriz

Todos los métodos siguientes son formas válidas de crear o declarar una matriz.

```
int m1[4];
  int m2[] = {1, 2, 3};
  int m3[4] = {1, 2, 3};
char m4[4] = "lcr";
```

Es importante resaltar que una matriz se puede declarar sin inicializarla, como se muestra en la primera declaración para m1. En tanto, como se ve en la segunda línea de dicha declaración, el arreglo m2 se declara sin un tamaño explícito; el compilador cuenta el número de elementos y crea la matriz con el tamaño apropiado. Además, es posible declarar el tamaño de la matriz e inicializarla al mismo tiempo, como en la tercera línea donde se declara m3.

Acceso a una matriz o arreglo

Los arreglos son **cero indexados**, esto significa que, al referirse a una matriz, el primer elemento de dicha matriz está en el índice 0. Por tanto, respecto a la línea 3 de las declaraciones anteriores, m3[0] es igual a 1, m3[1] es igual a 2 y m3[2] es igual 3. Esto significa que para la matriz m3 de 3 elementos, el índice 2 es el último elemento. Por ende, el valor de m3[2] es 2; sin embargo, m3[3] es inválido.

El acceso más allá del índice final de una matriz (mediante el uso de un número de índice mayor que el tamaño declarado -1) resulta en una lectura de la memoria que está en uso para otros fines. Escribir en las localidades de memoria aleatoria, a menudo, puede conducir a resultados inesperados, como fallos o mal funcionamiento del programa. Este también puede ser un error difícil de encontrar.

A diferencia de BASIC o JAVA, el compilador de C no realiza ninguna comprobación para verificar si el acceso a una matriz se encuentra dentro de los límites del tamaño de la matriz que se ha declarado.

En resumen, para asignar un valor a una matriz se escribe:

```
m1[0] = 0
```

Y para recibir un valor desde una matriz se escribe:

```
variable = m1[0]
```

Matrices y los ciclos FOR

Muchas veces, las matrices se utilizan en el interior del ciclo for, donde el contador de ciclo se usa como el índice de cada elemento de la matriz. Por ejemplo:

```
int i; //Declaración de la variable i
for (i = 0; i < 3; i = i + 1) //Declaracion del ciclo FOR, donde
     //i toma valores de 0 a 2 con
     //incrementos de 1
{
m1[i]=m2[i]*2; //Uso de i como indice de la matriz

}
```

Datos tipo void

La palabra reservada void indica que lo esperado es que no devuelva información a la función donde fue llamada. En general, esta se usa en la declaración de funciones. Por ejemplo, en las siguientes líneas, dentro de la función acondicionamiento, se realiza un escalamiento de los valores en las variables; sin embargo, en este caso no se reportará ninguna información al programa principal.

```
void acondicionamiento()
{// escalamiento de valores en las variables//}
  void  escaneo()
{...}
```

Librerías

El entorno de Arduino® puede **extenderse a través** del uso de librerías, ya que estas proveen funcionalidad adicional para el uso de *sketches*, por ejemplo al trabajar con hardware o manipulación de datos. Constituyen una recopilación de código que puede hacer más fácil el trabajo con sensores, display, módulos rf, etcétera. Hoy día, existen cientos de librerías adicionales disponibles en Internet listas para ser usadas.

Del total de librerías, hay un importante número de de estas que ya se encuentran previamente instaladas con el IDE de Arduino®, pero además también se pueden descargar e instalar librerías adicionales. Para usar una librería dentro de un *sketch*, esta debe seleccionarse desde: **Sketch > Import Library** (***Importar Librería***).

Ahora bien, si se desea instalar librerías alternas a las que ya contiene la tarjeta de Arduino®, primero es necesario descargar la librería en cuestión y descomprimirla en una carpeta; esta contiene dos archivos, uno con sufijo "*.h*" y otro con sufijo "*.cpp*". En el directorio *sketchbook* se encuentra la carpeta "*libraries*" (si no existe, es necesario crearla), que es donde debe almacenarse la carpeta que contiene los archivos de la nueva librería, la cual ya estará disponible para ser importada al reiniciar el IDE de Arduino®.

En las siguientes tablas se listan las características de las librerías más importantes.

Librerías estándar	
EEPROM	Para leer y escribir en memorias "permanentes"
Ethernet	Para conectar a Internet mediante el uso del *Ethernet Shield*
Firmata	Para comunicarse con aplicaciones en la computadora mediante el uso de un protocolo estándar Serial
LiquidCrystal	Para controlar displays de cristal líquido (LCD)
Servo	Para controlar servomotores
SoftwareSerial	Para la comunicación serial de cualquier terminal digital
Stepper	Para controlar motores paso a paso (*Stepper motors*)
Wire	Interfaz de dos cables, oTwo Wire Interface (*TWI*/I^2C), para enviar y recibir datos a través de una red de dispositivos y sensores

Tabla A.7

Librerías compatibles con versiones de *Wiring*	
Matrix	Librería para manipular displays de matrices de LED básicas
Sprite	Librería básica para manipulación de *sprites* para usar en animaciones con matrices de *LED*

Tabla A.8

Librerías comunicación (*networking* y protocolos)	
Messenger	Para procesar mensajes de texto desde la computadora
NewSoftSerial	Versión mejorada de la librería *SoftwareSerial*
OneWire	Para controlar dispositivos (de *Dallas Semiconductor*) que usan el protocolo *One Wire*
PS2Keyboard	Para leer caracteres de un teclado PS2
Simple Message System	Para enviar mensajes entre Arduino® y la computadora
SSerial2Mobile	Para enviar mensajes de texto o e-mails mediante el uso de un teléfono móvil (vía comandos AT, a través de software serial)
Webduino	Librería de web server extendible; para usar con *Arduino Ethernet Shield*
X10	Para enviar señales X10 a través de líneas de corriente AC
XBee	Para comunicaciones entre *XBees* en modo *API*
SerialControl	Para controlar de manera remotaotras tarjetas Arduino® mediante una conexión serial

Tabla A.9

Librerías sensores	
Capacitive Sensing	Para convertir dos o más terminales en sensores capacitivos
Debounce	Para lectura de inputs digitales con ruido

Tabla A.10

Librerías displays y LED	
Improved LCD library	Mejora la inicialización del *LCD* de la librería *LCD* oficial de Arduino®
GLCD	Para graficar rutinas para *LCD* basados en el chipset *KS0108* o equivalentes
LedControl	Para controlar matrices de LED o displays de siete segmentos con *MAX7221* o *MAX7219*
LedControl	Librería alternativa a la librería *Matrix*; para controlar múltiples LED con chips *Maxim*
LedDisplay	Para control de una matriz de LED *HCMS-29xx*

Tabla A.11

Librerías generación de frecuencias y audio	
Tone	Para generar frecuencias de audio de onda cuadrada

Tabla A.12

Librerías motores y PWM	
TLC5940	Para el controlador de *PWM* de 16 canales y 12 *bits*

Tabla A.13

Librerías medición de tiempo	
DateTime	Para llevar el registro de fecha y hora actual en el software
Metro	Para cronometrar acciones en intervalos regulares
MsTimer2	Para utilizar *timer 2 interrupt*, con el fin de disparar una acción cada *N* milisegundos

Tabla A.14

Librerías utilidades	
TextString	También conocida como *String*; para manejar los *strings*
PString	Para imprimir en búfer
Streaming	Para simplificar declaraciones de impresión

Tabla A.15

Configuración de entradas y salidas

Terminales digitales

En esta sección se aborda el funcionamiento y la configuración de las terminales de la tarjeta Arduino® en los modos de entradas o salidas.

Propiedades de las terminales configuradas como entrada INPUT

De fabricación, las terminales de Arduino® (ATMEGA) son de entrada, por lo que no es necesario configurarlas explícitamente como entradas. Sin embargo, es sabido que las terminales configuradas como entradas están en estado de alta impedancia; esto es, cuando las terminales de entrada hacen demandas extremadamente pequeñas en el circuito que están muestreando, se dice que equivale a una resistencia en serie de 100 MΩ frente a la terminal. Lo que significa que para cambiar de un estado a otro la terminal de entrada, se requiere muy poca corriente. Y puede hacer posible el uso de estas terminales para tareas como la utilización de un *sensor capacitivo al tacto*, la lectura de un LED como un *fotodiodo* o la lectura de un sensor analógico con un esquema como el *RCTime*.

Cuando las terminales de entrada están libres y sin conexión hacia otros circuitos, estas reflejan cambios en apariencia aleatorios en el estado de la terminal, recogiendo el ruido eléctrico del entorno o el acoplamiento capacitivo del estado de una terminal próxima.

Resistencias Pull-up y Pull-down

A menudo, las resistencias Pull-up se utilizan para colocar las terminales de entrada a un estado conocido cuando no existe un estado de entrada. Esto se puede hacer al añadir una resistencia Pull-up (a +5 V) o una resistencia Pull-down (resistencia referenciada a tierra) en la entrada; 10 KΩ suele ser el valor más común.

Pero, también hay resistencias Pull-up de 20 KΩ, convenientemente integradas en el chip ATMEGA, a las que se puede acceder desde el software. El acceso a las resistencias Pull-up incorporadas se realiza conforme lo que se observa a continuación.

```
pinMode(terminal, INPUT);     // Pone la terminal como
       // entrada
digitalWrite(terminal, HIGH);     // Activa la resistencia
      // pull-up
```

Nota

La terminal digital 13 es más difícil de usar que otras terminales digitales porque esta tiene un LED y una resistencia asociada soldados a la tarjeta. En el caso de esta terminal, si la resistencia Pull-up 20 KΩ del interior se activa, se situará alrededor de 1.7 V en lugar de los 5 V que se esperan, debido a que el LED integrado y la resistencia en serie bajan el nivel del voltaje, lo que se traduce en el hecho de que siempre retornará a LOW. Sin embargo, resulta imprescindible utilizar la terminal 13 como entrada digital, para lo cual es recomendable usar una resistencia Pull-down externa.

Las resistencias Pull-up proporcionan suficiente corriente para generar una tenue luz visible cuando se usa un LED conectado a una terminal que se ha configurado como entrada. También es posible usar pinMode( ) para ajustar las terminales como salidas.

Hay tener en cuenta que las resistencias Pull-up son controladas por los mismos registros (posiciones de memoria interna del chip) que controlan si una terminal está en alto, HIGH, o en bajo, LOW. Por consiguiente, si una terminal que se configura para tener las resistencias Pull-up activas cuando está dispuesta como entrada, esta debe tener la terminal a alto HIGH, si la terminal es cambiada como salida OUTPUT, con la función pinMode(). Pero, esto también funciona en la otra dirección; esto es, una terminal de salida que queda en un estado alto, HIGH, tendrá las resistencias Pull-up activas si cambia a entrada INPUT con la función pinMode().

Propiedades de las terminales configuradas como salida OUTPUT

Las terminales configuradas como salida OUTPUT, con la función pinMode(), se encuentran en un estado de baja impedancia. Esto significa que estas terminales pueden proporcionar una cantidad sustancial de corriente a otros circuitos. Las terminales del ATMEGA están capacitadas para proporcionar corriente positiva o corriente negativa de hasta 40 mA a otros dispositivos o circuitos, que es suficiente para activar un LED (no olvidar la resistencia en serie) o para utilizarse con algunos sensores; sin embargo, esta corriente no es suficiente para activar de manera adecuada la mayoría de relés, solenoides o motores.

Los cortocircuitos en las terminales de Arduino® o la extracción de alta corriente desde estas, pueden dañar o destruir los transistores de salida en la terminal e inclusive dañar por completo el chip ATMEGA. A menudo, estas circunstancias se traducen en que una o varias de las terminales del microcontrolador estén en "estado muerto"; no obstante, en algunas ocasiones, el resto del chip puede seguir funcionando de manera adecuada. Por esta razón, se deben conectar las terminales de salida a otros dispositivos con resistencias de 470 Ω o 1 KΩ, limitando la corriente máxima que es requerida para una aplicación particular desde las terminales.

Terminales analógicas

Convertidor analógico–digital

El controlador ATMEGA que usa Arduino® incluye un convertidor analógico-digital de 6 canales y tiene una resolución de 10 bits, retornando enteros desde 0 a 1 023. Estas terminales se usan en especial para la lectura de sensores analógicos; además, tienen toda la funcionalidad de las terminales de entrada-salida de propósito general GPIO.

En consecuencia, si se requieren más terminales de propósito general de entrada-salida, y no se está usando ninguna terminal analógica, estas terminales pueden usarse como GPIO.

Mapeo de terminales

Las terminales analógicas de Arduino® son de la A0 hasta la A5. Como puede suponerse, estas son terminales exclusivas de Arduino®, debido a que estas no corresponden a los números de las terminales del chip ATMEGA328. Es importante resaltar que las terminales analógicas se usan de manera idéntica que las digitales; así, que, por ejemplo, se puede tener un código como el que se muestra a continuación para configurar una terminal analógica y establecerlo a HIGH.

```
pinMode(A1, OUTPUT);
digitalWrite(A1, HIGH);
```

Resistencias Pull-up

Las terminales analógicas también hacen uso de las resistencias Pull-up, las cuales funcionan de forma similar que en las terminales digitales; se activan cuando se usan instrucciones como la siguiente.

```
digitalWrite(A0, HIGH);  // activa la resistencia  pull-up en la
     //  terminal analogica 0
```

Mientras la terminal está configurada como entrada INPUT, es importante considerar que si se activa una resistencia Pull-up, cuando se usan algunas clases de sensores, los valores obtenidos por la función **analogRead()** se verán afectados. Por otra parte, es necesario resaltar que el comando **analogRead** no funciona de manera adecuada si la terminal analógica está establecida como salida; si este es el caso, habrá que volver a establecerla como entrada y, posteriormente, usar **analogRead**. De igual manera, si la terminal está establecida como de salida, y se estableció como HIGH, la resistencia Pull-up permanecerá activa, aunque la terminal se vuelva establecer como de entrada.

Operadores

Operadores aritméticos

Operador de asignación

Debido a que el operador de asignación utiliza el signo de igualdad "=", este provoca que el microcontrolador de la tarjeta Arduino® evalué el valor o la expresión en el lado derecho del signo "=" y lo almacene en la variable que se encuentra en el lado izquierdo. Es importante destacar que esta variable tiene que ser capaz de mantener el valor almacenado en esta, por lo que si no es lo suficientemente grande para contenerlo, dicho valor almacenado en la variable será incorrecto.

Ejemplo

```
int variable;    // Declara una variable.
variable=10;     // Almacena el valor de 10 en variable.
```

Operadores de suma, resta, multiplicación y división

Para representar los operadores de suma, diferencia, producto y cociente se hace uso de los signos o símbolos matemáticos "+", "–", "*", "/", respectivamente. Estos operadores devuelven los resultados de dichas operaciones, las cuales se realizan mediante el uso de los tipos de datos de cada uno de los operandos, esto implica que la operación puede desbordarse si el resultado es mayor que el que se puede almacenar en el tipo de datos. Por otra parte, si los operandos son de clases diferentes, se utiliza la clase del "más grande" para realizar el cálculo. De este modo, si uno de los operandos es del tipo **float** o del tipo **double**, se usa punto flotante para el cálculo.

Ejemplos

```
a = a + 2;
b = b - 2;
c = c * 2;
d = d / 2;
```

Sintaxis

```
resultado = valor1 + valor2;
resultado = valor1 - valor2;
resultado = valor1 * valor2;
resultado = valor1 / valor2;

valor 1 - Cualquier variable o constante.
valor 2 - Cualquier variable o constante.
```

Operador módulo

El operador módulo está representado por el símbolo "%"; su función principal consiste en indicar al microcontrolador de la tarjeta Arduino® que calcule el residuo de la división entre dos enteros.

Ejemplo

```
residuo = 2 % 3;
```

Sintaxis

```
resultado = dividendo % divisor
dividendo: El número que se va a dividir.
divisor: El número que va a dividir.
```

Operadores condicionales y de comparación

Para representar los operadores condicionales y de comparación se usan los símbolos siguientes: "if", "==", "!=", "<", ">"; así como algunas de sus combinaciones: "<=" y ">=". En general, el operador de condicional "if" se usa en conjunto con uno o más operadores de comparación, como: "==", "!=", "<", ">", "<=" y ">=", para verificar si la condición en cuestión es verdadera. El formato más usado de este tipo de operadores es el que se muestra a continuación.

```
if (Variable > 40)
{
  // codigo del programa.
}
```

En el código anterior, se verifica si "Variable" es mayor a 40; en caso de que esta condicións sea verdadera, se ejecutan las líneas "código del programa", que se encuentran después de la instrucción if, dentro de llaves: "{ }". En este caso, el operador de comparación dentro del paréntesis es: ">"; sin embargo, pudo haberse usado otro o algunas combinaciones de otros, como se muestra en la siguiente lista.

```
x == 40 (x es igual a 40)
x != 40 (x no es igual a 40)
x <  40 (x es menor a 40)
x <= 40 (x es menor o igual a 40)
x >= 40 (x es mayor o igual a 40)
```

Operadores Booleanos

Se pueden usar dentro de operaciones condicionales o en una sentencia if.

Operador AND

Se representa con el símbolo "&&". Este operador realiza una operación de AND lógica; es decir, es verdadera si ambos operandos son verdaderos. Por ejemplo, en la línea siguiente el resultado es verdadero, ya que ambas entradas están presentes.

```
if (Variable(2) == 1 && Variable(3) == 1) { }
```

Se representa con el símbolo "||". Este operador realiza una operación de OR lógico; es decir, devuelve un valor verdadero si alguno de los operandos es verdadero. Por ejemplo, en la siguiente línea de código el resultado es verdadero si alguno de los valores de "x" o "y" es mayor que cero.

```
if (x > 0 || y > 0) { ... }
```

Operador NOT

Se representa con el símbolo "!". Se utiliza para negar el resultado en cuestión; es decir, devuelve un valor verdadero si el valor de la variable es falso.

```
if (!x) { ... }
```

Operadores de composición

Operadores de incremento y disminución

Se representan con los símbolos "++" y "—", respectivamente. Se utilizan para incrementar y disminuir, de manera respectiva, el valor de una variable del tipo entero o long.

Ejemplo

```
x = 2;
y = ++x;       // x ahora guarda 3, y guarda 3
y = x--;      // x guarda 2 de nuevo, y sigue guardando 3
```

Sintaxis

```
x++;     // incrementa x en uno y regresa el valor anterior de x
++x;    // incrementa x en uno y regresa el nuevo valor de x
x-- ;   // disminuye x en uno y regresa el valor anterior de x
--x ;  // disminuye x en uno y regresa el nuevo valor de x
```

Operadores de composición suma, composición resta, composición multiplicación y composición división

Para representar este tipo de operaciones de composición se hace uso de los símbolos matemáticos: "+=", "-=", "*=", "/=", respectivamente. Estos operadores son una forma simplificada de la sintaxis completa, como se muestra en la lista siguiente.

Ejemplo

```
x = 2;
x += 4;      // x ahora es 6
x -= 3;      // x ahora es 3
x *= 10;     // x ahora es 30
x /= 2;      // x ahora es 15
```

Sintaxis

```
x += y;   // equivalente a la expresión x = x + y;
x -= y;   // equivalente a la expresión x = x - y;
x *= y;   // equivalente a la expresión x = x * y;
x /= y;   // equivalente a la expresión x = x / y;
```

Estructuras de control

Estructura If/else

Como se dijo antes, en la sección de operadores, el operador "if" se considera un operador condicional; sin embargo, también puede ser parte de una estructura de control con ramificaciones cuando se usa en conjunto con el operador "else".

El operador conjunto resultante **if/else** permite control sobre el sentido del flujo de las líneas de código, en comparación con el operador *if* básico, ya que permite agrupar múltiples bloques de código para su evaluación. Por ejemplo, véanse las siguientes líneas de código.

```
if (magnitud < 10)
{
  // acción A
```

```
}
else
{
// acción B
}
```

Como se puede observar en las líneas de código anteriores, "else" realiza una evaluación de la instrucción if; de esta manera, es posible encadenar múltiples evaluaciones en una sola estructura de condiciones. En este caso, para cada bloque de código se hace una evaluación y se continúa con el siguiente bloque, solo cuando su propio resultado sea falso; por el contrario, si el resultado es verdadero, el bloque de código actual se ejecuta y el programa evita las siguientes evaluaciones. Si ninguna evaluación devuelve un valor verdadero, entonces "else" será ejecutado.

```
if (magnitud < 10)
{
  // ejecutar A
}
else if (magnitud >= 20)
{
// ejecutar B
}
else
{
  // ejecutar C
}
```

Otra forma de expresar ramificaciones y realizar comprobaciones mutuamente exclusivas es con la declaración **switch case**; en Arduino®, esta instrucción trabaja de forma semejante a las estructuras en lenguaje C/C++. Debido al enfoque y a los objetivos de este libro, no se profundizarán aquí temas más especializados del área de programación; para mayor información, se recomienda revisar documentación referente en libros dedicados a este tema.

Estructura for

La declaración **for** es una de las declaraciones más usada para repetir un bloque de código. Un incremento de un contador es usado, por lo normal, para aumentar y terminar con el ciclo. La estructura **for** resuelve la mayoría de las necesidades donde se requieren operaciones repetitivas, e incluso se usa para operaciones con vectores, que operan sobre conjuntos de datos y terminales.

El ciclo **for** tiene tres partes o argumentos en su inicialización

for (**inicialización**; **condición**; **incremento**) {//función(es);}

La **inicialización** se produce solo la primera vez. Por tanto, cada vez que inicia la repetición de un ciclo, se revisa la **condición**, si esta es cierta, el bloque de funciones y el **incremento** del contador se ejecutan y la condición es evaluada de nueva cuenta. Por otro lado, si la condición es falsa, el ciclo termina.

El ciclo **for** en Arduino® (C/C++) es más flexible, en comparación con otros lenguajes, incluyendo BASIC. Aquí, cualquiera o todos los parámetros pueden ser omitidos, sin embargo los puntos y coma ";" son obligatorios. La inicialización, condición e incremento también pueden ser cualquier declaración válida en C, con variables independientes. Asimismo, se puede usar cualquier tipo de variable, incluidos los float, ya que estos tipos de declaración **for**, a pesar de que son poco usados, pueden proporcionar una solución válida a algunos problemas de programación.

Ejemplo

```
   for (int i=0; i <= 255; i++){
salida(5, i);
delay(10);
   }
```

Por ejemplo, al usar la multiplicación en el parámetro de incremento, podemos generar una progresión logarítmica.

```
for(int x = 2; x < 100; x = x * 1.5){
println(x);
}
```

Este código generará: 2,3,4,6,9,13,19,28,42,63,94.

Estructura while

En el ciclo **while**, los bloques de código se ejecutan de manera continua, hasta que la evaluación de la expresión dentro del paréntesis, (), resulta falsa. Así que algo debe modificar la variable comprobada, de lo contrario el ciclo **while** nunca terminará. Lo que modifique la variable puede estar en el código, como una variable que se incrementa o ser una condición externa, como el valor que da un sensor.

Ejemplo

```
var = 0;
while(var < 200){
  // repetitivo 200 veces
  var++;
}
```

Sintaxis

```
while(expresion){...}

expresion - una sentencia C (booleana) que se evalúa para generar un valor verdadero o
falso
```

Estructura do – while

El ciclo "'do'" trabaja de la misma manera que el ciclo "'while'", a excepción de que la condición se comprueba al final del ciclo, por lo que este se ejecuta "siempre", al menos una vez.

```
do
{
                                // bloque de instrucciones
} while (condición);

Ejemplo:
do
{
  delay(50);                    //  espera a que los sensores se
        //   estabilicen
  x = readSensors();           //   comprueba los sensores

} while (x < 100);        //   si se cumple la condición se repite el ciclo
```

Estructura break

Break se usa para salir de los ciclos do, for o while, pasando por alto la condición normal del ciclo. Además, también se usa para salir de una estructura de control switch.

Ejemplo

```
for (x = 0; x < 255; x ++)
{
digitalWrite(PWMpin, x);
    sens = analogRead(sensorpin);
```

```
    if (sens > threshold){              //
                          // detect
x = 0;
       break;                          //  sale del ciclo for.
    }
    delay(50);
}
```

Estructura continue

La sentencia **continue** omite el resto de iteracciones de un ciclo (**do**, **for** o **while**). Esto es, continúa saltando a la condición de ciclo y procediendo con la siguiente iteración.

Ejemplo

```
for (x = 0; x < 255; x ++)
{
    if (x > 40 && x < 120){      // crea un salto en estos
      // valores
        continue;
    }

    digitalWrite(PWMpin, x);
    delay(50);
}
```

Estructura return

Se utiliza principalmente para terminar una función y devolver un valor a la función que la llama; sin embargo, puede no devolver nada.

Ejemplo

Una función que compara la entrada de un sensor a un umbral:

```
int comprobarSensor(){
    if (analogRead(0) > 400) {
        return 1;
else{
        return 0;
}
}
```

La palabra clave **return** es útil para depurar una sección de código sin tener que comentar una gran cantidad de líneas de código posiblemente incorrecto.

```
void loop(){

// código magnifico a comprobar aqui

return;

// el resto del programa del que se desconfía
// que nunca sera ejecutado por estar detras de return
}
```

Sintaxis

```
return;

return valor; // ambas formas son correctas

valor: cualquier variable o tipo constante
```

Ejercicios propuestos

Actividad 1

Instalación del entorno de desarrollo y drivers

Descargue e instale el entorno de desarrollo de Arduino® llamado: *Windows Installer*, que se ubica en la siguiente dirección de Internet: *http://arduino.cc/en/Main/Software*. Enseguida, instale los controladores (solo conecte la tarjeta Arduino® a su computadora mediante un cable USB, para el caso de las versiones Windows Vista, 7 u 8). Acto seguido, se muestra la leyenda: "Listo para usarse", si el proceso se realizó de manera correcta.

Actividad 2

Carga a la tarjeta Arduino® con código de programa

Ejecute el software de Arduino® instalado en la actividad 1; entonces, se desplegará el ambiente de programación Arduino®, y cargue el código denominado *Blink*, que se encuentra en el menú Archivo>Ejemplos>Basics. Observe cómo el **led** de la terminal 13 trabaja en forma intermitente, con un tiempo de encendido y apagado de un segundo.

Actividad 3

Uso del led de la terminal 13

Con base en el ejemplo anterior, modifique y cargue el código de tal manera que el **led** trabaje de forma intermitente a intervalos de 5 segundos.

Actividad 4

Led intermitente

Con base en el código de la actividad 2, modifique y cargue el programa de tal manera que el **led** de la terminal 13 trabaje de forma intermitente con un intervalo de encendido de 3 segundos y de apagado de 1 segundo.

Actividad 5

Lectura e impresión de señal analógica

Suponga que existe una señal analógica conectada a la terminal A2. Desarrolle un programa para que lea esta señal e imprima, cada segundo, los valores de la lectura en pantalla. Apóyese en el ejemplo AnalogReadSerial que se encuentra en el menú Archivo>Ejemplos>Basics.

Actividad 6

Configuración de salidas

Configure como salida las terminales 7, 8, 9, 10 y 11 de la tarjeta Arduino®.

Actividad 7

Secuencia de encendido y apagado

Con base en la actividad 6, genere una secuencia básica de encendido y apagado de leds (corrimiento) en las terminales configuradas como salidas, con un tiempo de encendido y apagado de 3 segundos, para cada uno de estos. No olvide conectar una resistencia adecuada (220 Ω) a las terminales de salida.