



Arduino. Nivel I

octubre, 2019

Objetivos del nivel

- Conocer el entorno de desarrollo para Arduino
- Conocer los aspectos elementales del lenguaje de programación que se utiliza en Arduino: variables, condiciones, ciclos.
- Aprender a mostrar Inputs/Outputs (entradas y salidas) tanto analógicas como digitales

Prerrequisitos del nivel

- Lógica de Programación Nivel II
- Electrónica Nivel II

Acerca de este manual

Este manual pertenece al Centro de Asesoramiento y Desarrollo Informático C.A. (CADI F1). Para obtener más información sobre este u otros cursos visite nuestro sitio Web www.cadif1.com, escribanos a la dirección de correo cadi@cadif1.com o visítenos en nuestra sede ubicada en la Av. Pedro León Torres con calle 59, Centro Comercial Sotavento, piso 2 oficina 27, Barquisimeto estado Lara, Venezuela. Tlf. 0251-7179247, 0251-4410268.

Las marcas mencionadas en este manual son propiedad de sus respectivos dueños. Copyright 2019. Todos los derechos reservados.

Contenido del nivel

Capítulo 1. Conociendo Arduino

- 1.1.- Microcontroladores.
- 1.2.- Arduino.
- 1.3.- Tarjetas Disponibles.

Capítulo 2. El Entorno de Desarrollo

- 2.1.- Entorno de Trabajo Web.
- 2.2.- Emulador.
- 2.3.- Entorno de Escritorio.
- 2.4.- Configuración Del Entorno de Escritorio.

Capítulo 3. Iniciando la Programación en Arduino

- 3.1.- Compilación y Transferencia.
- 3.2.- Estructura de un Programa Arduino.
- 3.3.- Comentarios.

Capítulo 4. Salidas. Parte 1

- 4.1.- Serial Monitor.
- 4.2.- Enviando Salidas.
- 4.3.- Retardos.
- 4.4.- Millis().

Capítulo 5. Salidas. Parte 2

- 5.1.- Configuración de Pines.
- 5.2.- Digital Output.
- 5.3.- Analog Output.

Capítulo 6. Variables

- 6.1.- Declaración de Variables.

- 6.2.- Asignaciones.
- 6.3.- Constantes.

Capítulo 7. Operadores Aritméticos

- 7.1.- Operadores Aritméticos.
- 7.2.- Operadores y Asignaciones.
- 7.3.- Números Aleatorios.

Capítulo 8. Condiciones. Parte 1

- 8.1.- Operadores Relacionales.
- 8.2.- If.
- 8.3.- If/else.

Capítulo 9. Entradas Por el Puerto Serial

- 9.1.- Serial.available().
- 9.2.- Serial.read().
- 9.3.- Serial.readString().
- 9.4.- Serial.find().
- 9.5.- Serial.findUntil().
- 9.6.- Serial.parseInt().
- 9.7.- Serial.parseFloat().

Capítulo 10. Entradas Digitales

- 10.1.- Introducción.
- 10.2.- Digital Read.

Capítulo 11. Entradas Analógicas

- 11.1.- Introducción.
- 11.2.- Analog Read.

Capítulo 12. Condiciones. Parte 2

- 12.1.- Operadores Lógicos.
- 12.2.- If/else If/else.
- 12.3.- Switch...case.

Capítulo 13. Operaciones Con String

- 13.1.- Métodos de String.
- 13.2.- Método de conversión.
- 13.3.- Métodos Char.

Capítulo 14. Ciclos. Parte 1

- 14.1.- For.

Capítulo 15. Ciclos. Parte 2

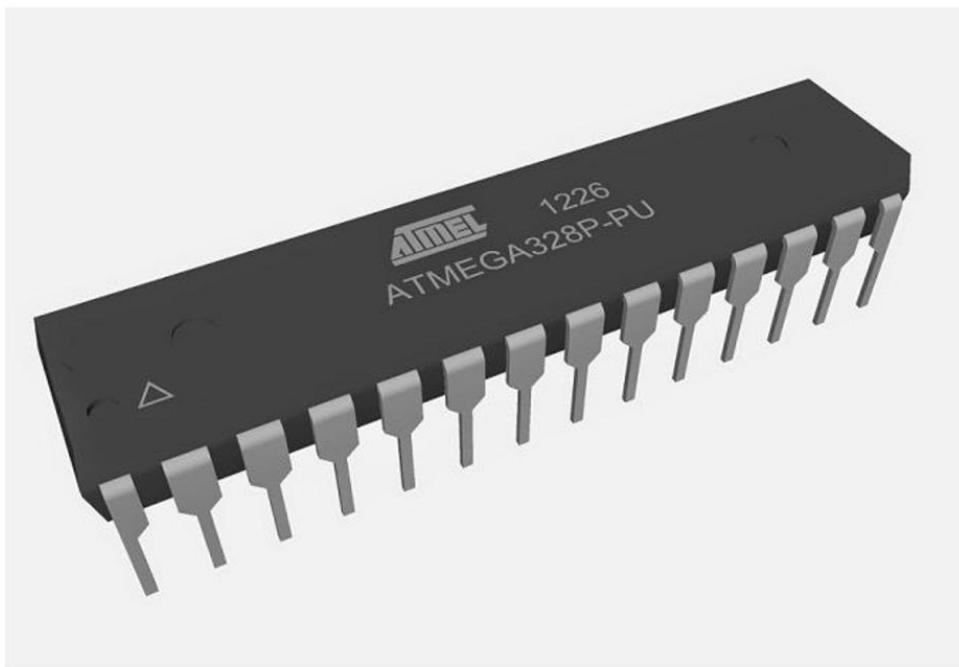
- 15.1.- While.
- 15.2.- Do...while.

Capítulo 1. CONOCIENDO ARDUINO

1.1.- Microcontroladores

Un microcontrolador es un circuito integrado digital que puede ser usado para muy diversos propósitos debido a que es programable. Está compuesto por una unidad central de proceso (CPU), memorias (ROM y RAM) y líneas de entrada y salida (periféricos).

Un microcontrolador tiene los mismos bloques de funcionamiento básicos de una computadora lo que permite tratarlo como un pequeño dispositivo de cómputo. Puede usarse para muchas aplicaciones algunas de ellas son: manejo de sensores, controladores, juegos, calculadoras, agendas, avisos lumínicos, secuenciador de luces, cerrojos electrónicos, control de motores, relojes, alarmas, robots, entre otros. El límite es la imaginación.



Como el hardware ya viene integrado en un solo chip, para usar un microcontrolador se debe especificar su funcionamiento por software a través de programas que indiquen las instrucciones que el microcontrolador debe realizar. En una memoria se guardan los programas y un elemento llamado CPU se encarga de procesar paso por paso las instrucciones del programa.

Los lenguajes de programación típicos que se usan para este fin son ensamblador y C, pero antes de grabar un programa al microcontrolador hay que compilarlo a hexadecimal que es el formato con el que funciona el microcontrolador.

Para diseñar programas es necesario conocer los bloques funcionales básicos del microcontrolador, estos bloques son:

- CPU (Unidad central de proceso)
- Memoria ROM (Memoria de solo lectura)
- Memoria RAM (Memoria de acceso aleatorio)
- Líneas de entrada y salida (Periféricos)

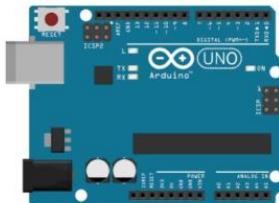
La CPU posee, de manera independiente, una memoria de acceso rápido para almacenar datos denominada registros, si estos registros son de 8 bits se dice que el microcontrolador es de 8 bits.

1.2.- Arduino

Es una plataforma de desarrollo basada en una placa electrónica de hardware libre que incorpora un microcontrolador re-programable y una serie de pines hembra, los que permiten establecer conexiones entre el microcontrolador y los diferentes sensores y actuadores de una manera muy sencilla.

Una placa electrónica es una PCB (“Printed Circuit Board”, “Placa de Circuito Impreso” en español). Las PCBs están fabricadas en un material no conductor, la cual consta de distintas capas de material conductor.

Una PCB es la forma más compacta y estable de construir un circuito electrónico. Así que la placa Arduino no es más que una PCB que implementa un determinado diseño de circuitería interna.



Entre las características más resaltantes de Arduino estan:

- es libre y extensible: Esto quiere decir que cualquiera que desee ampliar y mejorar el diseño hardware de las placas como el entorno de desarrollo, puede hacerlo sin problemas.
- posee una gran comunidad: Gracias a su gran alcance hay una gran comunidad trabajando con esta plataforma, lo cual genera una cantidad de documentación bastante extensa, la cual abarca casi cualquier necesidad.
- su entorno de programación es multiplataforma: Se puede instalar y ejecutar en sistemas operativos Windows, Mac OS y Linux.
- lenguaje de programación de fácil compresión: Su lenguaje de programación basado en C++ es de fácil compresión que permite una entrada sencilla a los nuevos programadores y a la vez con una capacidad tan grande, que los programadores más avanzados pueden exprimir todo el potencial de su lenguaje y adaptarlo a cualquier situación.

La enorme flexibilidad y el carácter libre y abierto de Arduino hacen que se pueda utilizar este tipo de placas prácticamente para cualquier cosa, desde relojes hasta básculas conectadas, pasando por robots, persianas controladas por voz y muchas otras cosas.

La respuesta es que se puede hacer y construir casi de todo. Arduino es una plataforma para programar un microcontrolador y por lo tanto puede hacer lo que se puede imaginar, todo depende de la imaginación.

Se llegó a construir incluso un robot BB8 a tamaño real con una placa Arduino y conectividad bluetooth para poder ser controlado desde un smartphone.



Entre otras aplicaciones que se le ha dado a las placas Arduino estan:

-Cámaras espías: con el uso de un módulo acelerómetro que detecte cuando se gire se podrá tomar una foto sin que la mayoría se dé cuenta, la cámara puede integrarse en un vaso o en cualquier lugar pequeño para pasar desapercibida.

- Control de tráfico: Se puede construir un semáforo inteligente con Arduino, no sólo mostrará las luces rojo, verde y amarillo para cada situación, sino que también incluye un sensor para ver un carro que se acerque al semáforo y que cambie la luz a verde en caso de que en el sentido contrario no vengan carros.



1.3.- Tarjetas Disponibles

Arduino cuenta con varios prototipos y modelos de placas a elegir, dependiendo de que tan grande sea el proyecto para el cual será usado. Entre los modelos más populares están:

- Arduino UNO.
- Arduino Nano.

- Arduino Leonardo.
- Arduino YUN.

Cada una de estas placas tienen características particulares que las diferencian de las demás.



Arduino UNO:

Es una placa basada en el microcontrolador ATmega328P. Tiene 14 pines de entrada/salida digital (de los cuales 6 pueden ser usados con PWM), 6 entradas analógicas, un cristal de 16Mhz, conexión USB, conector jack de alimentación, terminales para conexión ICSP y un botón de reseteo. Tiene toda la electrónica necesaria para que el microcontrolador opere, simplemente hay que conectar la energía por el puerto USB ó con un transformador AC-DC.

Características:

- Microcontrolador: ATmega328
- Voltaje de entrada (alimentación): 7-12V
- Pines: 14 pines digitales de I/O
- Entradas analógicas: 6
- Memoria flash: 32k
- Velocidad: 16MHz



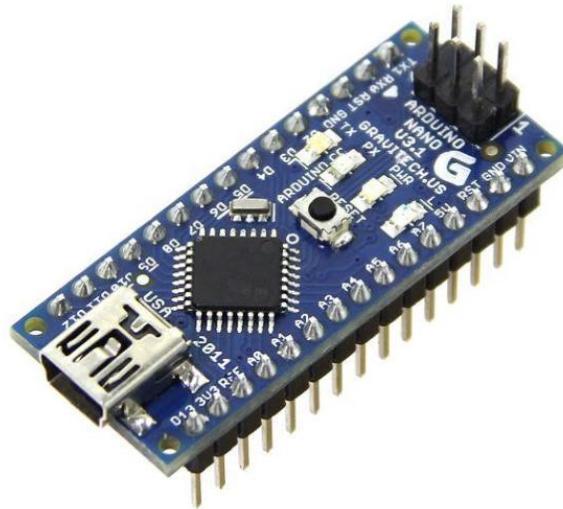
Arduino Nano:

Es una placa de desarrollo de tamaño compacto, completa y compatible con protoboards, basada en el microcontrolador ATmega328P. Tiene 14 pines de entrada/salida digital (de los cuales 6 pueden ser usando con PWM), 6 entradas analógicas, un cristal de 16Mhz, conexión Mini-USB, terminales para conexión ICSP y un botón de reseteo. Posee las mismas capacidades que un Arduino UNO, tanto en potencia del microcontrolador como en conectividad, solo se ve recortado en su conector USB, conector jack de alimentación y los pines cambia un formato de pines header.

Características:

- Microcontrolador: ATMega328
- Voltaje de operación: 5V
- Voltaje de entrada (alimentación): 7-12V

- I/O Digitales: 14 (6 son PWM)
- Memoria Flash: 32KB
- EEPROM: 1KB
- Velocidad: 16MHz



Arduino Leonardo:

Utiliza un microcontrolador ATmega32U4 que permite un diseño mucho más sencillo. Una de las ventajas de este microcontrolador es que dispone de USB nativo por hardware y por lo tanto no necesita de ninguna conversión serie-USB. También permite a la placa ser utilizada y programada como un dispositivo de entrada para emular un teclado, ratón, etc.

Características:

- Microcontrolador: ATmega32u4
- Voltaje de operación: 5V
- Voltaje de entrada (alimentación): 7-12V
- Pines I/O Digitales: 20
- Canales PWM: 7
- Entradas analógicas: 12

- Corriente Máxima de los pines I/O: 40 mA
- Corriente Máxima de los pines 3.3V: 50 mA
- Memoria Flash: 32 KB (4 KB usados para el bootloader)
- SRAM: 2.5 KB
- EEPROM interna: 1 KB
- Velocidad: 16 MHz



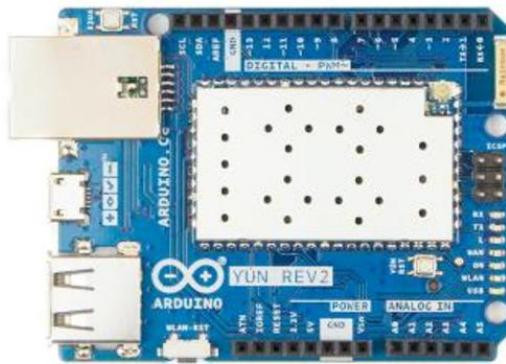
Arduino YUN:

Tiene un sistema basado en Linux que permite conexiones y aplicaciones de red avanzadas. Se puede conectar la placa a la red Wi-Fi o a una red cableada de una forma sencilla gracias al panel web de Yún. El panel web permite administrar la configuración de la placa y la carga de sketches.

Características:

- Procesador: Atheros AR9331
- Arquitectura: MIPS @400MHz
- Voltaje de entrada (alimentación): 3.3V
- Puerto Ethernet: IEEE 802.3 10/100Mbit/s
- Conexión WiFi: IEEE 802.11b/g/n
- USB Type-A: 2.0 Host/Device
- Lector de tarjetas: Micro-SD

- RAM: 64 MB DDR2
- Memoria Flash: 32 MB
- Soporte para PoE tipo 802.3af



Capítulo 2. EL ENTORNO DE DESARROLLO

2.1.- Entorno de Trabajo Web

Para trabajar en Arduino existen muchos entornos de trabajo con el cual puede ser manejado, entre ellos están:

- Arduino Software (IDE) (Desktop)
- Arduino Web
- Visual Studio Code
- Emulador TinkerCad

Todos los IDEs permiten hacer básicamente lo mismo:

- crear nuevos programas
- compilarlos
- subirlos a una placa compatible con Arduino

El Arduino Web Editor es una herramienta dispuesta en el sitio web oficial de Arduino en la dirección: arduino.cc. Permite escribir código y programar sketches (código fuente de un programa Arduino) a cualquier placa Arduino desde el navegador, ya sea (Chrome, Firefox, Safari o Edge). Es recomendable utilizar Google Chrome. Este IDE es parte de Arduino Create, una plataforma online que permite a los desarrolladores escribir código, acceder a tutoriales, configurar placas y compartir proyectos. Diseñado para proveer a los usuarios un flujo de trabajo continuo. El Arduino Web está alojado de manera online, como se muestra en la siguiente imagen:



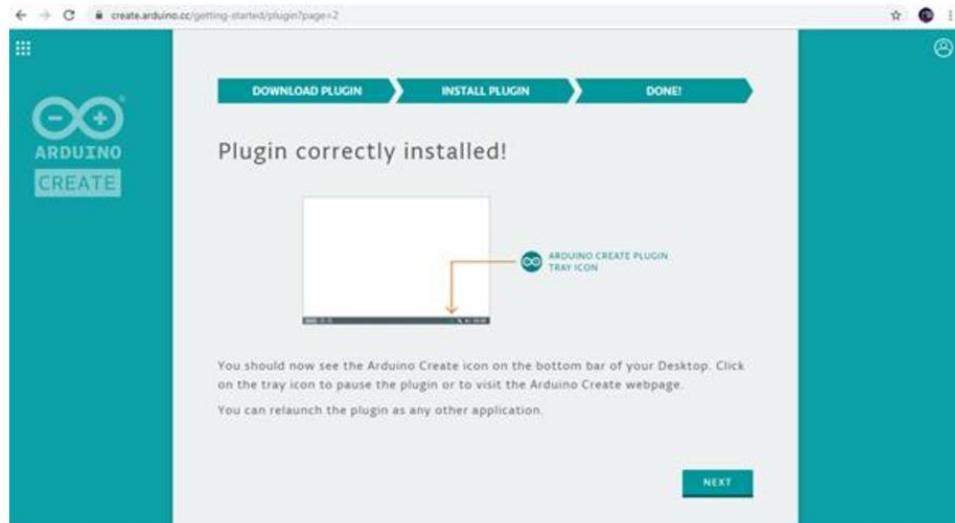
Al hacer clic en el botón "Code Online" se redirige a la dirección <https://create.arduino.cc/editor>. El primer paso, es crear una cuenta en la plataforma. Completado el formulario dar clic en el botón "CREATE ACCOUNT". (Activar la cuenta a través del email de activación).

A screenshot of the Arduino sign-up form. The top navigation bar includes links for HOME, STORE, SOFTWARE, EDUCATION, RESOURCES, COMMUNITY, and HELP. On the right side of the header is a 'SIGN IN' button. The main form is titled 'SIGN UP' and contains fields for 'Username' (with placeholder 'jose_valera'), 'Email' (with placeholder 'jomiiva5016@gmail.com'), 'Password' (with placeholder '*****'), and 'Confirm password' (with placeholder '*****'). Below these fields is a box containing five checkboxes, each preceded by a green circular icon with a checkmark. The checkboxes relate to privacy policy, newsletter consent, marketing offers, automated processing, and a reCAPTCHA verification. At the bottom left is a 'No soy un robot' checkbox next to a reCAPTCHA logo. On the bottom right is a large blue 'CREATE ACCOUNT' button.

Una vez que la cuenta sea creada exitosamente, se inicia sesión. El siguiente paso es descargar el plugin que se indica para poder instalar los drivers y poder programar las placas de Arduino.



Una vez el plugin sea detectado exitosamente dar “click” en el botón “next” y ya se podrá utilizar el editor.



El editor web cuenta con 3 grandes secciones:



La primera columna permite navegar entre:

- Tu Sketchbook: una colección de todos los sketches.

- Ejemplos: Sketches que demuestran todos los comandos básicos de Arduino.
- Librerías: Son paquetes que pueden ser incluidos al sketch para proveer funcionalidades extras.
- Monitor Serial: Permite recibir y enviar información a la placa a través del cable USB.
- Help: Son links de ayuda y un glosario sobre los términos y conceptos de Arduino.
- Preferencias: Es una opción que permite personalizar la vista y el comportamiento del editor, tal como el tamaño de la fuente y el color del tema.

Al seleccionar una de estas opciones se mostrarán en el side panel los ítems (segunda columna).

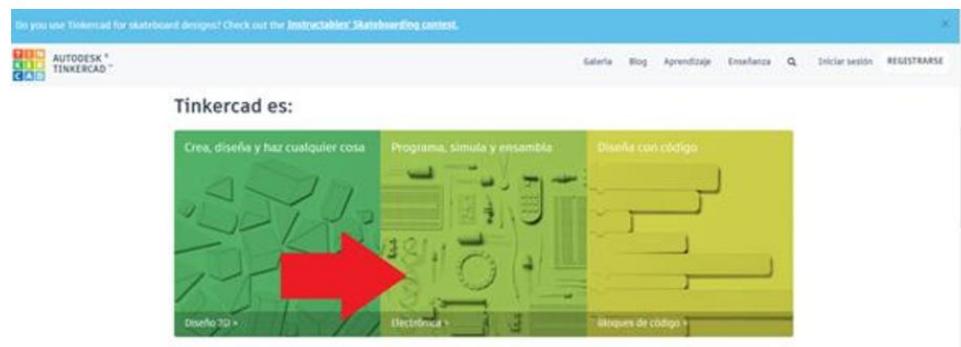
La tercera columna es el "Code area" donde se podrá escribir, verificar y programar código en las placas, guardar sketches en la nube y compartirlos.

2.2.- Emulador

Para trabajar con Arduino sin tener físicamente una placa se puede usar un emulador de software. Existen muchas opciones en el mercado, una de ellas es TinkerCad (<https://www.tinkercad.com>). Es una colección de software de herramientas online y gratuita que ayuda a la gente del mundo a pensar, crear y diseñar.



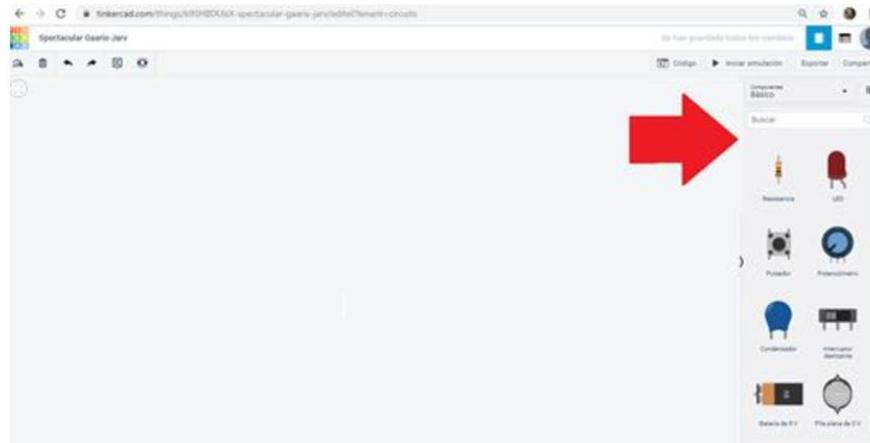
En la página de tinkerCad, se hace clic en la sección de electrónica para ingresar y utilizar tal herramienta.



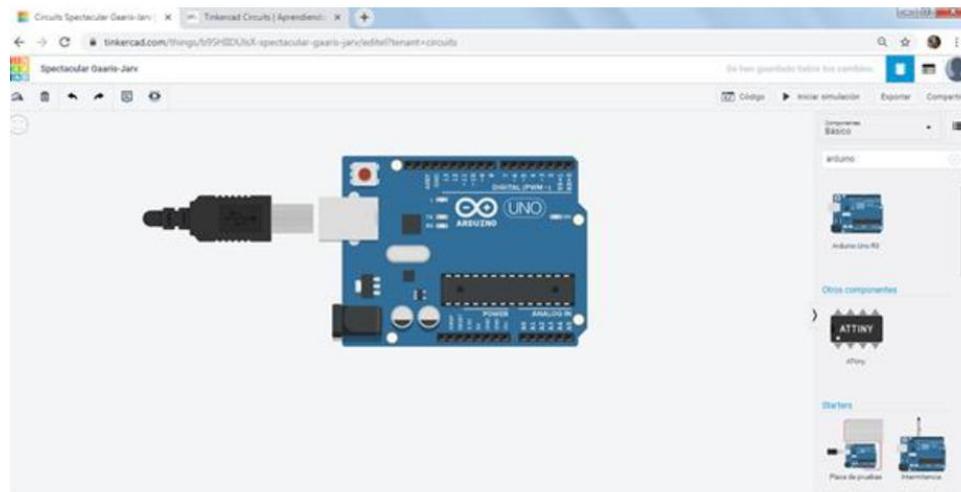
Luego, clic en el botón "Comenzar a utilizar tinkerCad", posteriormente indicará que se registre o se cree una cuenta, completar el registro e ingresar a “circuits” para empezar a diseñar un circuito.



Una vez completado los pasos anteriores, se mostrará un área de trabajo en la cual, del lado derecho habrá un panel con componentes electrónicos que se pueden arrastrar al área de trabajo en caso de que se quieran usar:



Entre los componentes, se busca el Arduino y se arrastra al área de trabajo para utilizarlo:



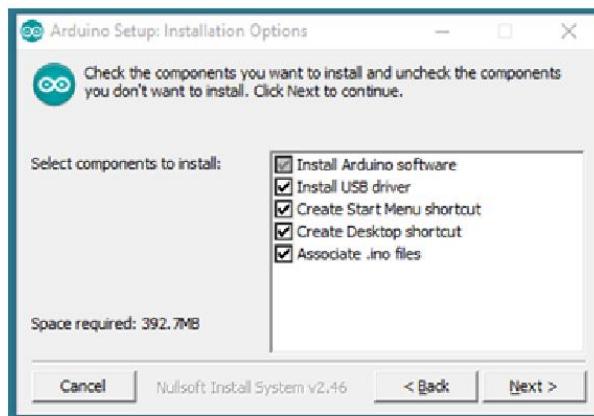
Una vez el Arduino se encuentre en el área de trabajo, en la parte superior derecha en “codigo” dar click en el panel, desplegar el select y cambiar el tipo de código de “Bloque” a “Texto”, se podrá visualizar el código por defecto que da tinkerCad que corresponde a un “Blink”. Iniciar simulación y visualizar como empieza a encender y apagar el led del Arduino.

The screenshot shows the Tinkercad simulation interface. At the top, there are tabs for "Código" (Code), "Iniciar simulación" (Start Simulation), "Exportar" (Export), and "Compartir" (Share). Below the tabs, there is a dropdown menu set to "Texto" (Text) and a toolbar with icons for download, file, and simulation. A dropdown menu for the board is set to "1 (Arduino Uno R3)". The main area contains the following Arduino code:

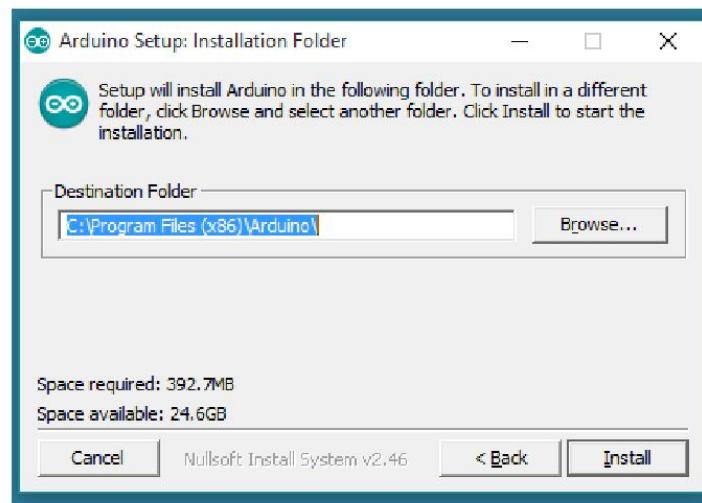
```
1 void setup()
2 {
3     pinMode(13, OUTPUT);
4 }
5
6 void loop()
7 {
8     digitalWrite(13, HIGH);
9     delay(1000); // Wait for 1000 millisecond(s)
10    digitalWrite(13, LOW);
11    delay(1000); // Wait for 1000 millisecond(s)
12 }
```

2.3.- Entorno de Escritorio

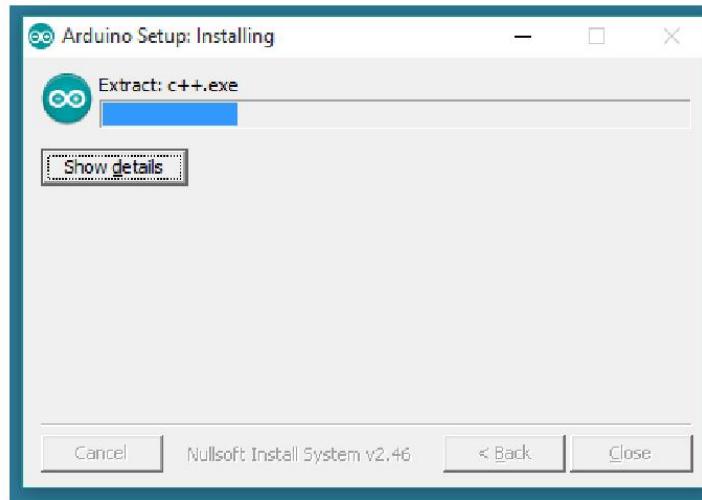
El IDE de escritorio de Arduino se puede descargar de la página oficial: arduino.cc. Hay versiones del IDE para Windows, Linux y MacOs. Está programado en Java y es distribuido bajo la licencia GPL. Cuando la descarga finalice, se procede con la instalación.



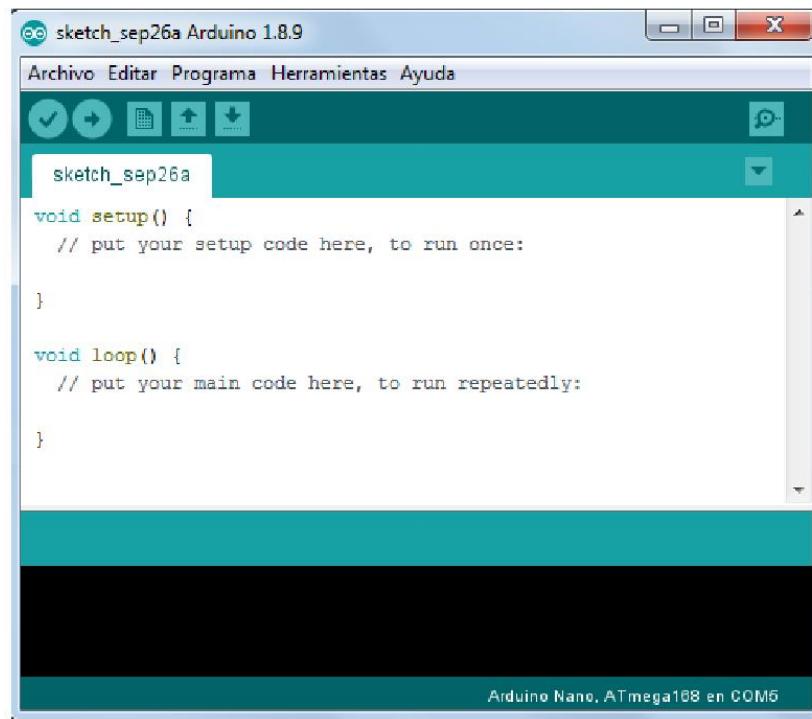
Seleccionar el directorio donde se desea instalar la aplicación:



El proceso extraerá e instalará todos los archivos requeridos para ejecutar apropiadamente el IDE. Al completarse la instalación se podrá acceder a la aplicación a través del acceso directo creado en el escritorio o a través del ejecutable dentro de la carpeta donde fue instalado Arduino IDE.



Luego de instalado, al abrir el IDE se puede visualizar como la siguiente imagen:

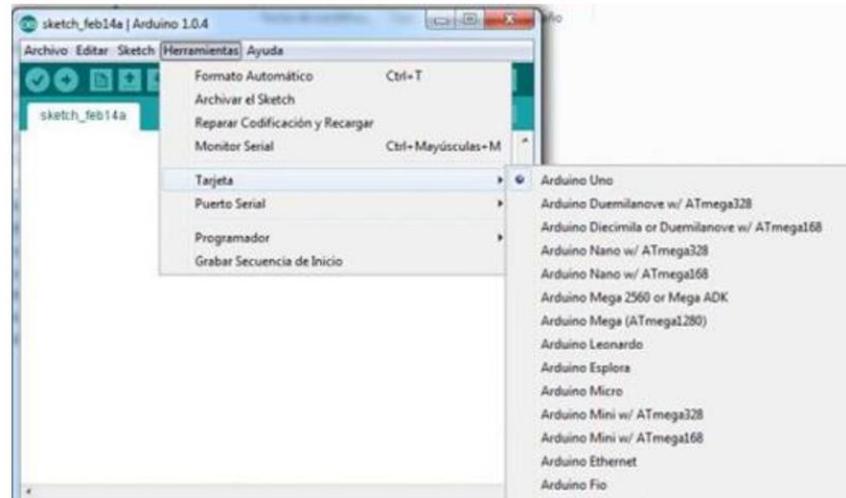


2.4.- Configuración Del Entorno de Escritorio

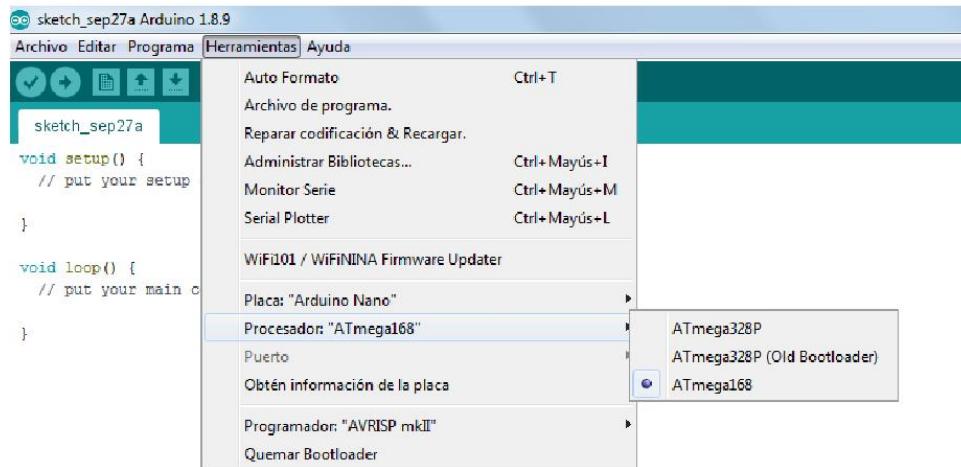
Antes de programar, se deben establecer 3 aspectos:

- Placa con la que se está trabajando.
- Procesador que contiene la placa.
- Puerto serial al cual está conectada la placa.

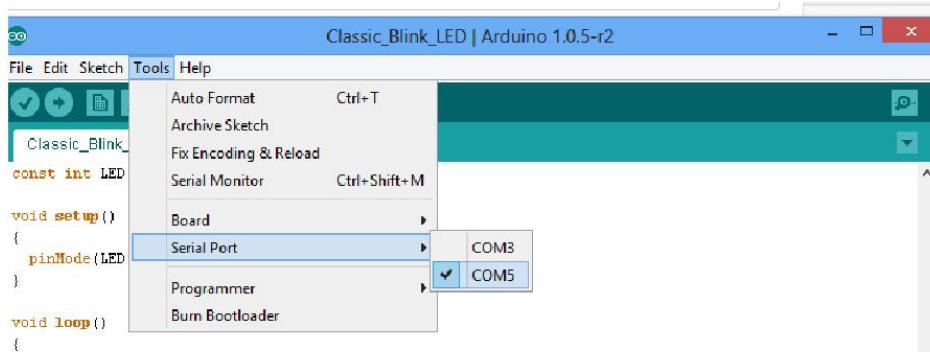
Todo esto se establece en la opción "Herramientas" del menú principal. Lo primero que debe seleccionarse es la placa con la que se trabajara, que dependerá del modelo que se tenga.



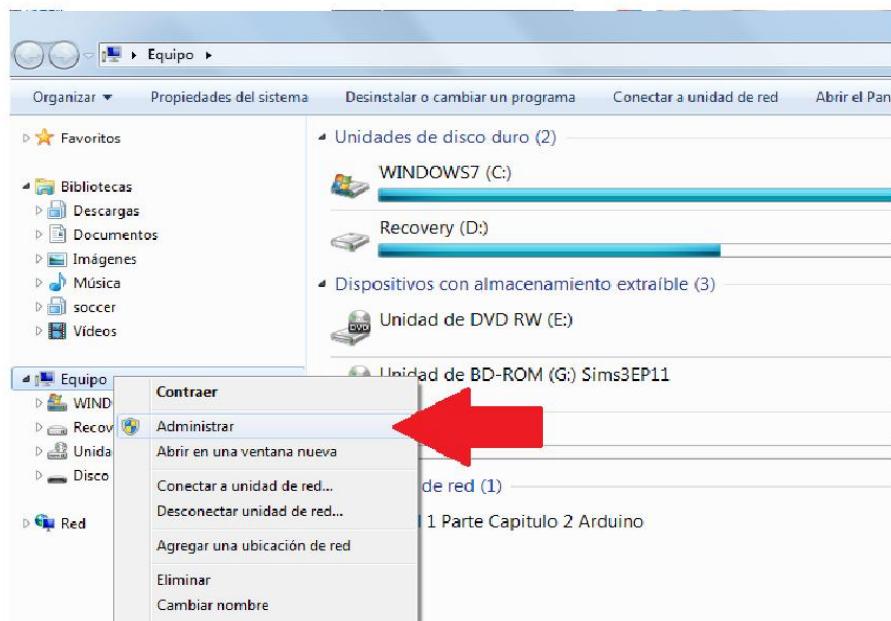
Seleccionar Procesador de la placa Arduino: dependiendo del microprocesador se debe seleccionar el que posee el Arduino. Este se puede visualizar en la placa del Arduino. Para algunos casos del ATmega328P se debe seleccionar el de Old Bootloader.



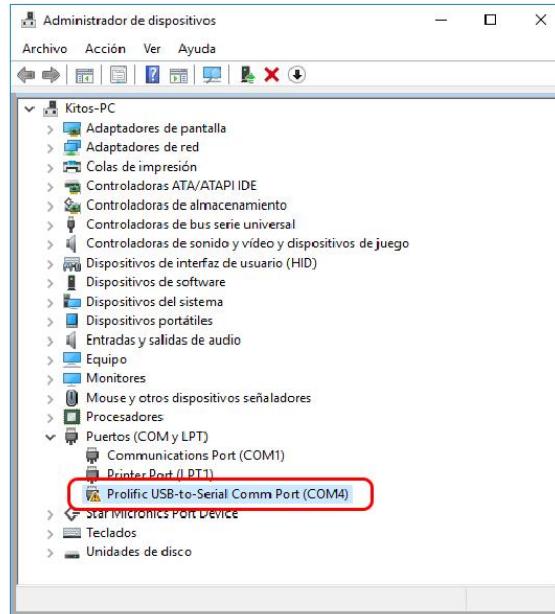
Normalmente el puerto serial (Serial Port) es detectado automáticamente, pero si hay conectados varios Arduinos al PC, habrá que seleccionar el puerto. El “puerto serial” se habilitará y mostrará el puerto que está siendo usado, en el cual está conectado el Arduino.



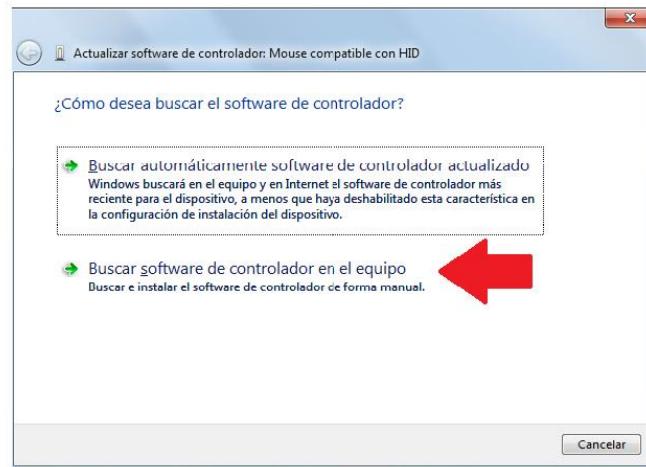
En caso de que el Arduino esté conectado pero el Serial Port no esté habilitado, un posible problema es no están bien instalados los Drivers USB. Para verificarlo, se debe abrir la herramienta de Administración de equipos de Windows realizando los siguientes pasos:



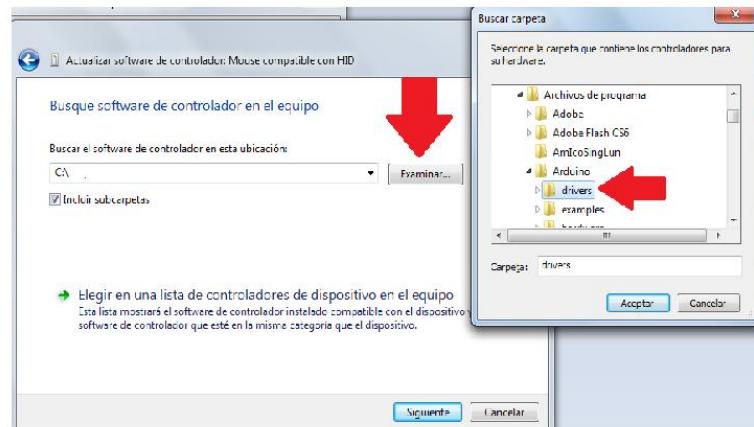
Dentro del administrador de equipos, dar “click” en “Administrador de dispositivos”. Dar Click derecho sobre el dispositivo no reconocido y “Propiedades”. Una vez allí en la pestaña “Controlador”, ir a “actualizar controlador”.



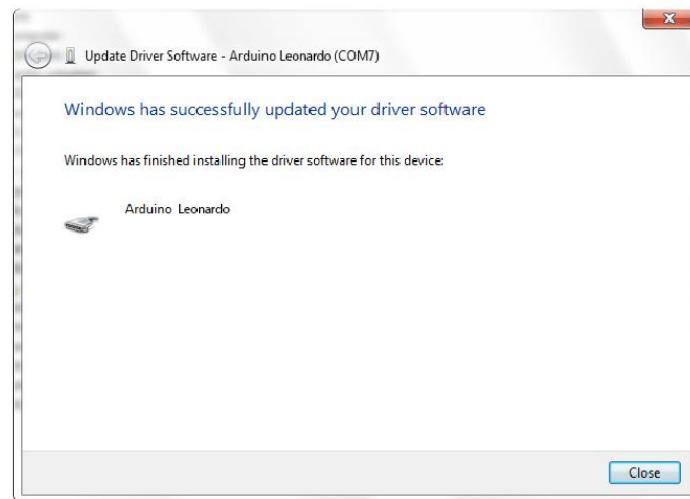
Escoger la segunda opción, “Buscar el software de controlador en el equipo”:



Una vez dentro, seleccionar "Examinar" y buscar la ruta de la carpeta "drivers" dentro de la carpeta donde se descargó la carpeta "Arduino". Aceptar y Siguiente. Después de haber instalado correctamente el driver, se habilitará la opción "serial port" en el IDE.



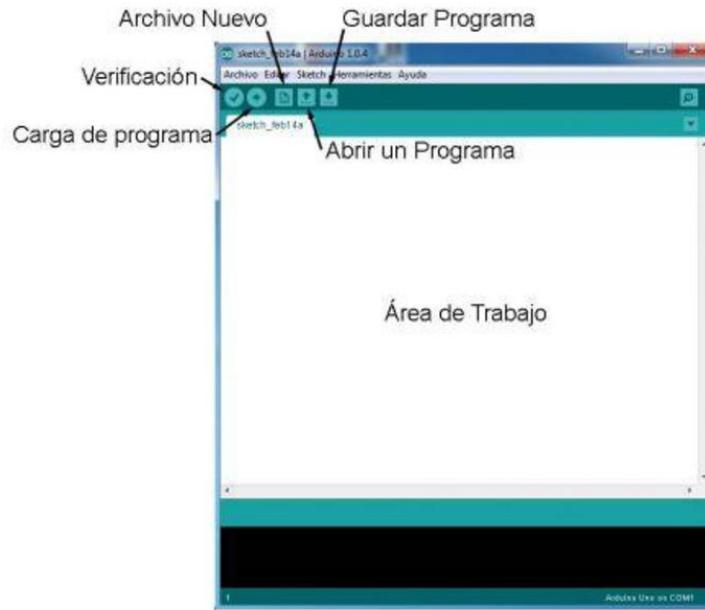
Después de haber instalado correctamente el driver, se habilitará la opción "serial port" en el IDE.



Capítulo 3. INICIANDO LA PROGRAMACIÓN EN ARDUINO

3.1.- Compilación y Transferencia

Al abrir el IDE de escritorio de Arduino se encuentran varias funciones básicas:



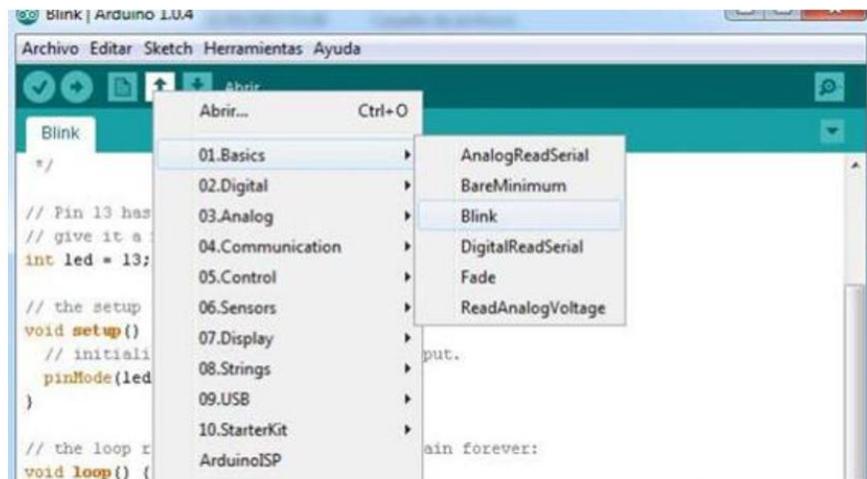
La descripción de cada función es la siguiente:

- Área de trabajo: Esta es el área donde se deberá escribir el programa que se quiere ejecutar en el Arduino.
- Verificación: Verifica que la sintaxis del código es correcta y que se ha cometido ningún error pulsando sobre este ícono, si hay algún error, no se cargará el programa en el Arduino.

- Carga de programa: Cuando esté listo el programa y no exista ningún error, se cargará el programa al Arduino pulsando sobre este botón.
- Archivo nuevo: Abrirá una nueva área de trabajo.
- Abrir un programa: Al pulsar sobre este botón, Se desplegará la opción de abrir un archivo desde una ubicación específica o cargar en nuestra área de trabajo una serie de programas o librerías ya creadas y que tiene Arduino por defecto.
- Guardar programa: Guardara en una ubicación especificada por el usuario el contenido del área de trabajo

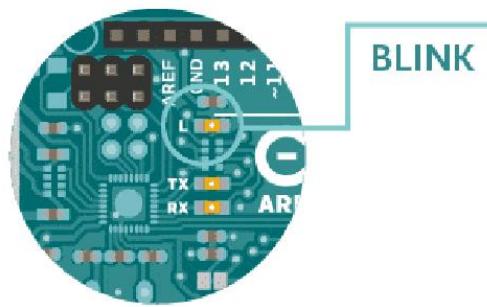
Como ejemplo y a modo de verificar que la placa funciona correctamente, se carga un programa que trae por defecto el software de Arduino que se llama "blink" (en español "parpadeo"). Este es un sencillo programa que lo único que va a hacer es poner a parpadear un led que está asociado al pin 13 del Arduino.

Para cargar este programa en el área de trabajo, pulsar sobre el icono "abrir" y se mostrará una lista con varios programas de ejemplo pre cargador que trae Arduino. Seleccionar el programa "Blink" que se encuentra en la sección "Basics".



Una vez abierto, se podrá ver la estructura básica del programa. Con el programa en el área de trabajo, se debe hacer clic en el botón "Cargar" para pasar el código del programa

al Arduino. Una vez cargado el programa, se podrá ver como el led asociado al pin 13, empieza a parpadear a intervalos de 1 segundo. Si este es el caso, la programación ha sido exitosa.



3.2.- Estructura de un Programa Arduino

La estructura básica de programación de Arduino es bastante simple y divide la ejecución en dos partes principales que por defecto son agregadas a cualquier programa:

- `setup()`: constituye la preparación del programa. Se trata de la primera función que se ejecuta al iniciar el programa. Esta función se ejecuta una única vez y es empleada para configurar todo lo necesario para la correcta ejecución del programa.
- `loop()`: incluye el código a ser ejecutado una y otra vez, dada una cantidad de microsegundos que depende de la velocidad del microcontrolador de la placa donde se esté ejecutando el programa (de ahí su nombre `loop` que significa bucle o ciclo).

Ejemplo de un programa básico es el siguiente:

The screenshot shows the Arduino IDE interface. The title bar reads "sketch_sep26a Arduino 1.8.9". The menu bar includes "Archivo", "Editar", "Programa", "Herramientas", and "Ayuda". Below the menu is a toolbar with icons for save, upload, and other functions. The main window displays the code for "sketch_sep26a":

```
void setup() {
    pinMode(pin, OUTPUT); //Establece 'pin' como salida
}

void loop() {
    digitalWrite(pin, HIGH); // Activa 'pin'
    delay(1000); // Pausa un segundo
    digitalWrite(pin, LOW); // Desactiva 'pin'
    delay(1000);
}
```

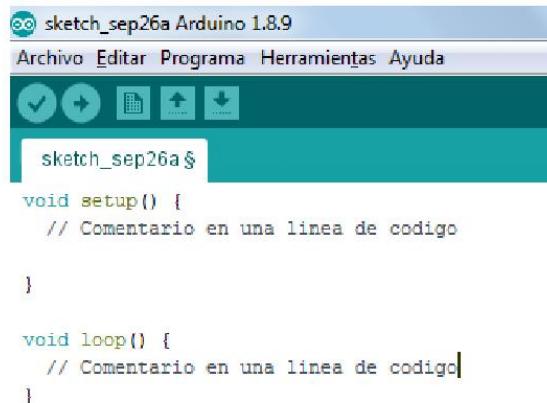
The status bar at the bottom right indicates "Arduino Nano, ATmega168 en COM5".

3.3.- Comentarios

Los comentarios son líneas en el programa que son ignoradas por el compilador y se utilizan para informar o documentar la forma en que funciona el programa. Al ser ignorados por el compilador, no se exportan al procesador, por lo que no ocupan ningún espacio. Hay 2 tipos de comentarios: de línea y de bloque.

- Comentarios de línea:

Para hacer un comentario en una línea de código se debe utilizar // (doble slash) antes del texto que se desea comentar. Todo contenido después de las barras dentro de la misma línea será ignorado por el compilador.



The screenshot shows the Arduino IDE interface with the title bar "sketch_sep26a Arduino 1.8.9". The menu bar includes "Archivo", "Editar", "Programa", "Herramientas", and "Ayuda". Below the menu is a toolbar with icons for save, upload, and other functions. The code editor window contains the following code:

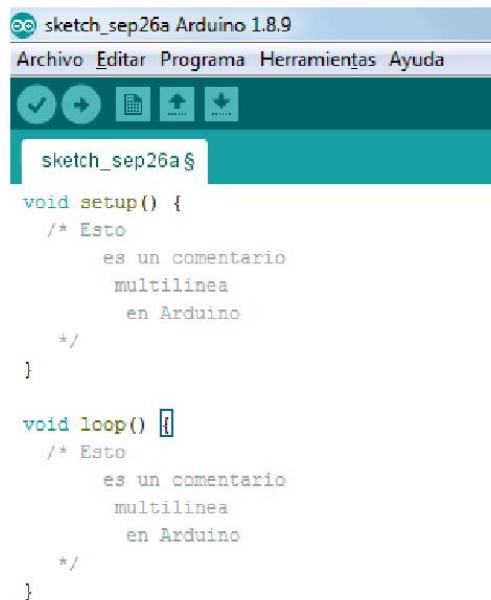
```
void setup() {
    // Comentario en una linea de codigo

}

void loop() {
    // Comentario en una linea de codigo
}
```

- Comentarios múltiples líneas:

Para hacer un comentario de múltiples líneas se debe encerrar entre los caracteres `/* */` las líneas de código que se desean comentar. Es necesario cerrar el comentario con `*/` porque si no todas las líneas siguientes del código serán tomadas como comentario.



The screenshot shows the Arduino IDE interface with the title bar "sketch_sep26a Arduino 1.8.9". The menu bar includes "Archivo", "Editar", "Programa", "Herramientas", and "Ayuda". Below the menu is a toolbar with icons for save, upload, and other functions. The code editor window contains the following code:

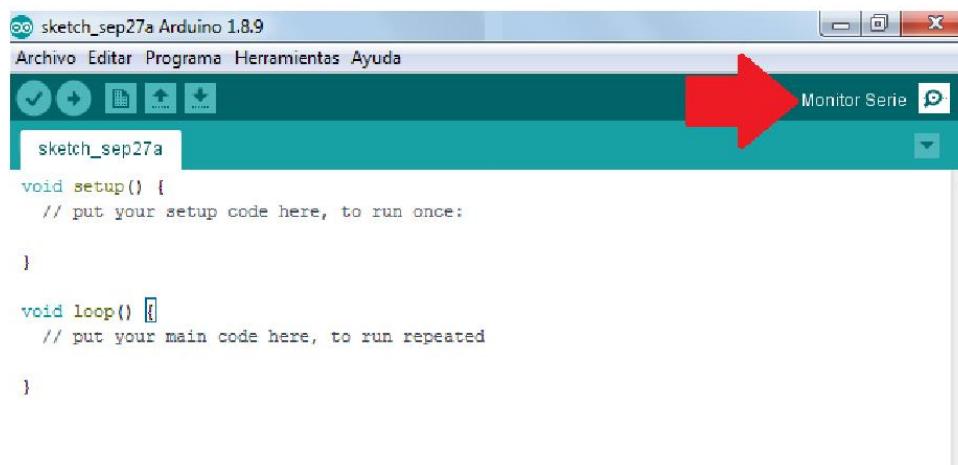
```
void setup() {
    /* Esto
       es un comentario
       multilinea
       en Arduino
    */
}

void loop() {
    /* Esto
       es un comentario
       multilinea
       en Arduino
    */
}
```

Capítulo 4. SALIDAS. PARTE 1

4.1.- Serial Monitor

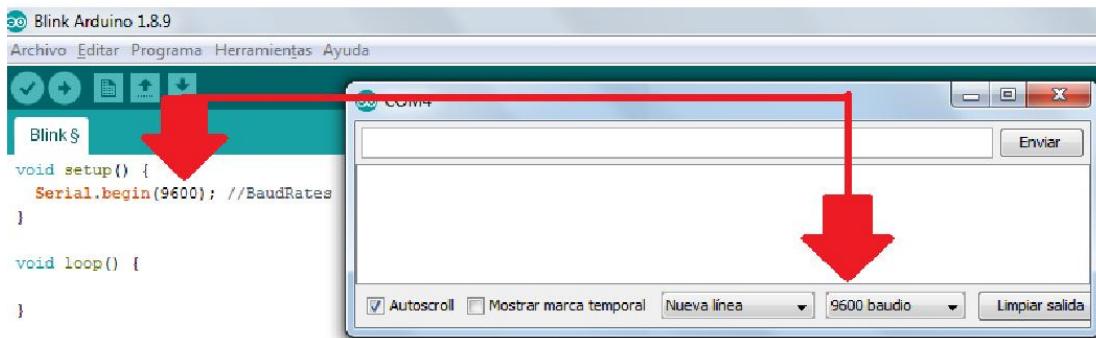
Cuando el código cargado en una placa de Arduino empieza su ejecución, es difícil saber en un momento dado qué está haciendo o cuál instrucción está ejecutando. Para ayudar en este fin, el IDE de Arduino tiene una herramienta llamada "Serial Monitor" que puede ser de gran ayuda. El Serial Monitor es una ventana emergente que actúa como un terminal separado que puede recibir y enviar información a la placa. Está ubicado en la parte superior derecha en el ícono de una lupa, como se muestra en la siguiente imagen:



El Serial Monitor se puede utilizar para mostrar mensajes desde el código (salidas) o para recibir datos desde la computadora. Primero, debe estar conectada la placa de Arduino a la computadora a través del USB. Una vez al hacer click en Monitor Serial aparecerá una nueva ventana. Esta ventana contiene:

- Un formulario donde se pueden enviar datos al Arduino.
- Un área donde se muestran los datos enviados desde el Arduino.
- Un menú para seleccionar la velocidad de BaudRates.

En la función `setup()` se inicializa la comunicación serial indicando el Baud Rate (velocidad de transmisión) con que la información será transferida a la computadora o desde esta. Por ejemplo:



En el ejemplo anterior, se inicializa la comunicación serial especificando el Baud Rate en 9600, aunque pueden ser otros valores: 19200, 38400, 57600, etc. Para que el monitor serial pueda mostrar los mensajes enviados por el código de la placa, debe coincidir este valor con el especificado en la ventana del Serial Monitor.

4.2.- Enviando Salidas

Para hacer que aparezcan mensajes en el Serial Monitor se pueden utilizar varias funciones de "Serial", entre las más usadas están:

- `print`
- `println`
- `write`

En general, todas harán que aparezca información en la pantalla del Monitor Serial. Sus diferencias se explicarán a continuación.

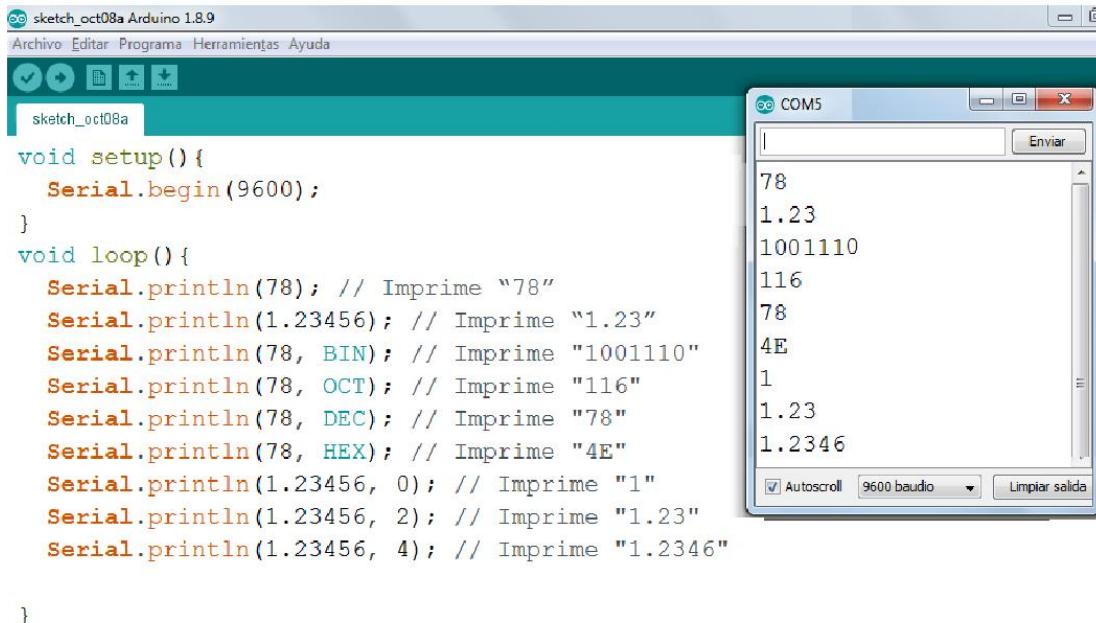
`Serial.print()`: imprime un dato legible en código ASCII. Este formato puede tomar muchas formas. Los números son impresos utilizando el código ASCII para cada número, los números decimales son impresos con 2 decimales, los caracteres y textos son impresos como son enviados. Ejemplo:

```
Serial.print(78);      // Imprime "78"
Serial.print(1.23456); // Imprime "1.23"
```

Un segundo parámetro opcional especifica el formato o base a usar, permite valores BIN (Binarios o de base 2), OCT (octal o de base 8), DEC (decimal o de base 10), HEX (hexadecimal o de base 16). Para números decimales este segundo parámetro especifica la cantidad de decimales a imprimir. Ejemplo:

```
Serial.print(78, BIN);    // Imprime "1001110"
Serial.print(78, OCT);    // Imprime "116"
Serial.print(78, DEC);    // Imprime "78"
Serial.print(78, HEX);    // Imprime "4E"
Serial.print(1.23456, 0); // Imprime "1"
Serial.print(1.23456, 2); // Imprime "1.23"
Serial.print(1.23456, 4); // Imprime "1.2346"
```

`Serial.println()`: funciona igual que `Serial.print()` excepto que al término del último carácter se agregan dos bytes adicionales, estos son: retorno de carro y salto de línea. Esto hará que al mostrar un mensaje, el cursor salte a la línea siguiente. Por ejemplo:



`Serial.write()`: Esta función envía un byte o una serie de bytes al serial monitor.



4.3.- Retardos

El retardo en Arduino es una función que permite hacer esperar al procesador una cierta cantidad de tiempo para avanzar a la siguiente instrucción. Generalmente se utiliza para apreciar visualmente los cambios de valores lógicos, como por ejemplo, el encendido y apagado de un led, evitar ruido en las señales eléctricas o para no “embasurar” el bus de datos, entre otras cosas. Existen dos formas de hacer retrasos:

- `delay`
- `delayMicroseconds`

Las diferencias entre estas dos funciones se explicarán a continuación:

`Delay()`: pausa el programa por la cantidad de tiempo (milisegundos) especificado como parámetro (hay 1000 milisegundos en 1 segundo). La sintaxis general es:

`delay(ms)`

Donde "ms" es el número de milisegundos que estará en pausa el programa antes de ejecutar la siguiente instrucción. En el siguiente ejemplo, se imprime "HOLA MUNDO" cada 10 segundos en el serial monitor.

```

4.3.Retardos_Delay Arduino 1.8.9
Archivo Editar Programa Herramientas Ayuda
4.3_Retardos_Delay
void setup() {
    Serial.begin(9600);
}
void loop() {
    Serial.println("HOLA MUNDO");
    delay(10000);
}

```

The Serial Monitor window shows the text "HOLA MUNDO" printed 10 times, indicating a 10-second delay between each print statement.

DelayMicroseconds(): pausa el programa por la cantidad de microsegundos especificados por parámetro (hay mil microsegundos en 1 milisegundo y un millón de microsegundos en 1 segundo). El mayor valor que puede tomar DelayMicroseconds() para un retardo preciso es de 16383. Para retardos más largos que mil microsegundos, es recomendable usar delay(). La sintaxis general es:

```
delayMicroseconds(us);
```

Donde "us" es el número de microsegundos que pausara el programa.

```

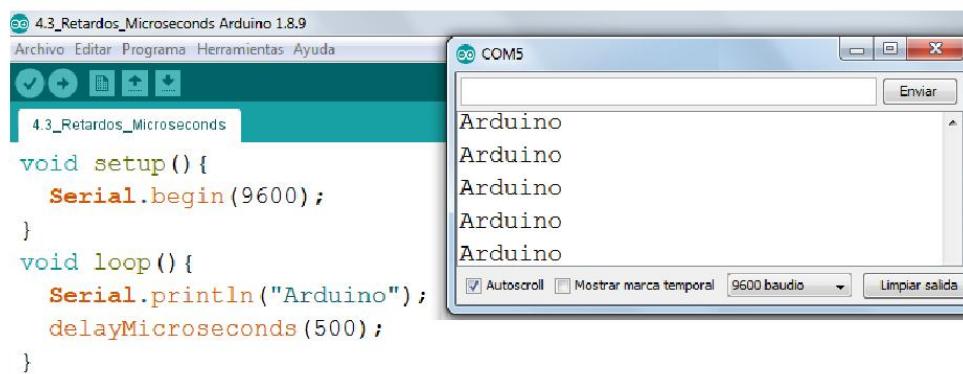
pruebaarduino Arduino 1.8.9
Archivo Editar Programa Herramientas Ayuda
pruebaarduino
void setup() {
    Serial.begin(9600); //BaudRates
}

void loop() {
    Serial.println("Arduino");
    delayMicroseconds(500);
}

```

The Serial Monitor window shows the word "Arduino" printed 15 times, with a 500-microsecond delay between each print statement.

El siguiente ejemplo imprime el texto “Arduino” cada 500 microsegundos en el serial monitor:



The screenshot shows the Arduino IDE interface. On the left, the code for '4.3_Retardos_Microseconds' is displayed:

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  Serial.println("Arduino");
  delayMicroseconds(500);
}
```

On the right, the Serial Monitor window titled 'COM5' is open, showing the output of the program. It displays the word 'Arduino' five times in a row, each time separated by a small delay. The monitor settings are set to 9600 baud.

4.4.- Millis()

A diferencia de los retardos, millis() retorna el número de milisegundos que han transcurrido desde que empezó a correr el programa actual. Este número volverá a 0 después de 50 días.

Sintaxis:

time = millis()

Valor de retorno:

Número de milisegundos que han pasado desde que empezó la ejecución del programa.

Tipo de dato: unsigned long.

The screenshot shows the Arduino IDE interface. On the left, the code editor displays a sketch named 'sketch_oct14a' with the following content:

```
unsigned long time;

void setup() {
    Serial.begin(9600);
}

void loop() {
    Serial.print("Time: ");
    time = millis();

    Serial.println(time);
    delay(1000);
}
```

On the right, the串行监视器 (Serial Monitor) window titled 'COM5' shows the output of the sketch. It displays a series of time values printed by the code:

Time: 0
Time: 999
Time: 1999
Time: 3000
Time: 4000
Time: 5000
Time: 6000

The monitor also includes settings at the bottom: 'Autoscroll' (checked), 'Mostrar' (unchecked), '9600 baudio', and a 'Limpieza de salida' button.

Capítulo 5. SALIDAS. PARTE 2

5.1.- Configuración de Pines

Los pines de un componente son cada uno de los contactos metálicos que este posee, los mismos son fabricados de un material conductor. Estos pines cumplen una función específica: enviar o recibir señales eléctricas.

Los pines encargados de recibir señales eléctricas son denominados pines de entrada, mientras que los pines encargados de enviar señales eléctricas son denominados pines de salida.

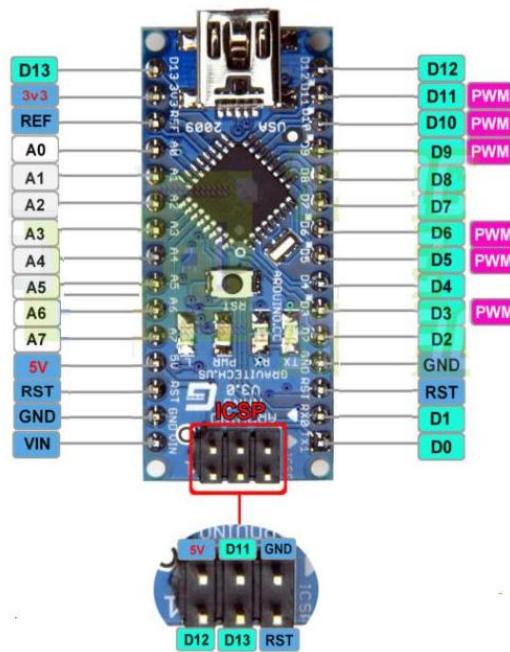
Arduino posee pines de salidas de valores constantes, estos son: 5V, 3.3V y 0V (GND), pueden ser utilizados en el apartado "POWER" de la placa.



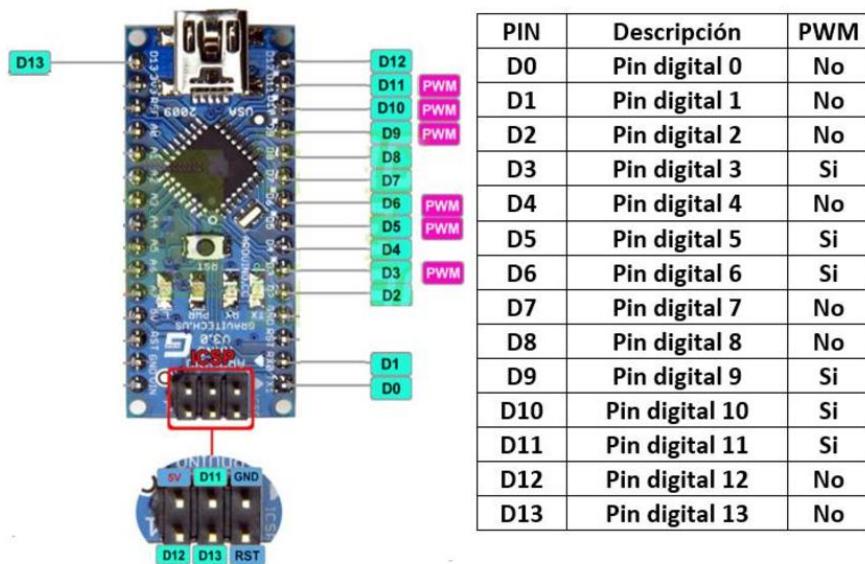
Para utilizar un Arduino hay que saber las funciones de cada uno de sus pines. Algunos pines están configurados para funcionar solamente como entrada o como salida. Cabe destacar que también hay pines que solamente reciben señales analógicas o digitales.

De esta manera se podrán configurar correctamente los pines para el buen funcionamiento del Arduino y así evitar errores de lecturas o salidas.

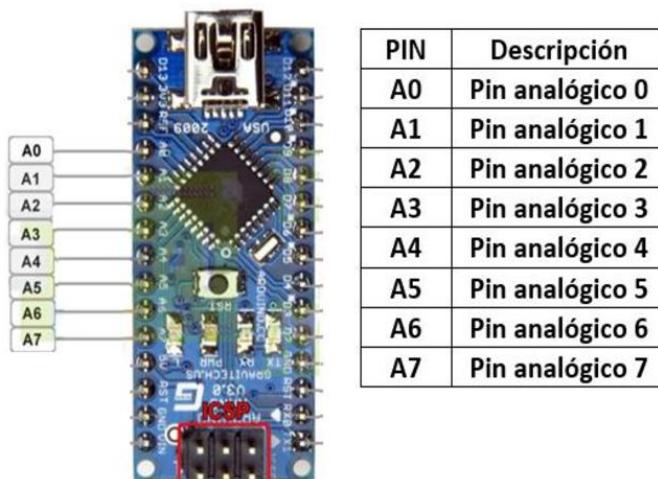
En la siguiente imagen se muestran cada uno de los pines del Arduino Nano:



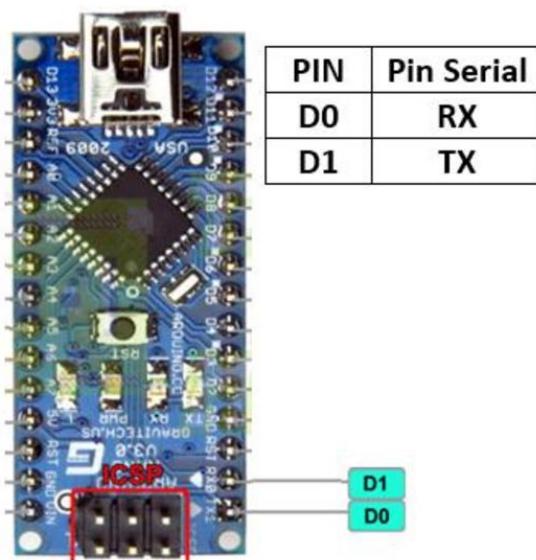
Pines Digitales: Estos pines solamente pueden poseer 2 valores en sus salidas: 0V o 5V. Estos son identificados con la letra D y el n mero del pin. Pueden ser utilizados tanto como entrada o salida. Algunos de estos pines poseen la caracter stica PWM, los cuales pueden tener en su salida valores entre 0V y 5V (incluyendo los extremos), llamadas tambi n salidas anal gicas.



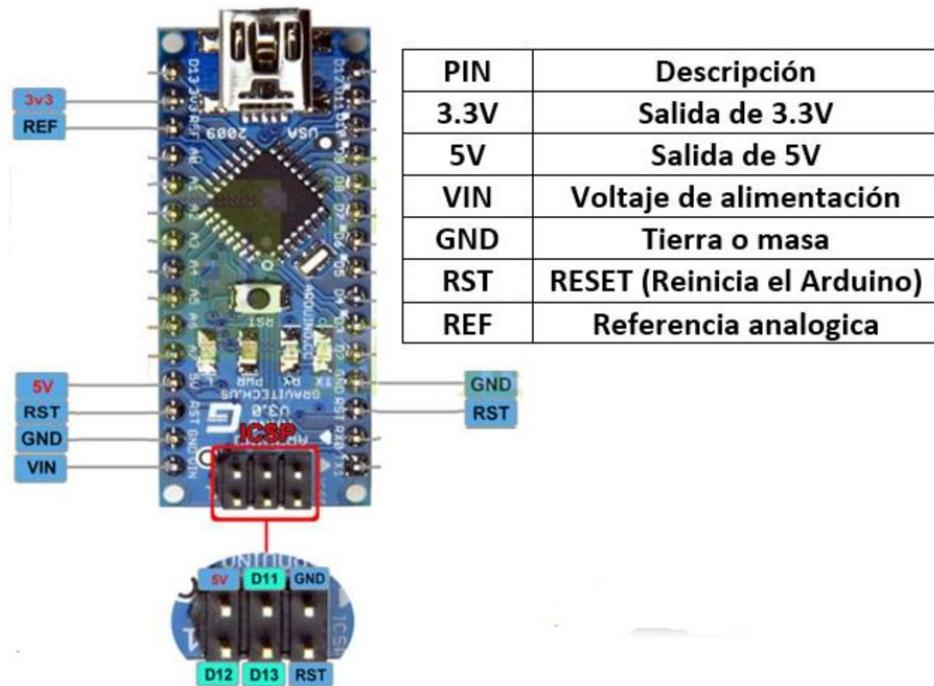
Pines Analógicos: Estos pines solamente pueden ser utilizados como entradas analógicas. Para salidas analógicas se utilizarán los pines PWM. Estos son identificados con la letra A y el número de pin.



Pines de comunicación: Estos pines son usados para la comunicación entre el Arduino y la computadora u otros dispositivos. Conectar algún componente o sensor a estos pines podría interferir con la comunicación, incluso dar errores de transferencia a la placa. Por lo tanto, si se utilizará el serial monitor no se deben conectar a nada para evitar errores (estos son identificados con las letras RX y TX)



Algunos pines del Arduino poseen una función en concreto, algunos son para dar valores constantes o para reiniciar el Arduino.



Por defecto, los pines de todos los Arduinos están configurados como entrada, así que para utilizarlos como salidas hay que cambiar su configuración por defecto. Para esto, se utiliza la función "pinmode" en la función setup() del programa. La forma general de usar la función es la siguiente:

```
pinMode(pin,OUTPUT);
```

Esta función es exclusiva de los pines digitales. Configura el pin especificado a que se comporte como una salida si recibe como segundo parámetro "OUTPUT", y "pin" es el número del pin al que se quiere configurar.

Ejemplo:

```
pinMode(10, OUTPUT); // configura el pin número 10 como salida.
```

5.2.- Digital Output

Los pines configurados como salida están en un estado de baja impedancia. Esto significa que pueden suministrar una substancial cantidad de corriente a otros circuitos.

Los pines del Atmega pueden suministrar hasta 40mA de corriente a otros dispositivos o circuitos, esto es suficiente corriente para encender un LED pero no es suficiente corriente para manejar relays, solenoides o motores.

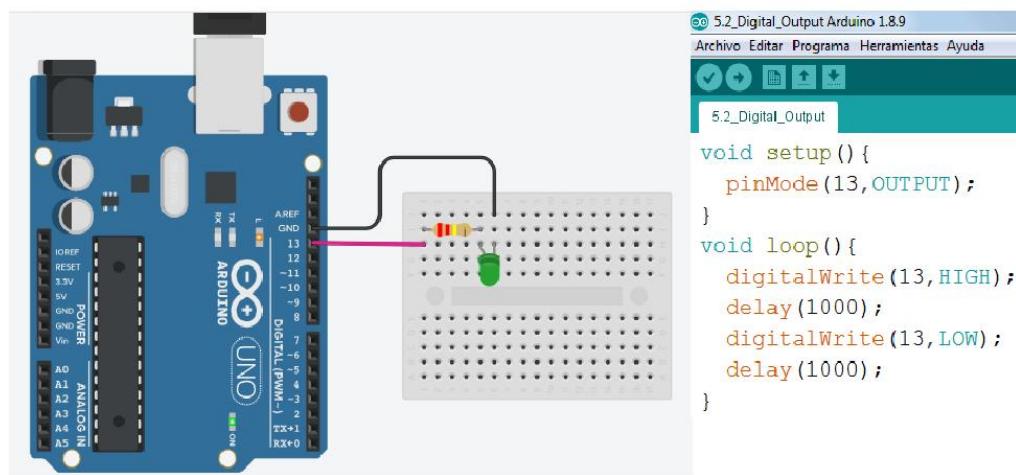
La función digitalWrite() establece en alto o en bajo un pin digital, si el pin ha sido configurado como una salida con pinMode(). La sintaxis general de la función es la siguiente:

```
digitalWrite(pin,value)
```

Donde “Pin” es el número de pin del Arduino y “value” el valor de voltaje que se desea enviar como salida. Los posibles valores son:

- HIGH: 5V o 3.3V (depende del máximo voltaje que puede dar como salida la placa).
- LOW: 0V (ground).

El siguiente ejemplo muestra como encender un led que está conectado al pin digital 13 durante 1 segundo y luego lo apaga durante 1 segundo:



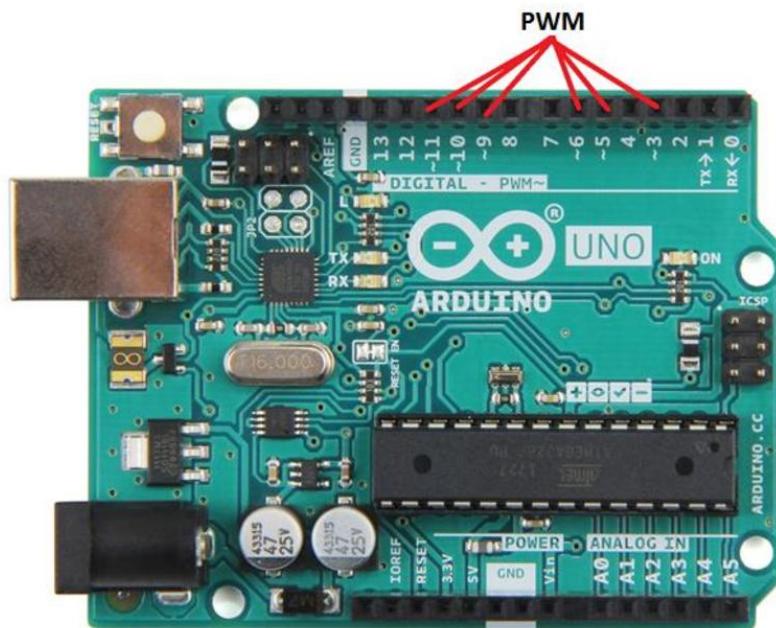
5.3.- Analog Output

La función PWM permite generar valores intermedios de salidas entre 0V Y 5V escalados en 256 niveles, siendo 0 el más bajo (0V) y 255 el más alto (5V).

El microcontrolador alterna estados HIGH y LOW un tiempo proporcional al valor requerido. De esta manera, con 127 estaría el 50% del tiempo HIGH y el otro 50% restante a LOW, dando en la salida 2.5V.

Las personas no pueden captar estos cambios ya que es a una frecuencia muy rápida y el ojo humano no puede captarlo.

En algunas placas, los pines con esta función poseen el signo ~ al lado de su número.



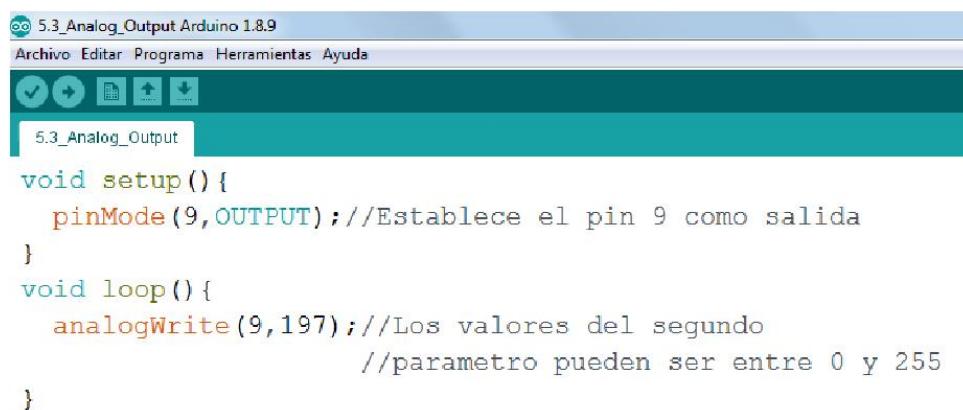
La función `analogWrite()` establece un valor a un pin analógico de salida (PWM). Puede ser usado para variar el brillo de un led o manejar un motor en diferentes velocidades.

Después de llamar `analogWrite()`, el pin generará el voltaje establecido entre los 255 valores. La sintaxis general es:

```
analogWrite(pin,value)
```

Donde "pin" es el pin al que se le establecerá el valor en el Arduino(PWM) y "value" es el valor entre 0 y 255 (0 = 0V y 255 = 5V).

En el siguiente ejemplo, se establece el pin 9 como salida y se envía a través de este un valor de 197, equivalente a 3.86 V aproximadamente:



The screenshot shows the Arduino IDE interface with the title bar "5.3_Analog_Output Arduino 1.8.9". The menu bar includes "Archivo", "Editar", "Programa", "Herramientas", and "Ayuda". Below the menu is a toolbar with icons for file operations. The main window displays the following code:

```
void setup() {
    pinMode(9,OUTPUT); //Establece el pin 9 como salida
}
void loop(){
    analogWrite(9,197); //Los valores del segundo
                        //parametro pueden ser entre 0 y 255
}
```

Capítulo 6. VARIABLES

6.1.- Declaración de Variables

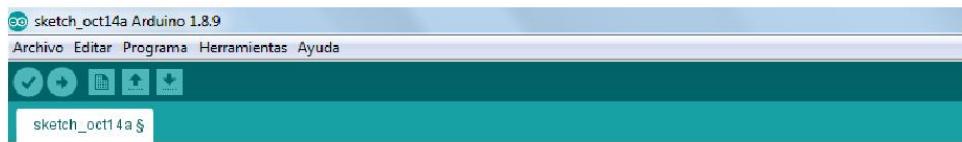
Una variable es un espacio de memoria del programa a la cual se le asigna un valor. Este valor puede variar durante la ejecución del programa. A este espacio de memoria se le hace referencia a través del nombre dado a la variable. El rango de valores que puede tomar una variable depende de su tipo de dato. Entre los tipos de datos más básicos que se pueden utilizar en Arduino se encuentran:

Nombre	Dato	Espacio que ocupa	Valores
<code>bool</code>	true / false	1 Byte	true o false
<code>byte</code>	Número entre 0 y 255	1 Byte	Número entre 0 y 255
<code>char</code>	Carácter de 1 byte	Al menos 1 Byte	Carácter alfanumérico
<code>double</code>	Números decimales	4 Bytes	[-3.4028235E+38 , 3.4028235E+38]
<code>float</code>	Números decimales	4 Bytes	[-3.4028235E+38 , 3.4028235E+38]
<code>int</code>	números enteros de 2 bytes	2 Bytes	[-32768, 32767]
<code>String</code>	Texto		Secuencia de caracteres char.

Existe otros tipos de datos que se utilizan en fine más específicos:

Nombre	Dato	Espacio que ocupa	Valores
<code>long</code>	Números de gran tamaño	4 Bytes	-2147486648 a 2147486647
<code>short</code>	Tipo de dato de 16 bits	2 Bytes	-32768 a 32767
<code>unsigned char</code>	Al igual que "byte" codifica números de 0 a 255	1 Byte	Número entre 0 a 255
<code>unsigned int</code>	A diferencia de int, solamente almacena enteros positivos	2 Bytes	[0 a 65535]
<code>unsigned long</code>	A diferencia de long, almacenar valores únicamente positivos	4 Bytes	[0 a 4294967295]
<code>word</code>	Almacena un número sin signo de al menos 16 bits	2 Bytes	[0 a 65535]
<code>size_t</code>	Es un tipo de dato capaz de almacenar el tamaño de un objeto en bytes		
<code>void</code>	Declaración de funciones vacías		

Para declarar una variable se debe especificar el nombre de la variable y el tipo de dato. El tipo de dato de la variable debe ser especificado antes del nombre y este dependerá de los valores que se necesiten almacenar en dicha variable. Por ejemplo:



```
sketch_oct14a Arduino 1.8.9
Archivo Editar Programa Herramientas Ayuda
sketch_oct14a §

int numero1; //Se declara una variable tipo int
long numero2; //Se declara una variable tipo long
bool boton1,boton2; //Se pueden declarar 2 variables
                    // del mismo tipo de dato
                    // separadas por coma (,)
char caracter; //Se declara una variable tipo char
float numdecimal; //Se declara una variable de tipo float.
void setup() {
}
void loop() {
}
```

Las variables deben declararse antes de hacer referencia a ellas. Pueden declararse en cualquier lugar del código, sin embargo, es una buena práctica declararlas al principio del programa para utilizarlas en cualquier línea de código donde se necesiten.

En Arduino, los identificadores son sensibles a mayúsculas y minúsculas, por lo tanto, al hacer referencia a estas deben usarse tal cual como fue declarada.

6.2.- Asignaciones

Las variables pueden ser inicializadas (asignarle un valor) cuando éstas son declaradas o se les puede asignar un valor durante la ejecución del programa. A una variable se le puede asignar un valor literal (constante) o el valor de otra variable. Para los tipos de datos char, el carácter debe estar encerrado entre comillas simples (''), mientras que para un String el texto debe estar entre comillas dobles (""). Ejemplo:

The screenshot shows the Arduino IDE interface. On the left, the code editor displays a sketch named 'sketch_oct14a' with the following content:

```
sketch_oct14a Arduino 1.8.9
Archivo Editar Programa Herramientas Ayuda
sketch_oct14a

int numero = 1 ,numero2 = 10;
char char1 = 'A', char2 = 'B';
void setup() {
    Serial.begin(9600);
}
void loop() {
    Serial.println(numero2);
    numero2 = numero;
    Serial.println(numero2);
    Serial.println(char2);
    char2 = char1;
    Serial.println(char2);
    delay(8000);
}
```

On the right, the Serial Monitor window is titled 'COM5'. It shows the output of the sketch's serial communication. The text in the monitor is:

```
10
1
B
A
```

Below the monitor, there are settings for 'Autoscroll' (checked), '9600 baudio', and a 'Limpiar salida' button.

En caso de que a una variable de tipo entero se le asigne el valor de una variable de tipo char, esta almacenará el código ASCII del carácter que almacenado en la variable.

The screenshot shows the Arduino IDE interface. On the left, the code editor displays the following sketch:

```
sketch_oct14a Arduino 1.8.9
Archivo Editar Programa Herramientas Ayuda
sketch_oct14a
int numero = 5;
char letra = 'A';
String texto;
void setup() {
    Serial.begin(9600);
}
void loop() {
    Serial.println(numero);
    numero = letra;
    Serial.println(numero);
    texto = "HOLA";
    Serial.println(texto);
    delay(5000);
}
```

The code uses variables `numero` and `letra` to demonstrate how changes in one variable affect the other through assignment. The serial monitor window on the right, titled "COM5", shows the output of the serial print statements. The first two lines, "5" and "65", are highlighted with red boxes. The word "HOLA" is also visible in the monitor.

6.3.- Constantes

Una constante es una variable de solo lectura. Esto significa que puede ser usada como cualquier otra variable, pero no puede modificarse su valor. En caso de intentar modificarla, el compilador marcará un error de “intento de asignación de valor a una constante”. Las constantes son declaradas con la palabra reservada “const”. Es de buena práctica declarar las constantes con mayúsculas.

```

7.3_Constantes Arduino 1.8.9
Archivo Editar Programa Herramientas Ayuda
7.3_Constantes
const int CONSTANTE = 5;
int numero = 2;
void setup() {
    Serial.begin(9600);
}
void loop() {
    numero = CONSTANTE;
    Serial.println(numero);
    delay(10000);
}

```

5

Autoscroll Mostrar marca temporal 9600 baudio Limpiar salida

Arduino por defecto predefine expresiones constantes. Estas son usadas para hacer los programas más fáciles de leer. Las constantes predefinidas se clasifican en:

Grupo	Constante
Para definir estados lógicos	true y false
Para definir los estados en los pines	HIGH y LOW
Para definir los pines si son de entrada o de salida	INPUT, INPUT_PULLUP y OUTPUT
Para componentes de la placa	LED_BUILTIN (el cual es el número del pin donde está ensamblado el led en la placa.)

Para definir valores lógicos están las constantes true y false (Constantes Booleanas)

false: es la más fácil de definir de las 2 y está definida como 0.

true: a menudo se dice que true está definido como 1, lo cual es correcto, pero true tiene una definición más extensa. Cualquier entero que sea diferente de 0 es true en una sentencia Booleana. Así que -1, 2 y 200 también toman valores true en una sentencia Booleana.

Estas constantes (true y false) están declaradas en minúsculas a diferencia de HIGH, LOW, INPUT y OUTPUT, por lo que para su uso, hay que escribirlas tal cual.

Para definir el nivel de un PIN:

Al momento de leer o escribir el valor de un pin digital solamente hay 2 valores posibles que el pin puede tomar o ser establecido: HIGH y LOW.

HIGH: el significado de HIGH (en referencia a un pin) es diferente dependiendo si el pin está configurado como entrada o salida.

Cuando un pin está configurado como entrada y su valor es leído, Arduino retornará el valor de HIGH si:

- El pin posee un voltaje mayor a 3V (Para placas de 5V)
- El pin posee un voltaje mayor a 2V (Para placas de 3.3V)

En caso de que el pin esté configurado como salida y se le establece el valor HIGH, el pin tendrá el valor de:

- 5 Voltios (en placas de 5 Voltios)
- 3.3 Voltios (en placas de 3.3 Voltios)

LOW también posee un significado diferente dependiendo si el pin está configurado como entrada o como salida. Cuando un pin está configurado como entrada y se lee su valor, Arduino retornará LOW si:

- El voltaje en el pin es menor que 1.5V (en placas de 5V).
- El voltaje en el pin es menor que 1V (en placas de 3.3V).

En caso de que esté configurado como salida y este se le establece el valor LOW, el valor del pin estará a 0V para placas de 5V y 3.3V.

Para definir la configuración del PIN:

Los pines digitales pueden ser usados como INPUT (entrada), INPUT_PULLUP (entrada con resistencia pullup interna de Arduino) o OUTPUT, dependiendo de esta configuración cambiará el comportamiento eléctrico del pin.

Pines configurados como INPUT: Se dice que están configurados con un alto estado de impedancia, lo cual lo hace útil para leer un sensor.

Pin configurado como INPUT_PULLUP:

Los microcontroladores ATmega de Arduino poseen una resistencia PULL-UP interna la cual se puede usar en caso de que se prefiera utilizar esta en vez de una resistencia externa.

Pines configurados como OUTPUT:

Estos pines están en un estado de baja impedancia, lo cual pueden suministrar una sustancial cantidad de corriente a otros circuitos, hasta un máximo de 40mA, para cargas mayores a 40mA se deberá usar un transistor u otro circuito de transferencia.

Para utilizar componentes propios de la placa LED_BUILTIN:

La mayoría de las placas Arduino poseen un pin conectado a un led en serie con una resistencia, la constante LED_BUILTIN es el número del pin en el cual el led está conectado. La mayoría de las placas tienen este led conectado al pin digital 13.

Capítulo 7. OPERADORES ARITMÉTICOS

7.1.- Operadores Aritméticos

Los operadores aritméticos toman valores numéricos (ya sean literales o variables) como sus operandos y retornan un valor numérico. Los operadores aritméticos estándar son: adición o suma (+), sustracción o resta (-), multiplicación (*), y división (/).

Signo	Descripción	Ejemplo
+	Realiza la operación de la adición	sum = operando1 + operando2;
-	Realiza la operación de la sustracción.	dif = operando1 - operando2;
*	Realiza la operación de la multiplicación.	mul = operando1 * operando2;
/	Realiza la operación de la división.	div = operando1 / operando2;
=	Este signo toma el valor de la derecha y lo almacena en la variable a la izquierda del signo “igual”	asig = 30 o asig = variable;
%	Calcula el residuo de un numero dividido por otro	resi = operando1 % operando2

The screenshot shows the Arduino IDE interface. The main window displays the code for a sketch named 'pruebaarduino'. The code performs basic arithmetic operations on variables num1 (150) and num2 (50), printing the results to the Serial Monitor. The Serial Monitor window, titled 'COM4', shows the output:

```
Los numeros son: 150 y 50
La suma es: 200
La resta es: 100
La multiplicacion es: 7500
La division es: 3.00
```

7.2.- Operadores y Asignaciones

Se pueden resumir operaciones aritméticas básicas y asignaciones usando operadores especiales. Entre ellos están:

Signo	Descripción	Ejemplo
++	Incrementa el valor de la variable por 1	<code>x++; // equivale a x = x + 1;</code>
--	Decrementa el valor de la variable por 1	<code>x--; // equivale a x = x - 1;</code>
+=	Realiza la adición de la variable con otra variable o constante	<code>x += y; // equivale a x = x + y;</code>
-=	Realiza la sustracción de la variable con otra variable o constante	<code>x -= y; // equivale a x = x - y;</code>
*=	Realiza la multiplicación de la variable con otra variable o constante	<code>x *= y; // equivale a x = x * y;</code>
/=	Es una manera de realizar la división de la variable con otra variable o constante	<code>x /= y; // equivale a x = x / y;</code>

The image shows the Arduino IDE interface. On the left, the code for 'pruebaarduino' is displayed:

```

int num1 = 10;
int num2 = 5;
void setup() {
    Serial.begin(9600); //BaudRates
    Serial.println("Los numeros son 10 y 5");
    Serial.print("El resultado con += es: ");
    num1 += num2;
    Serial.println(num1);
    // ahora num1 tiene el valor 15
    Serial.print("El resultado con -= es: ");
    num1 -= num2;
    Serial.println(num1);
    // ahora num1 tiene el valor 10
    Serial.print("El resultado con *= es: ");
    num1 *= num2;
    Serial.println(num1);
    //ahora num1 tiene el valor 50
}
void loop() {
}

```

On the right, the serial monitor window titled 'COM4' shows the output of the program:

```

Los numeros son 10 y 5
El resultado con += es: 15
El resultado con -= es: 10
El resultado con *= es: 50

```

7.3.- Números Aleatorios

En Arduino los números aleatorios son en realidad números pseudo-aleatorios, que son generados por medio de una función determinista y no aleatoria lo cual los hace aparentar ser aleatorios. Estos números pseudo-aleatorios se generan a partir de un valor inicial.

El uso de números aleatorios permite generar diferentes resultados al azar, de esta manera se puede simular el lanzamiento de una moneda, de un dado, entre otras cosas, y dependiendo del resultado el encendido o apagado de un led o alguna señal eléctrica a un circuito externo.

La función para generar números pseudo-aleatorios entre un rango establecido se llama `random()`.

Sintaxis: random(max) o random(min,max).

Parámetros:

min: el número mínimo del rango (obligatorio)

max: el número máximo del rango (opcional)

valor de retorno: un número generado al azar entre min y max-1. El tipo de dato del valor returned por random es long.

En el siguiente ejemplo se puede observar como la sucesión de números en diferentes ejecuciones del programa siempre es la misma.

```

sketch_oct14a Arduino 1.8.9
Archivo Editar Programa Herramientas Ayuda
sketch_oct14a
long randNumber;
void setup() {
    Serial.begin(9600);
}
void loop() {
    // Imprime un numero aleatorio entre 50 y 99
    randNumber = random(50,100);
    Serial.println(randNumber);
    delay(1000);
}

```

1	2	3
57	57	57
99	99	99
73	73	73
58	58	58
80	80	80
72	72	72
94	94	94
78	78	78

Si se requiere que la secuencia de random() difiera en sucesivas ejecuciones del programa, se debe utilizar randomSeed(). Esta función inicializa el generador de números aleatorios. Si se necesita que en cada ejecución la secuencia de números aleatorios sea diferente, hay que inicializarlo con un valor variable.

En algunas ocasiones puede ser útil secuencias pseudo-aleatorias que se repitan exactamente. Esto puede hacerse con randomSeed() especificando un número fijo por parámetro antes de usar la secuencia aleatoria.

Sintaxis: randomSeed(int).

En el siguiente ejemplo se inicializa el generador de números aleatorios con randomSeed() asignándole por parámetro el número 20. Cada vez que se ejecute el programa mostrará la misma sucesión de números al menos que el valor de la inicialización sea modificado.

The screenshot shows the Arduino IDE interface. On the left, the code for 'sketch_oct14a' is displayed:

```
sketch_oct14a Arduino 1.8.9
Archivo Editar Programa Herramientas Ayuda
sketch_oct14a
long randNumber;
void setup() {
    Serial.begin(9600);
    randomSeed(20); ← Red arrow points here
}
void loop() {
    // Imprime un numero aleatorio entre 50 y 99
    randNumber = random(50,100);
    Serial.println(randNumber);
    delay(1000);
}
```

On the right, the Serial Monitor window shows three columns of data. Each column has a different colored border (orange, green, blue) around its first few rows. The data is as follows:

Column 1 (Orange)	Column 2 (Green)	Column 3 (Blue)
90	90	90
86	86	86
55	55	55
87	87	87
80	80	80
52	52	52
80	80	80

The 'Enviar' button is visible at the top right of the Serial Monitor window.

Capítulo 8. CONDICIONES. PARTE 1

8.1.- Operadores Relacionales

Los operadores relacionales son símbolos que se utilizan para comparar valores, creándose de esa forma una expresión lógica. Si el resultado de la comparación es correcto, la expresión es considerada verdadera, en caso contrario es falsa. Internamente, un valor "true" es equivalente al valor 1 y un valor "false" es equivalente al valor 0.

Signo	Descripción	Sintaxis
<code>==</code>	Es igual a	<code>x == y</code>
<code>!=</code>	No es igual a	<code>x != y</code>
<code><</code>	Es menor que	<code>x < y</code>
<code>></code>	Es mayor que	<code>x > y</code>
<code><=</code>	Es menor o igual a	<code>x <= y</code>
<code>>=</code>	Es mayor o igual	<code>x >= y</code>

En el siguiente ejemplo se le asigna a una variable de tipo bool una expresión lógica, al evaluar tal condición en la asignación, imprimirá en el serial monitor 0 o 1.

```

sketch_oct14a Arduino 1.8.9
Archivo Editar Programa Herramientas Ayuda

sketch_oct14a

long randNumber;
bool aprobado;
void setup(){
    Serial.begin(9600);
    randomSeed(2);
}
void loop(){
    //Numeros entre 0 y 20
    randNumber = random(21);
    aprobado = randNumber >= 10;
    //Imprime 1 si es true
    //Imprime 0 si es false
    Serial.print(randNumber);
    Serial.print(" ");
    Serial.println(aprobado);
    delay(1000);
}

```

8.2.- If

La estructura de control "if" verifica una condición y ejecuta el código establecido en ella en caso de que la expresión lógica colocada entre paréntesis sea verdadera (true). La sintaxis general es la siguiente:

```

if(expresionLogica)
    InstruccionCondicionada;

```

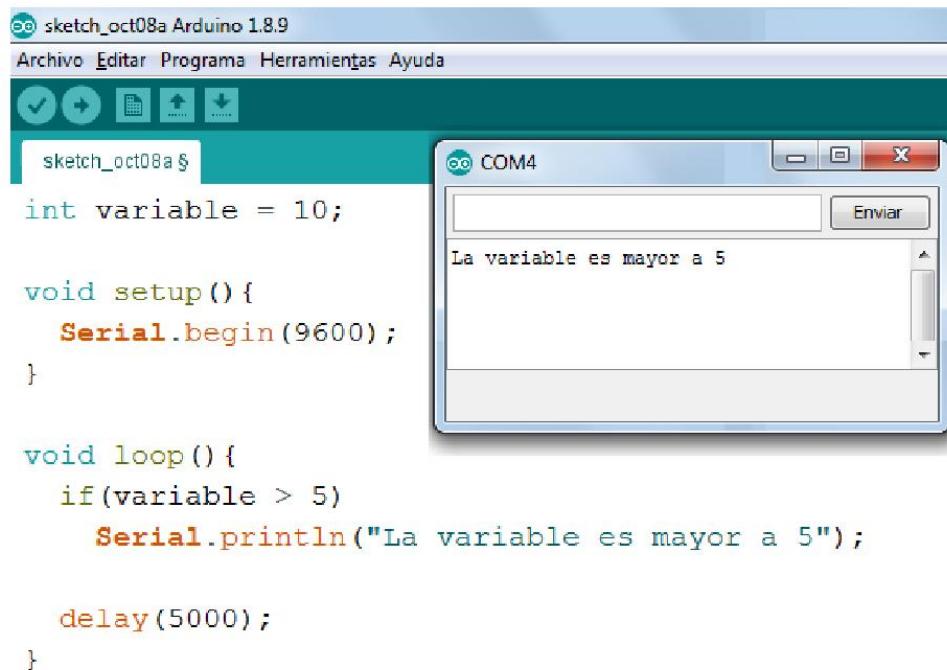
En caso de que sean varias las líneas de código que se desean condicionar, se encierran las instrucciones entre llaves ({}), todas las instrucciones condicionadas:

```

if(expresionLogica){
    InstruccionCondicionada1;
    InstruccionCondicionada2;
    InstruccionCondicionadaN;
}

```

En el siguiente ejemplo se muestra un mensaje en el serial monitor cuando el valor de la variable sea mayor a 5:



The image shows the Arduino IDE interface. On the left, the code editor window displays a sketch named "sketch_oct08a" with the following code:

```
int variable = 10;

void setup() {
    Serial.begin(9600);
}

void loop() {
    if(variable > 5)
        Serial.println("La variable es mayor a 5");

    delay(5000);
}
```

On the right, the Serial Monitor window titled "COM4" shows the output "La variable es mayor a 5".

En el siguiente ejemplo se evalúa si una variable es mayor o igual a 5, en caso de ser verdadero se multiplica la variable por 5 y se imprime en el Serial Monitor:

```

sketch_oct08a Arduino 1.8.9
Archivo Editar Programa Herramientas Ayuda
sketch_oct08a
int variable = 9;
void setup(){
    Serial.begin(9600);
}
void loop(){
    if(variable >= 5){
        Serial.print("El numero de la variable es: ");
        Serial.println(variable);
        variable *= 5;
        Serial.print("Su valor multiplicado por 5 es: ")
        Serial.println(variable);
    }
    delay(5000);
}

```

8.3.- If/else

La estructura if...else permite tener un mayor control en el flujo de la ejecución que con solamente if. Si la condición evaluada en "if" es falsa el bloque de código de "else" será ejecutado. La sintaxis general es:

```

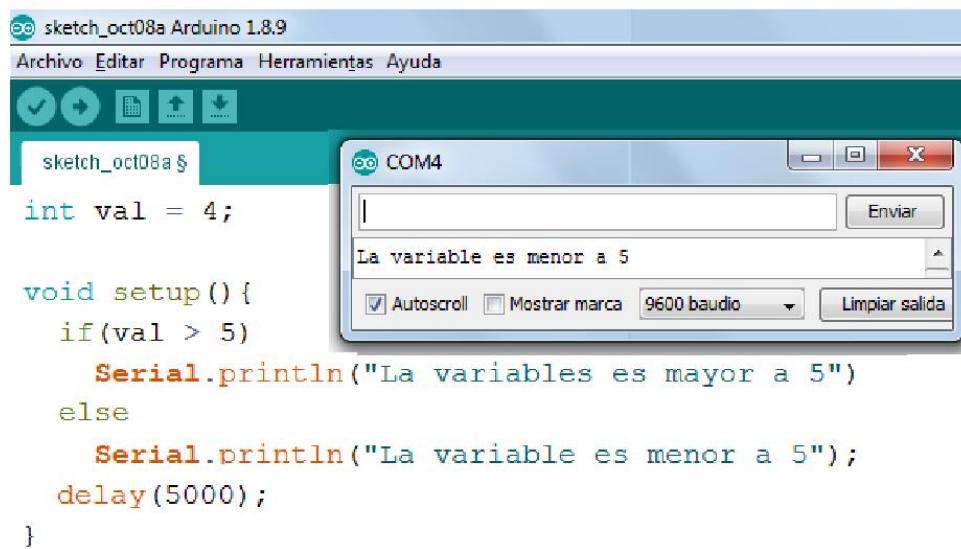
if(expresionLogica) {
    //Instrucciones
    //Instrucciones
} else {
    //Instrucciones
    //Instrucciones
}

```

Al igual que "if", si las instrucciones dentro de if/else solamente posee una instrucción, no será necesario usar las llaves ({}). La sintaxis general es:

```
if(expresionLogica)
    //Instrucción
else
    //Instrucción
```

El siguiente ejemplo se muestra un mensaje cuando la variable sea mayor a 5, en caso contrario, se muestra otro mensaje:



El siguiente ejemplo muestra el uso de llaves cuando las instrucciones dentro de las estructuras condicionales tengan más de 1 línea de código. Además, dependiendo del valor de la variable se le sumará el valor 20 o se le restará 10.

The screenshot shows the Arduino IDE interface. On the left, the code for 'sketch_oct08a' is displayed:

```
sketch_oct08a Arduino 1.8.9
Archivo Editar Programa Herramientas Ayuda
sketch_oct08a
int variable = 14;
void setup() {
    Serial.begin(9600);
}
void loop() {
    Serial.print("La variable es: ");
    Serial.println(variable);
    if(variable <= 5){
        variable += 20;
        Serial.print("El resultado es: ");
        Serial.println(variable);
    }else{
        variable -= 10;
        Serial.print("El resultado es: ");
        Serial.println(variable);
    }
    delay(5000);
}
```

On the right, the 'COM5' serial monitor window shows the output of the code execution:

La variable es: 14
El resultado es: 4

En el siguiente ejemplo se le asigna una expresión lógica a una variable bool y muestra el valor que esta toma (true o 1, false o 0) en el serial monitor. En las expresiones lógicas sin operador relacional, lo que se verifica es si la variable tiene un valor distinto de 0 para ejecutar lo que está condicionado, en caso contrario, ejecuta el else.

The screenshot shows the Arduino IDE interface. On the left, the code editor displays a sketch named "8.3.3_if_else_Ejemplo". The code itself is as follows:

```
float nota = 11.5;
bool aprobado = nota >= 10;
void setup() {
    Serial.begin(9600);
}
void loop(){
    if(aprobado){
        Serial.println(aprobado);
    }else{
        Serial.println(aprobado);
    }
    delay(8000);
}
```

A red arrow points from the word "aprobado" in the "if" condition to the corresponding line in the Serial Monitor window. The Serial Monitor window, titled "COM5", shows the value "1" in its text input field, indicating the state of the variable "aprobado". The monitor also includes settings for "Autoscroll", "Retorno de carro", "9600 baudio", and a "Limpiar salida" button.

Capítulo 9. ENTRADAS POR EL PUERTO SERIAL

9.1.- Serial.available()

Enviar datos a la placa Arduino desde el serial monitor permite al programador simular sensores y entradas provenientes del exterior.

El programador puede probar como funciona su sistema y en caso de fallas o resultados no deseados, puede depurar su código.

No solamente funciona para depurar código, además, permite recibir datos provenientes del puerto COM de la computadora, recibir información de otros dispositivos y poder establecer una conexión entre Arduinos.

Para leer los datos provenientes del Serial Monitor es conveniente verificar primero si hay datos en el buffer, hay que hacer uso de la función Serial.available().

El uso de Serial.available() permite obtener el número de bytes disponibles para leer en el puerto serial. Estos son datos que ya llegaron y están almacenados en el buffer serial el cual es de 64 bytes.

Dentro de la casilla roja se debe escribir la información que se desee enviar al serial dandole click en el botón "Enviar".

El siguiente ejemplo imprime en pantalla un texto si hay información disponible en el buffer.

```

sketch_oct14a Arduino 1.8.9
Archivo Editar Programa Herramientas Ayuda
sketch_oct14a
void setup() {
    Serial.begin(9600);
}
void loop() {
    if(Serial.available()>0) {
        Serial.println("Hay informacion en el buffer");
    }
    delay(2000);
}

```

9.2.- Serial.read()

La función encargada de leer de byte en byte el buffer de 64bytes es `Serial.read()`, esta función cada vez que lea un byte del buffer lo limpia, por lo tanto, en caso de haber más de 1 byte en el buffer, la segunda vez que se utilice `Serial.read()` leerá el byte que previamente era el segundo, y así sucesivamente hasta limpiar el buffer.

Sintaxis: `Serial.read()`

Valor de retorno: El primer byte del buffer y lo limpia.

```

sketch_oct14a Arduino 1.8.9
Archivo Editar Programa Herramientas Ayuda
sketch_oct14a
char a;
void setup() {
    Serial.begin(9600);
}
void loop() {
    if(Serial.available()>0) {
        a = Serial.read();
        Serial.print(a);
        delay(500);
    }
}

```

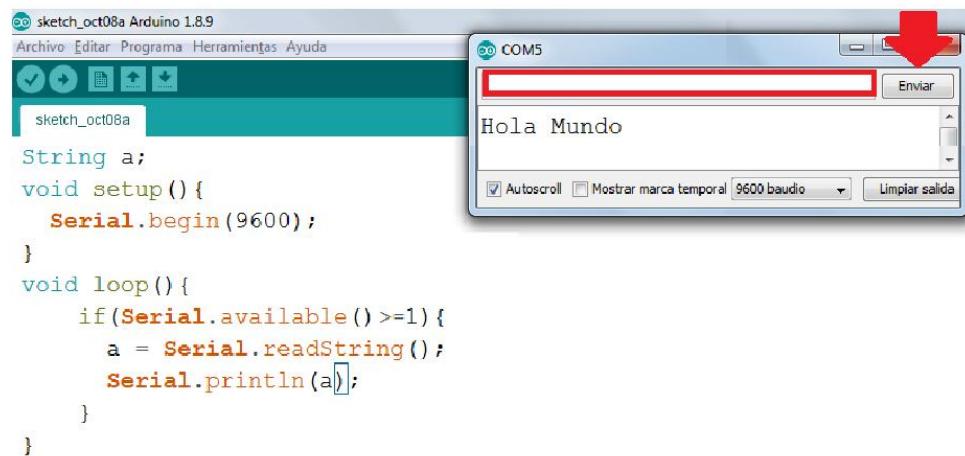
9.3.- Serial.readString()

Esta función a diferencia de Serial.read(), permite leer todos los bytes que contenga el buffer y lo limpia.

Sintaxis: Serial.readString()

Valor de retorno: String leído proveniente del serial.

En el siguiente ejemplo muestra el texto "Hola Mundo" que fue ingresado desde el serial monitor.



9.4.- Serial.find()

La función Serial.find() lee la información del buffer hasta que encuentre el objetivo o el texto que esté buscando. La función retornará "true" si encuentra el texto y "false" si no lo hace.

Sintaxis:

Serial.find(target)
Serial.find(target,length)

Parametros:

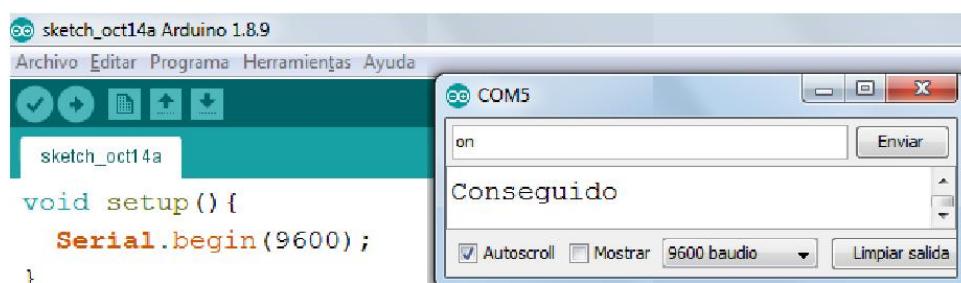
target: el texto que se desea buscar.

length: el tamaño del texto.

Valor de retorno:

true si encuentra el texto buscado, false en caso contrario.

El siguiente ejemplo muestra en el serial monitor el texto "conseguido" si se encuentra el texto "on" en el buffer, el texto "on" debe ser enviado al serial para poder realizar la búsqueda.



```
sketch_oct14a Arduino 1.8.9
Archivo Editar Programa Herramientas Ayuda
sketch_oct14a
void setup() {
    Serial.begin(9600);
}
void loop() {
    if(Serial.available()) {
        if(Serial.find("on")){
            Serial.println("Conseguido");
        }
    }
}
```

Conseguido

 Autoscroll Mostrar 9600 baudio Limpiar salida

9.5.- Serial.findUntil()

La función Serial.findUntil() lee la información del buffer hasta que encuentre el texto que esté buscando o hasta encontrar la terminación de la cadena, la función retornará "true" si encuentra el texto y "false" si no lo hace.

Sintaxis:

Serial.findUntil(target, terminal)

Parámetros:

target: el texto a buscar.

terminal: la finalización de la cadena buscada.

Valor de retorno:

true si encuentra el texto buscado, false en caso contrario.

La función Serial.findUntil() lee la información del buffer hasta que encuentre el texto que esté buscando o hasta encontrar la terminación de la cadena, la función retornará "true" si encuentra el texto y "false" si no lo hace.

Sintaxis:

Serial.findUntil(target, terminal)

Parámetros:

target: el texto a buscar.

terminal: la finalización de la cadena buscada.

Valor de retorno:

true si encuentra el texto buscado, false en caso contrario.

Este ejemplo busca la palabra "hola" hasta el salto de línea en el buffer del serial. Por lo que cada vez que se envíe "hola" al serial, este imprimirá "Encontrado".

```

sketch_oct14a Arduino 1.8.9
Archivo Editar Programa Herramientas Ayuda
sketch_oct14a §
void setup()
{
    Serial.begin(9600);
}
void loop()
{
    if(Serial.available())
    {
        if(Serial.findUntil("hola","\n")){
            Serial.println("Encontrado");
        }
    }
}

```

hola

Encontrado

Autoscroll Mostrar 9600 baudio Limpiar salida

9.6.- Serial.parseInt()

La función Serial.parseInt() devuelve el primer número entero válido del buffer y lo quita (los caracteres que no sean dígitos o el signo menos son ignorados). Si al cabo de 1 segundo no se ha encontrado un número entero válido en el buffer, por defecto retornará 0.

Sintaxis:

Serial.parseInt()
Serial.parseInt(lookahead)

Parámetros:

lookahead: valores permitidos:

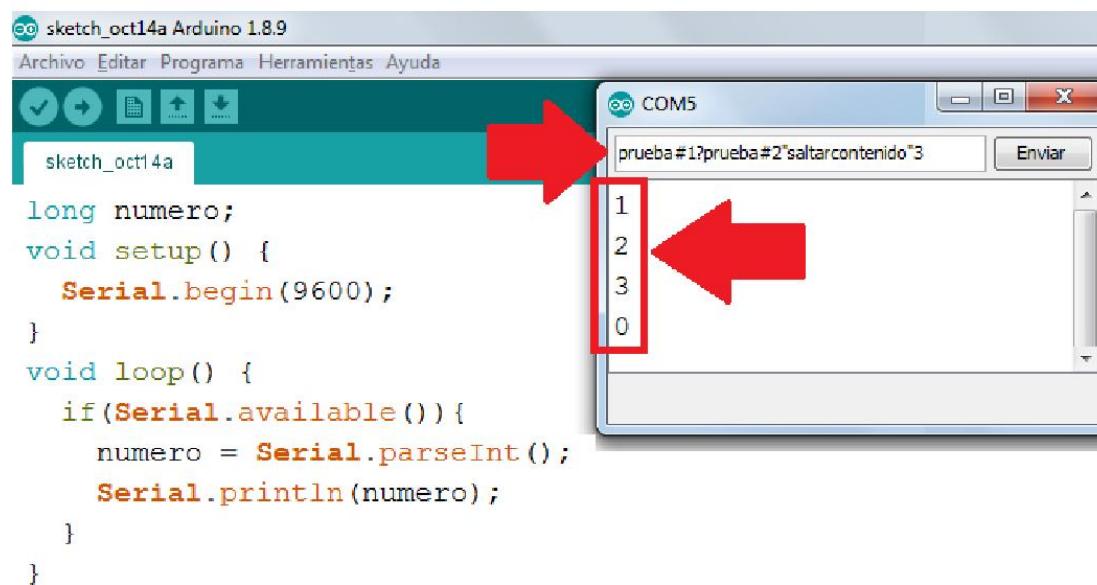
*SKIP_ALL: Todos los caracteres que no sean dígitos o el signo menos son ignorados (valor por defecto).

*SKIP_WHITESPACE: Solamente las tabulaciones, espacios y espaciados son ignorados.

Valor de retorno:

tipo de dato: long.

En el siguiente ejemplo se envía al serial monitor el texto "prueba#1?prueba#2'saltarcontenido'3", con el uso de Serial.parseInt() todo carácter que no sea un número es ignorado, por lo que la variable "numero" únicamente tomará los valores que se encuentren en el texto y los mostrará 1 por 1 en el serial monitor, al cabo de 1 segundo la función retorna el valor de 0 ya que no encontró más números disponibles para leer en el buffer.



9.7.- Serial.parseFloat()

La función Serial.parseFloat() devuelve el primer número flotante válido del buffer y lo quita. Si al cabo de 1 segundo por defecto retornará 0.

Sintaxis:

Serial.parseFloat()
Serial.parseFloat(lookahead)

Parámetros:

lookahead: valores permitidos:

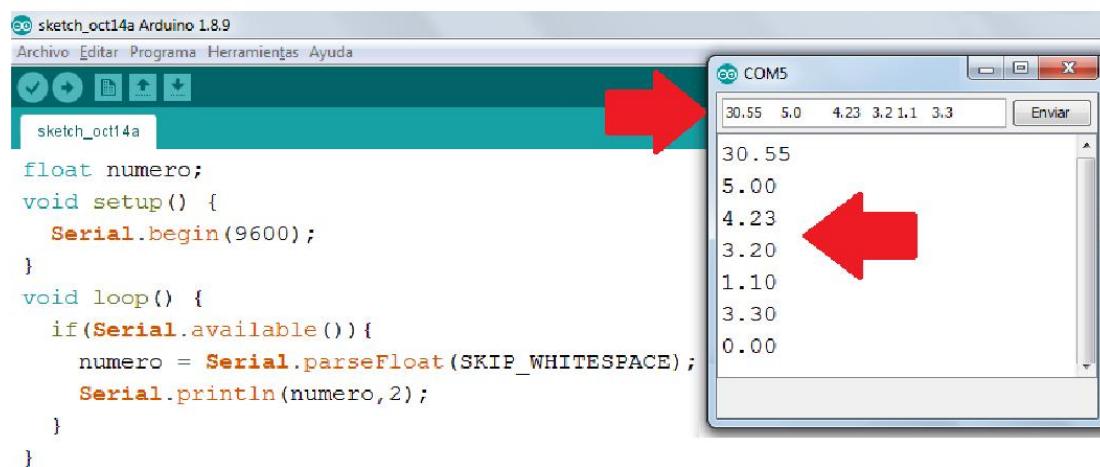
*SKIP_ALL: Todos los caracteres que no sean dígitos o el signo menos son ignorados (valor por defecto).

*SKIP_WHITESPACE: Solamente las tabulaciones, espacios y espaciados son ignorados.

Return:

Tipo de dato: float.

A diferencia de Serial.parseInt(), Serial.parseFloat() lee los valores de tipo float disponibles en el buffer, con el uso de SKIP_WHITESPACE las tabulaciones y espacios son ignorados por lo que solamente toma los valores de números decimales. Al cabo de 1 segundo retorna 0 ya que no encontró más valores disponibles en el buffer que leer.



The screenshot shows the Arduino IDE interface. On the left, the code for 'sketch_oct14a' is displayed:

```
float numero;
void setup() {
    Serial.begin(9600);
}
void loop() {
    if(Serial.available()) {
        numero = Serial.parseFloat(SKIP_WHITESPACE);
        Serial.println(numero,2);
    }
}
```

On the right, the 'COM5' serial monitor window shows the output of the program. Red arrows point from the code to the first two lines of the monitor output, which are '30.55' and '5.00'. The monitor also displays other values: 4.23, 3.20, 1.10, 3.30, and 0.00.

Output
30.55
5.00
4.23
3.20
1.10
3.30
0.00

Capítulo 10. ENTRADAS DIGITALES

10.1.- Introducción

Una de las funciones más interesantes de Arduino es su capacidad de poder entender su entorno con el mundo físico, gracias a que puede obtener lecturas de mediciones de voltaje y de sensores. Estas lecturas son obtenidas a través de un PIN del Arduino.

Una señal digital es una variación de voltaje entre 0 y +VCC sin pasar por valores intermedios. Por lo tanto, una señal digital dispone de 2 estados los cuales para Arduino son HIGH y LOW.

Estas 2 lecturas son suficientes para trabajar, ya que puede indicar si un motor esta encendido o apagado, si algún otro dispositivo electrónico está funcionando correctamente o para leer los valores de un sensor ultrasónico o de temperatura.

Si se quiere leer el valor de un pin digital hay que prevenir lecturas flotantes, esto sucede cuando se obtiene la lectura de un circuito abierto. Para ello se emplea el uso de una resistencia ya sea pull-down o pull-up, el cual le dará un estado conocido a la entrada cuando se quiera leer la entrada proveniente de un switch y este esté abierto o cuando un botón no esté siendo pulsado.

Una resistencia de 10K ohm es usualmente utilizada para estos casos, es un valor suficientemente pequeño para prevenir entradas "flotantes" y a la misma vez posee un valor suficientemente alto para no tomar mucha corriente del switch cuando este esté cerrado.

Si se usa una resistencia pull-down, el valor de entrada será LOW cuando el switch esté abierto y HIGH para cuando el switch esté cerrado.

Si se usa una resistencia pull-up, el valor de entrada será HIGH cuando el switch esté abierto y LOW para cuando el switch esté cerrado.

10.2.- Digital Read

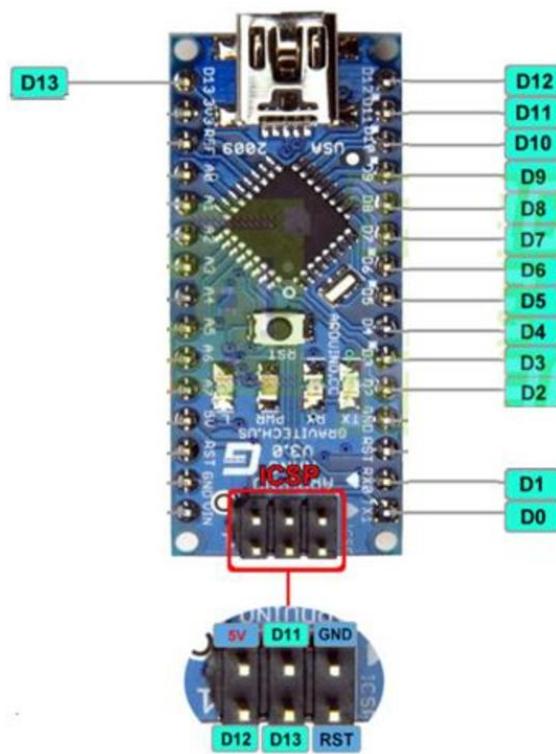
Como se explicó en capítulos anteriores, las placas Arduino poseen pines digitales, identificados con el código DX, donde X es el número del pin. La función por defecto de Arduino que se encarga de leer el valor de un pin digital es `digitalRead()`, el cual retorna HIGH o LOW dependiendo del estado actual del PIN especificado.

Sintaxis: `digitalRead(pin)`

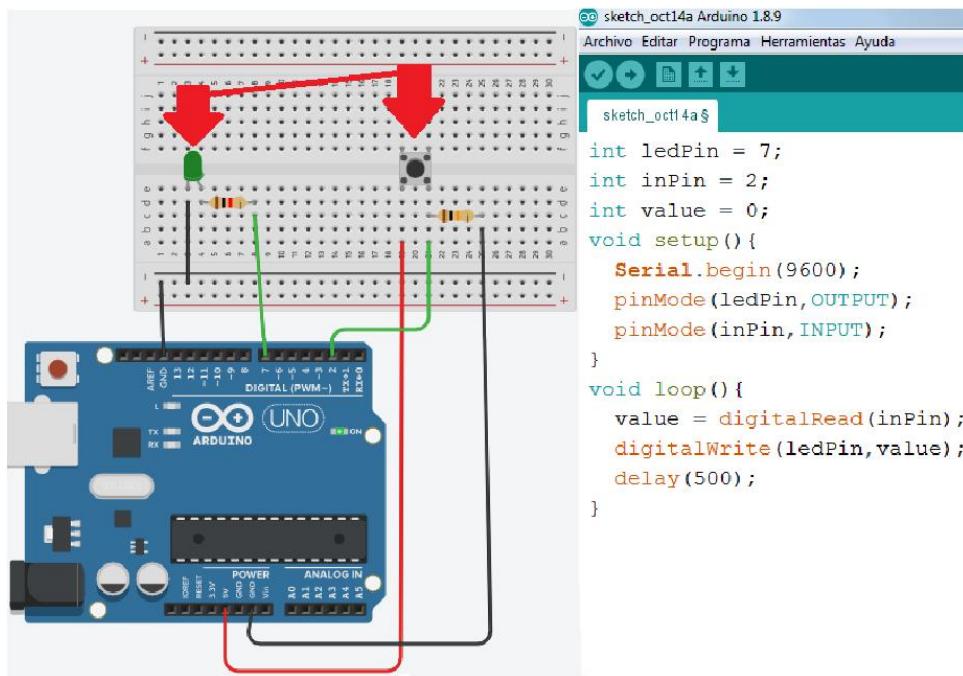
Parámetros

pin: el número del pin digital que se quiere leer.

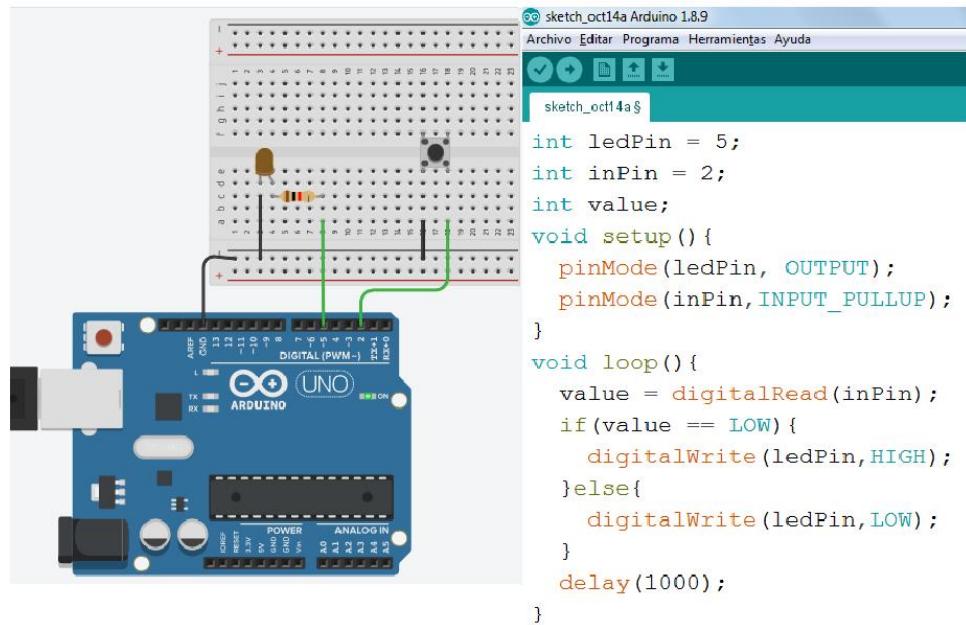
Valor de retorno: HIGH o LOW



El siguiente ejemplo enciende un LED conectado al pin 7 cada vez que se pulse el botón conectado al pin 2.



El siguiente ejemplo hace uso de la resistencia PULLUP interna del Arduino y se cambia la lógica del programa para cuando el pin este en "LOW" encienda el LED.



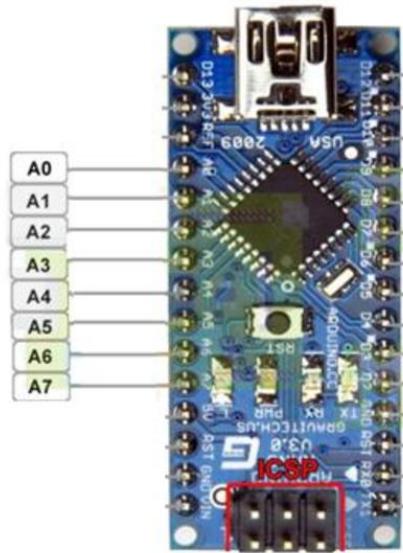
Capítulo 11. ENTRADAS ANALÓGICAS

11.1.- Introducción

Las entradas analógicas funcionan de manera similar a las entradas digitales, por lo que en montaje y código podría ser muy similar. Sin embargo, en ciertos aspectos son muy distintos.

Su principal diferencia es que una señal analógica es una magnitud que puede tomar cualquier valor entre el intervalo de 0V y +VCC. Por ejemplo, una señal analógica para un intervalo entre 0V Y 5V puede valer 2,35V o cualquier otro valor. Por norma general, las entradas analógicas son más escasas y más lentas.

La precisión de la medición de una entrada analógica se logra con el funcionamiento de un conversor analógico digital (ADC), que convierte una medición analógica en una medición digital codificada con número N de bits.



11.2.- Analog Read

La función que se encarga de leer el valor de un pin analógico en Arduino es `analog.read()`, dicha función retornará el valor del PIN especificado.

La placa Arduino contiene un convertidor analógico a digital (ADC) de 10 bits. Esto significa que mapeará voltajes de entrada de entre 0V y 5V en valores enteros entre 0 y 1023, por lo tanto, lee 4.9 mV por cada valor de los 1024.

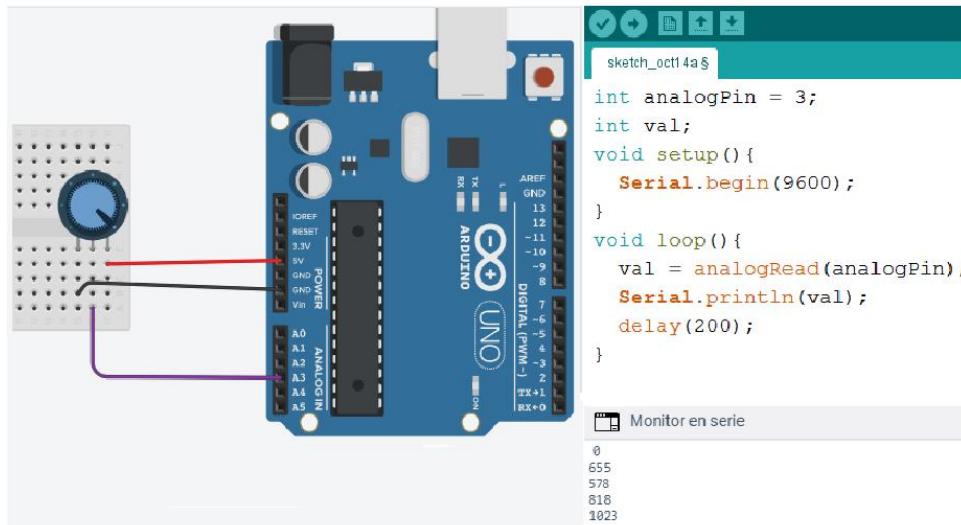
Arduino tarda unos 100 microsegundos (0.0001 s) para leer una entrada analógica, por lo que la velocidad de lectura máxima es de alrededor de 10.000 veces por segundo.

Sintaxis: `analogRead(pin)`

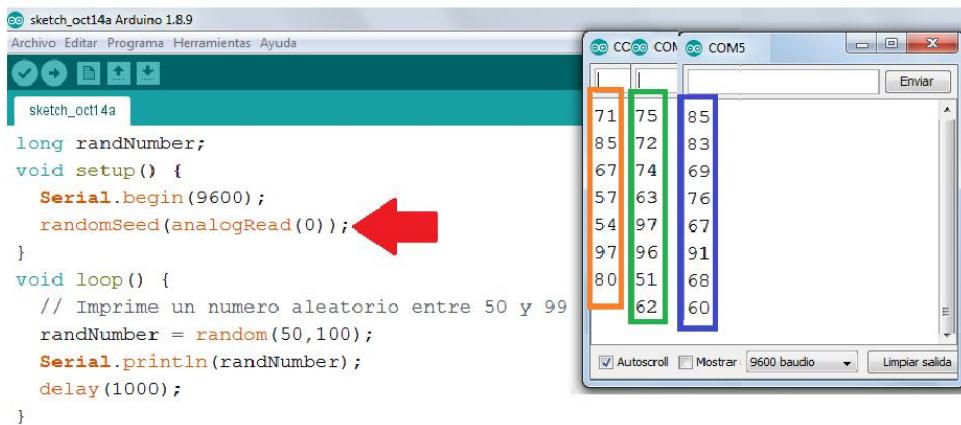
Parámetros: el número del pin analógico que se quiere leer, en la mayoría de las placas son desde A0 hasta A5.

Valor de retorno: La lectura del pin (valores entre 0 y 1023).

El siguiente ejemplo muestra como variando la perilla de un potenciómetro cambia el valor leído en el pin analógico donde su menor valor es 0 para cuando la perilla esté al mínimo y 1023 cuando se gire totalmente la perilla en sentido contrario.



En el siguiente ejemplo se hace uso de `analogRead(pin)` para generar secuencias de números aleatorios los cuales difieren en cada ejecución del programa. Esto se logra inicializando `randomSeed()` con el valor leído de un pin analógico que no está conectado a nada, de esta manera su valor no es constante:



Capítulo 12. CONDICIONES. PARTE 2

12.1.- Operadores Lógicos

Los operadores lógicos proporcionan un resultado a partir de que se cumpla o no una cierta condición, estos producen un resultado booleano, y sus operandos son también valores lógicos o asimilables a ellos (los valores numéricos son asimilados a cierto o falso según su valor sea cero o distinto de cero).

Signo	Descripción
!	Negación lógica
&&	Y lógico
	O lógico

El siguiente ejemplo hace uso de los operadores lógicos, y en caso de que la condición considerada sea verdadera imprime un texto de a qué operador lógico pertenece.

```

sketch_oct14a Arduino 1.8.9
Archivo Editar Programa Herramientas Ayuda
sketch_oct14a
long numero1,numero2,numero3,numero4;
void setup(){
    Serial.begin(9600);
}
void loop(){
    numero1 = random(0,15);
    numero2 = random(19,25);
    numero3 = random(4,6);
    numero4 = random(2,4);
    if(numero1 > 10 && numero2 > 20){
        Serial.println("Operador Logico &&");
    }
    if(numero3 != 5 || numero4 == 3){
        Serial.println("Operador Logico ||");
    }
    delay(1000);
}

```

12.2.- If/else If/else

La estructura if/else if/else permite tener un mayor control en el flujo de código que con solamente if/else, permitiendo múltiples tareas ser agrupadas.

La ejecución del programa continuará a la siguiente condición hasta que alguna sea verdadera, en caso de que una condición sea verdadera se ejecutará su bloque de código y saldrá de toda la construcción if/else if/else, si ninguna condición es verdadera el bloque de código de else será ejecutado.

Sintaxis

```
if(condicion1){  
    //Instrucciones...  
}  
else if(condicion2){  
    //Instrucciones...  
}  
else{  
    //Instrucciones  
}
```

The screenshot shows the Arduino IDE interface. On the left, the code for 'sketch_oct14a' is displayed:

```

long number1, number2, resultado;
void setup() {
    Serial.begin(9600);
    randomSeed(5);
}
void loop() {
    number1 = random(50);
    number2 = random(50);
    Serial.print("El resultado es: ");
    if(number1 > 40 || number2 < 20){
        resultado = number1 * number2;
        Serial.println(resultado);
    }else if(number1 == number2){
        resultado = number1 + number2;
        Serial.println(resultado);
    }else{
        resultado = number1 - number2;
        Serial.println(resultado);
    }
    delay(1000);
}

```

On the right, the 'COM5' serial monitor window shows the output of the program:

```

El resultado es: -10
El resultado es: -22
El resultado es: 78
El resultado es: -29
El resultado es: 168
El resultado es: -28
El resultado es: 220

```

The serial monitor also includes settings at the bottom: Autoscroll (checked), Mostrar marca temporal (unchecked), 9600 baudio, and Limpiar salida.

12.3.- Switch...case

Esta estructura como la anterior, controla el flujo del programa permitiéndole a los programadores especificar líneas de código que debe ser ejecutado en diferentes condiciones.

En particular, Switch compara el valor de una variable con el valor especificada en cada caso, cuando un caso es encontrado, el código dentro de él se ejecutará.

La palabra “break” es utilizada para poder salir de cada caso, típicamente se utiliza al final de cada uno, en caso de no estar “break” el código de los casos siguientes se ejecutarán.

Datos permitidos: int, char.

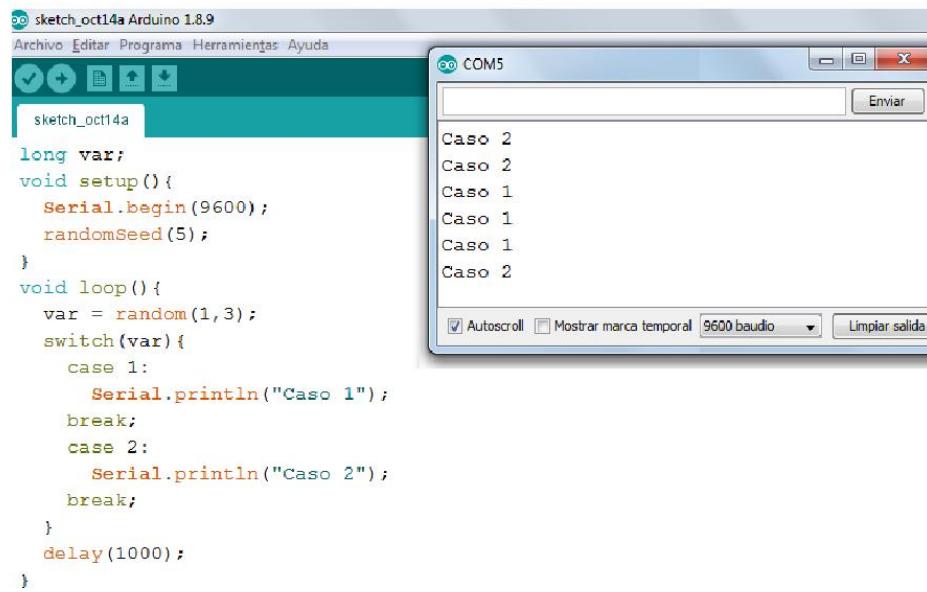
Parámetros:

var: variable cuyo valor se comparará con el de los casos.

case 1, case 2: constantes, cuyos valores se compararán con el de la variable.

Sintaxis

```
switch(var) {  
    case 1:  
        //Instrucciones...  
        break;  
    case 2:  
        //Instrucciones...  
        break;  
    default:  
        //Instrucciones...  
        break;  
}
```



```

sketch_oct14a Arduino 1.8.9
Archivo Editar Programa Herramientas Ayuda
sketch_oct14a
long var;
void setup(){
    Serial.begin(9600);
    randomSeed(5);
}
void loop(){
    var = random(1,3);
    switch(var){
        case 1:
            Serial.println("Caso 1");
            break;
        case 2:
            Serial.println("Caso 2");
            break;
    }
    delay(1000);
}

```

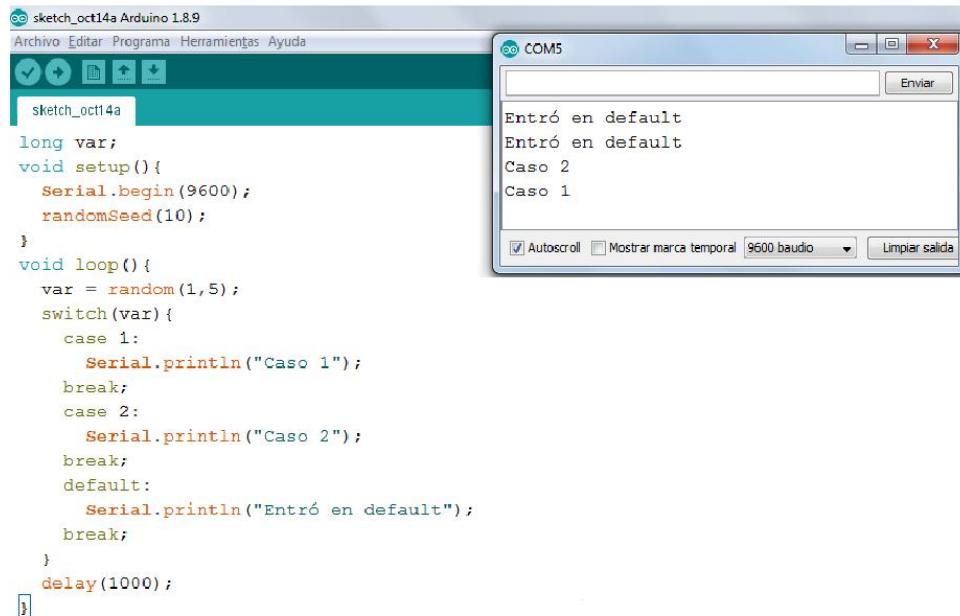
The serial monitor window shows the output:

```

Caso 2
Caso 2
Caso 1
Caso 1
Caso 1
Caso 2

```

Este ejemplo hace uso de default para cuando no encuentre un caso.



```

sketch_oct14a Arduino 1.8.9
Archivo Editar Programa Herramientas Ayuda
sketch_oct14a
long var;
void setup(){
    Serial.begin(9600);
    randomSeed(10);
}
void loop(){
    var = random(1,5);
    switch(var){
        case 1:
            Serial.println("Caso 1");
            break;
        case 2:
            Serial.println("Caso 2");
            break;
        default:
            Serial.println("Entró en default");
            break;
    }
    delay(1000);
}

```

The serial monitor window shows the output:

```

Entró en default
Entró en default
Caso 2
Caso 1

```

Capítulo 13. OPERACIONES CON STRING

13.1.- Métodos de String

Los siguientes métodos de String permiten alterar, manipular y modificar el String dependiendo del método que se utilice.

Función	Descripción	Retorno
myString.concat(parameter)	Concatena un String con otro.	true o false
myString.length()	Retorna el tamaño de caracteres que posee el String.	El tamaño del String.
MyString.remove(index) MyString.remove(index,count)	Modifica el String removiendo los caracteres a partir del índice que se indique hasta el final del String o hasta cuantos caracteres borrar a partir del índice.	Nada, modifica directamente el String.
MyString.substring(from) MyString.substring(from,to)	Obtiene un substring proveniente de un String. El índice del comienzo es obligatorio pero el índice del final es opcional. Si el índice del final es omitido, el substring continuará hasta el final del String.	El substring, tipo de dato: String.
myString.replace(substring1, substring2)	Permite reemplazar un String dentro del suministrado, por otro String establecido por parámetros.	Nada, modifica directamente el String.

Ejemplo

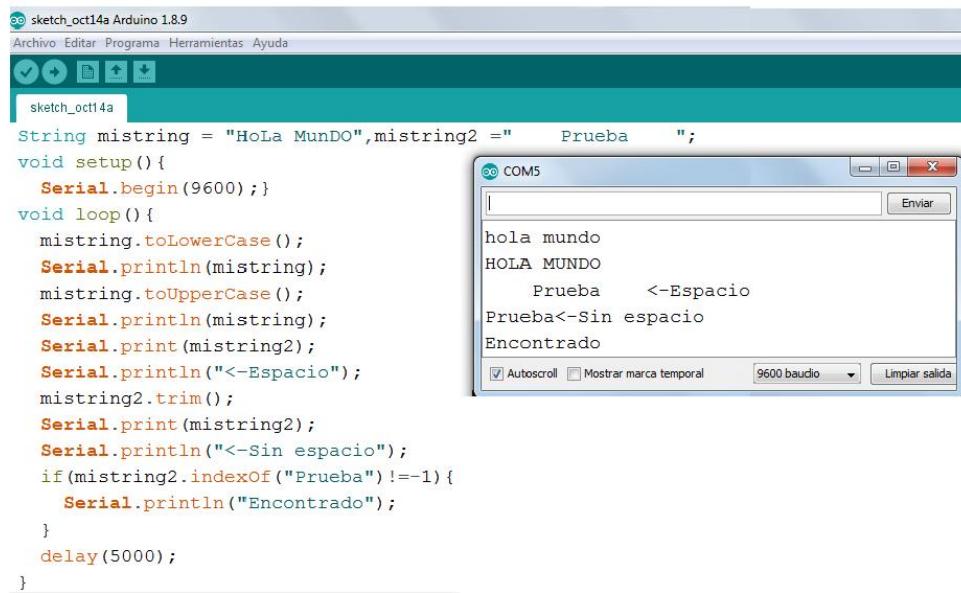
```

String mistring = "Hola Mundo", sub_string;
void setup() {
    Serial.begin(9600);
}
void loop() {
    mistring.concat("+Texto");
    Serial.println(mistring);
    Serial.print("Tamaño: ");
    Serial.println(mistring.length());
    mistring.remove(4);
    Serial.println(mistring);
    sub_string = mistring.substring(0,3);
    Serial.println(sub_string);
    mistring.replace(mistring, "Adios");
    Serial.println(mistring);
    delay(5000);
}

```

Función	Descripción	Retorno
myString.toLowerCase()	Obtiene la versión en minúsculas de un String.	Modifica directamente el String.
myString.toUpperCase()	Obtiene la versión en mayúsculas de un String.	
myString.trim()	Remueve los espacios al comienzo y los del final de este.	
myString.indexOf(val)	Encuentra un carácter o un String en otro String. Por defecto, realiza la búsqueda desde el comienzo del String, pero se puede indicar un índice para empezar desde ese lugar.	El índice donde encontró el String o -1 si no encuentra nada.
myString.indexOf(val, from)		

Ejemplo



```

sketch_oct14a Arduino 1.8.9
Archivo Editar Programa Herramientas Ayuda

sketch_oct14a

String mistring = "HoLa Mundo",mistring2 = "      Prueba      ";
void setup() {
    Serial.begin(9600);
}
void loop() {
    mistring.toLowerCase();
    Serial.println(mistring);
    mistring.toUpperCase();
    Serial.println(mistring);
    Serial.print(mistring);
    Serial.println("<-Espacio");
    mistring2.trim();
    Serial.print(mistring2);
    Serial.println("<-Sin espacio");
    if(mistring2.indexOf("Prueba") != -1){
        Serial.println("Encontrado");
    }
    delay(5000);
}

```

hola mundo
HOLA MUNDO
Prueba <-Espacio
Prueba<-Sin espacio
Encontrado

13.2. Métodos de conversión

Los siguientes métodos convierten una cadena de caracteres (String) en un valor de otro tipo de dato, por ejemplo, int o float:

Función	Descripción	Retorno
myString.toInt()	Convierte un String valido a un entero.	Tipo de dato long o 0 si el String no es válido.
myString.toFloat()	Convierte un String valido a un tipo de dato float.	Tipo de dato float o 0 si el String no es válido.
myString.toDouble()	Convierte un String valido a un tipo de dato double.	Tipo de dato double o 0 si el String no es válido.
myString.toCharArray(buf, len)	Copia el String a un arreglo de char.	Nada.

Ejemplo

The screenshot shows the Arduino IDE interface. The top menu bar includes 'Archivo', 'Editar', 'Programa', 'Herramientas', and 'Ayuda'. Below the menu is a toolbar with icons for file operations. The main area displays the code for 'sketch_oct14a'. The code uses strings to store values, converts them to floats and doubles, and prints them to the serial port. The 'Serial' library is used for communication. The 'loop()' function contains code to read from the serial port and print the results. The 'Serial.println()' command is used to output the data. The 'delay(4000);' command is included to prevent the loop from running too quickly. To the right of the code is a 'Serial Monitor' window titled 'COM5'. It shows the output of the program: '144.22', '144.22', '2434', and 'String a char'. There are also several checkboxes at the bottom of the monitor window.

```
String mistring1 = "144.22", mistring2 = "2434";
float flotante;
double vardouble;
long entero;
char arreglo_char[20];
void setup(){
    Serial.begin(9600);
}
void loop(){
    flotante = mistring1.toFloat();
    Serial.println(flotante);
    vardouble = mistring1.toDouble();
    Serial.println(vardouble);
    entero = mistring2.toInt();
    Serial.println(entero);
    delay(4000);
}
```

13.3.- Métodos Char

Los siguientes métodos pertenecen a los tipos de datos char y sirven para analizar qué carácter posee una variable char.

Función	Descripción	Retorno
isAlpha(thisChar)	Analiza si un char es una letra	true o false
isAlphaNumeric(thisChar)	Analiza si un char es alfanumérico (letra o un numero)	true o false
isLowerCase(thisChar)	Analiza si un char está en minúsculas	true o false
isUpperCase(thisChar)	Analiza si un char está en mayúsculas	true o false
isDigit(thisChar)	Analiza si un char es un numero	true o false
IsSpace(tischar)	Analiza si el char es un espacio en blanco ('\n', '\r', '\t')	true o false

Ejemplo

```

sketch_oct14a Arduino 1.8.9
Archivo Editar Programa Herramientas Ayuda
sketch_oct14a
char dato = 'A';
void setup() {
    Serial.begin(9600);
}
void loop() {
    Serial.println(isAlpha(dato));
    Serial.println(isAlphaNumeric(dato));
    Serial.println(isDigit(dato));
    Serial.println(isLowerCase(dato));
    Serial.println(isUpperCase(dato));
    Serial.println(isSpace(dato));
    delay(50000);
}

```

Capítulo 14. CICLOS. PARTE 1

14.1.- For

Es usado para repetir un bloque de código una cantidad de veces determinada. Un contador se incrementa para indicar la cantidad de veces que se ejecutara el ciclo y se detiene cuando una condición es falsa. For es útil para cualquier función repetitiva.

Sintaxis:

```
for(inicializacion;condicion;incremento) {  
    // Instrucciones  
}
```

Parámetros:

inicialización: sucede primero y solo 1 vez.

condición: condición que cada vez al empezar el ciclo se verifica, si esta es verdadera (true), el bloque de instrucciones y el incremento se ejecutarán, entonces la condición es verificada otra vez. Cuando la condición es falsa (false), el ciclo termina.

incremento: se ejecuta cada vez que la condición del ciclo sea verdadera (true).

```

sketch_oct08a Arduino 1.8.9
Archivo Editar Programa Herramientas Ayuda
sketch_oct08a
int i = 0;
void setup() {
    Serial.begin(9600);
}
void loop() {
    for(i=0;i<5;i++) {
        Serial.print("Ciclo numero: ");
        Serial.println(i+1);
    }
    delay(5000);
}

```

COM4

Ciclo numero: 1
Ciclo numero: 2
Ciclo numero: 3
Ciclo numero: 4
Ciclo numero: 5

Enviar

Autoscroll Retorno de carro 9600 baudio Limpiar salida

For también tiene la capacidad de hacer ciclos decrementativos, de igual forma este evaluará la condición antes de empezar un nuevo ciclo.

```

sketch_oct14a Arduino 1.8.9
Archivo Editar Programa Herramientas Ayuda
sketch_oct14a
int i;
void setup() {
    Serial.begin(9600);
}
void loop() {
    Serial.println("Cuenta regresiva:");
    for(i=10;i>=0;i--) {
        Serial.println(i);
        delay(1000);
    }
    delay(2000);
}

```

COM5

Cuenta regresiva:

10
9
8
7
6
5
4
3
2
1
0

Enviar

Autoscroll Mostrar marca temporal 9600 baudio Limpiar salida

Capítulo 15. CICLOS. PARTE 2

15.1.- While

Este ciclo se repetirá continuamente e infinitamente, hasta que la expresión entre los paréntesis () sea falsa (false). Algo debe cambiar el valor de la variable que se verifica la expresión o el ciclo nunca terminará. Esto debe estar en el código, tal como una variable que se incrementa o una condición externa como un sensor.

Sintaxis:

```
while(condicion){  
    //Instrucciones  
}
```

Parámetros: condición que se evalúa entre verdad o falso (true o false).

El siguiente ejemplo muestra como usar el ciclo while mientras una variable mantiene un valor menor a 12:

```
sketch_oct08a Arduino 1.8.9  
Archivo Editar Programa Herramientas Ayuda  
sketch_oct08a  
int i = 0;  
void setup() {  
    Serial.begin(9600);  
}  
void loop(){  
    while(i<12){  
        Serial.print("Ciclo numero: ");  
        Serial.println(i+1);  
        i++;  
    }  
    delay(5000);  
}
```

COM4
Enviar
Ciclo numero: 1
Ciclo numero: 2
Ciclo numero: 3
Ciclo numero: 4
Ciclo numero: 5
Ciclo numero: 6
Ciclo numero: 7
Ciclo numero: 8
Ciclo numero: 9
Ciclo numero: 10
Ciclo numero: 11
Ciclo numero: 12

Este ejemplo utiliza while junto a Serial.available() para verificar si hay información en el buffer, en caso de que haya, entrará en el ciclo y mostrará lo que contenga el buffer en el serial monitor.



```
sketch_oct14a Arduino 1.8.9
Archivo Editar Programa Herramientas Ayuda
sketch_oct14a
String mistring;
void setup() {
    Serial.begin(9600);
}
void loop() {
    while(Serial.available()){
        mistring = Serial.readString();
        Serial.println(mistring);
    }
}
```

COM5

While con Serial.available()

Enviar

Autoscroll Mostrar marca temporal 9600 baudio Limpiar salida

15.2.- Do...while

Este ciclo funciona de la misma manera que lo hace while, con la excepción de que la condición es verificada al final del ciclo, por lo tanto, este ciclo se ejecuta por lo menos 1 vez. La sintaxis general del ciclo do while es la siguiente:

```
do{
    //instrucciones
} while(condicion);
```

Donde "condición" es una expresión lógica que va a evaluarse.

El siguiente ejemplo muestra como se ejecuta un ciclo mientras que la variable "i" sea menor que 10:

The screenshot shows the Arduino IDE interface. The top menu bar includes 'Archivo', 'Editar', 'Programa', 'Herramientas', and 'Ayuda'. Below the menu is a toolbar with icons for save, upload, and refresh. The main window displays a code editor with the following sketch:

```
sketch_oct08a Arduino 1.8.9
Archivo Editar Programa Herramientas Ayuda
sketch_oct08a.ino

int i = 0;
void setup() {
    Serial.begin(9600);
}
void loop() {
    do{
        Serial.print("Ciclo numero: ");
        serial.println(i+1);
        i++;
    }while(i<10);
    delay(5000);
}
```

To the right of the code editor is a 'Serial Monitor' window titled 'COM4'. It has a text input field and a 'Enviar' (Send) button. The text area contains the following output:

Ciclo numero: 1
Ciclo numero: 2
Ciclo numero: 3
Ciclo numero: 4
Ciclo numero: 5
Ciclo numero: 6
Ciclo numero: 7
Ciclo numero: 8
Ciclo numero: 9
Ciclo numero: 10