



Arduino. Nivel II

diciembre, 2019



Objetivos del nivel

- Aprender a crear y utilizar procedimientos y funciones reusables.
- Crear librerías personalizadas.
- Manipular sensores de diferentes tipos.
- Controlar el comportamiento de objetos reales, tales como: relés, motores, entre otros.

Prerrequisitos del nivel

- Arduino Nivel I

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. Funciones. Parte 1

- 1.1.- Funciones.
- 1.2.- Ventajas de Utilizar Funciones.
- 1.3.- Llamado de Las Funciones.

Capítulo 2. Funciones. Parte 2

- 2.1.- Variables Globales y Locales.
- 2.2.- Funciones Con Parámetros.
- 2.3.- Parámetros Por Referencia.

Capítulo 3. Relés

- 3.1.- Introducción.
- 3.2.- Conexión Del Relé al Arduino.
- 3.3.- Modulo de Relé Para Arduino.

Capítulo 4. Funciones. Parte 3

- 4.1.- Funciones Con Valor de Retorno.
- 4.2.- Llamado de Funciones.

Capítulo 5. Librerías

- 5.1.- Librerías.
- 5.2.- Instalar Librería Con el Gestor de Librerías.
- 5.3.- Instalar Una Librería a Través de un .zip.

Capítulo 6. Lcd. Parte 1

- 6.1.- Liquid Crystal Display.
- 6.2.- Contraste Del Lcd.
- 6.3.- Conexión de Datos Del Lcd.

Capítulo 7. Lcd. Parte 2

- 7.1.- Librería Lcd.
- 7.2.- Inicializar la Pantalla.
- 7.3.- Enviando Información.

Capítulo 8. Arreglos. Parte 1

- 8.1.- Arreglos.
- 8.2.- Acceso a Elementos.
- 8.3.- Recorrer Arreglos.

Capítulo 9. Memoria EEPROM

- 9.1.- Introducción.
- 9.2.- Leer la Memoria Eeprom.
- 9.3.- Escribir la Memoria Eeprom.
- 9.4.- Operador Eeprom.

Capítulo 10. Interrupciones

- 10.1.- Introducción.
- 10.2.- Interrupciones en Arduino.
- 10.3.- Digitalpintointerrupt.
- 10.4.- Consideraciones Con Las Interrupciones.

Capítulo 11. Arreglos. Parte 2

- 11.1.- Matrices.
- 11.2.- Declaración de Matrices.
- 11.3.- Uso de Una Matriz.

Capítulo 12. Teclados Matriciales

- 12.1.- Introducción.
- 12.2.- Conexión.
- 12.3.- Programación.

Capítulo 13. Sensores. Parte 1

- 13.1.- Introducción.
- 13.2.- Sensor de Temperatura Tmp36.
- 13.3.- Sensor de Temperatura Dht22.

Capítulo 14. Sensores. Parte 2

- 14.1.- Sensor de Movimiento.
- 14.2.- Conexión.
- 14.3.- Programación.



Capítulo 1. FUNCIONES. PARTE 1

1.1.- Funciones

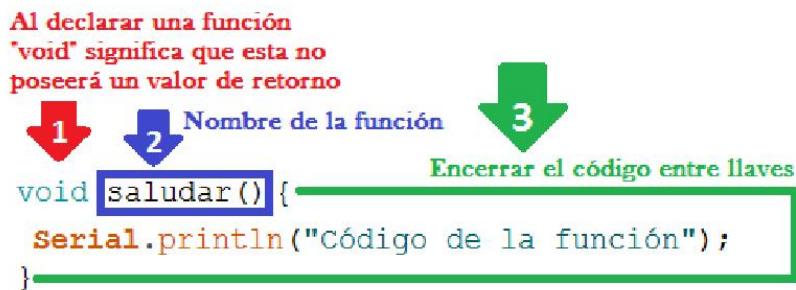
Una función es un conjunto de instrucciones que ejecutan una tarea determinada. Este conjunto de instrucciones puede ser ejecutadas al llamar el nombre de la función donde estas se encuentren.

Segmentar código en funciones permite al programador crear piezas modulares que realicen una tarea definida y que retornen al código del área de donde fue llamada. El caso típico para crear funciones es cuando se necesita usar la misma acción en diferentes partes del programa.

Para ejecutar un código en Arduino hay 2 funciones obligatorias: `setup()` y `loop()`. Por otro lado, `analogWrite()`, `digitalWrite()`, `analogRead()` y `digitalRead()` también son funciones que posee Arduino por defecto.

En caso de que Arduino no tenga una función que cubra las necesidades del programador, ésta debe ser creada.

Una función se crea de la siguiente manera:



Las partes de una función son:

- 1.- Se indica el tipo de dato del valor de retorno de la función, en caso de que esta no tenga valor de retorno, se debe declarar de tipo "void".
- 2.- Se le asigna un nombre a la función junto a paréntesis (), dentro los paréntesis van los parámetros que reciba la función, en caso de que los tenga.
- 3.- Se abren llaves ({}) para encerrar el código de la función.

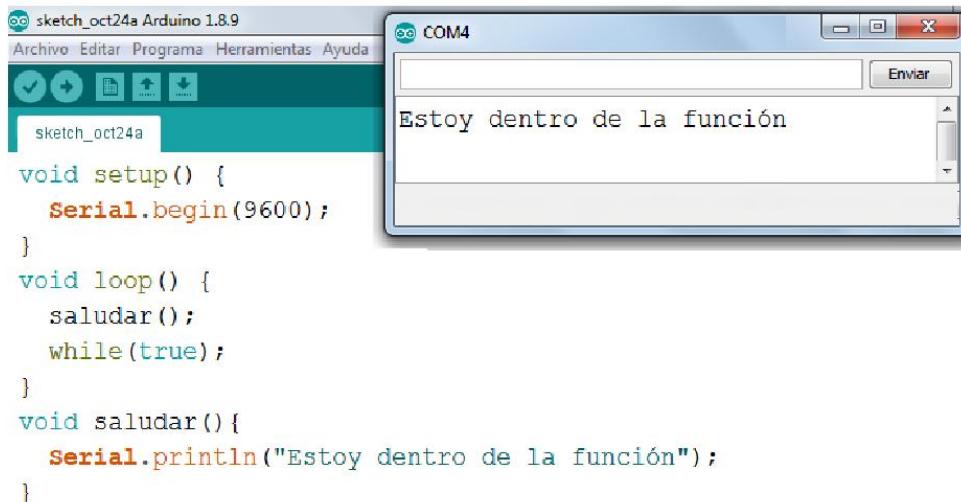
1.2.- Ventajas de Utilizar Funciones

Utilizar funciones tiene muchas ventajas:

- Ayudan al programador a ser organizado. Esto ayuda a conceptualizar el programa.
- Las funciones codifican una acción en un lugar, así que la función tiene que ser creada y depurada solo una vez.
- Reduce las probabilidades de error en la modificación, en caso que el código necesite ser cambiado.
- Las funciones hacen el sketch más pequeño y más compacto ya que pueden ser reusadas muchas veces.
- Hacen más fácil reutilizar el código en otros programas, ya que son modulares, además, también hacen el código más legible.

1.3.- Llamado de Las Funciones

El siguiente ejemplo muestra un mensaje en el Serial Monitor. Este mensaje se encuentra dentro de la función saludar(), por lo tanto, cada vez que se llame la función, se ejecutará el código que esté dentro de ella.



```
sketch_oct24a Arduino 1.8.9
Archivo Editar Programa Herramientas Ayuda
sketch_oct24a
void setup() {
    Serial.begin(9600);
}
void loop() {
    saludar();
    while(true);
}
void saludar(){
    Serial.println("Estoy dentro de la función");
}
```

COM4

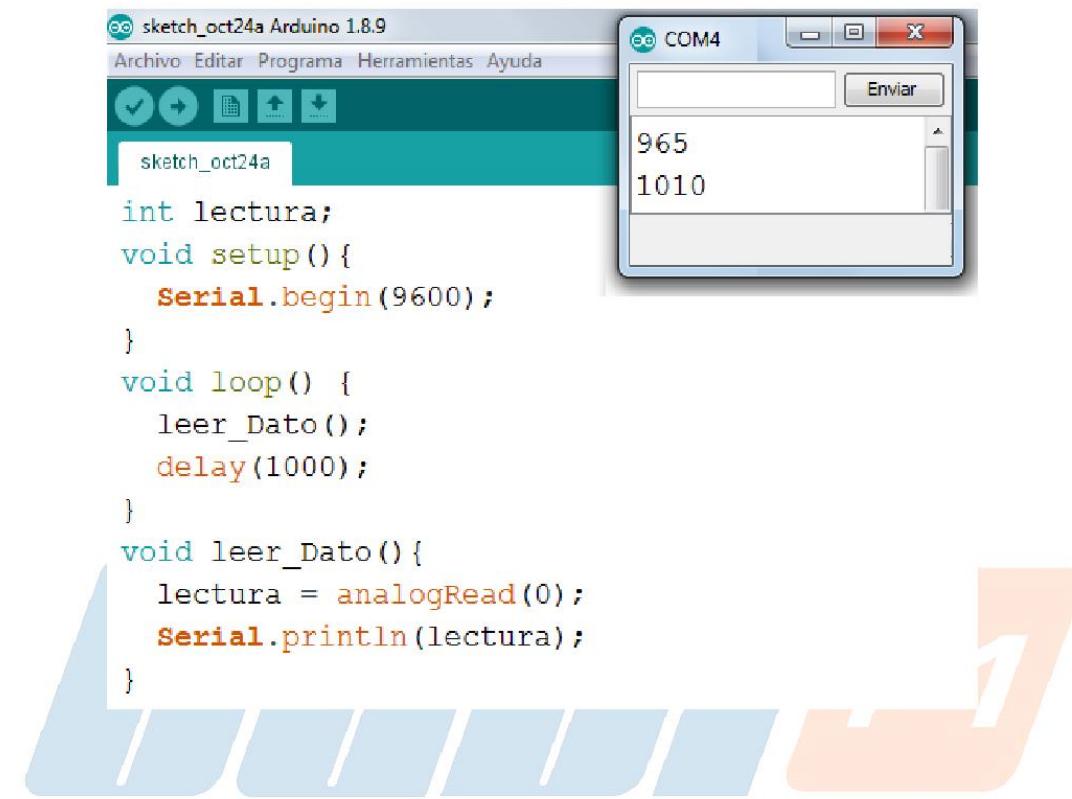
Enviar

Estoy dentro de la función

Cuando una función se declara no se ejecuta hasta que no sea llamada. A diferencia de las funciones "setup" y "loop" que son llamadas automáticamente al ejecutar el programa. Para llamar o ejecutar una función, se coloca su nombre (tal cual como se definió), seguido de un par de paréntesis.

Las funciones pueden estar definidas después de donde son llamadas. Es buena práctica dejar al inicio del programa las funciones "setup" y "loop", para luego dejar el desarrollo del resto de las funciones.

El siguiente ejemplo muestra en el Serial Monitor la lectura del pin analógico A0 al llamar la función leer_Dato():

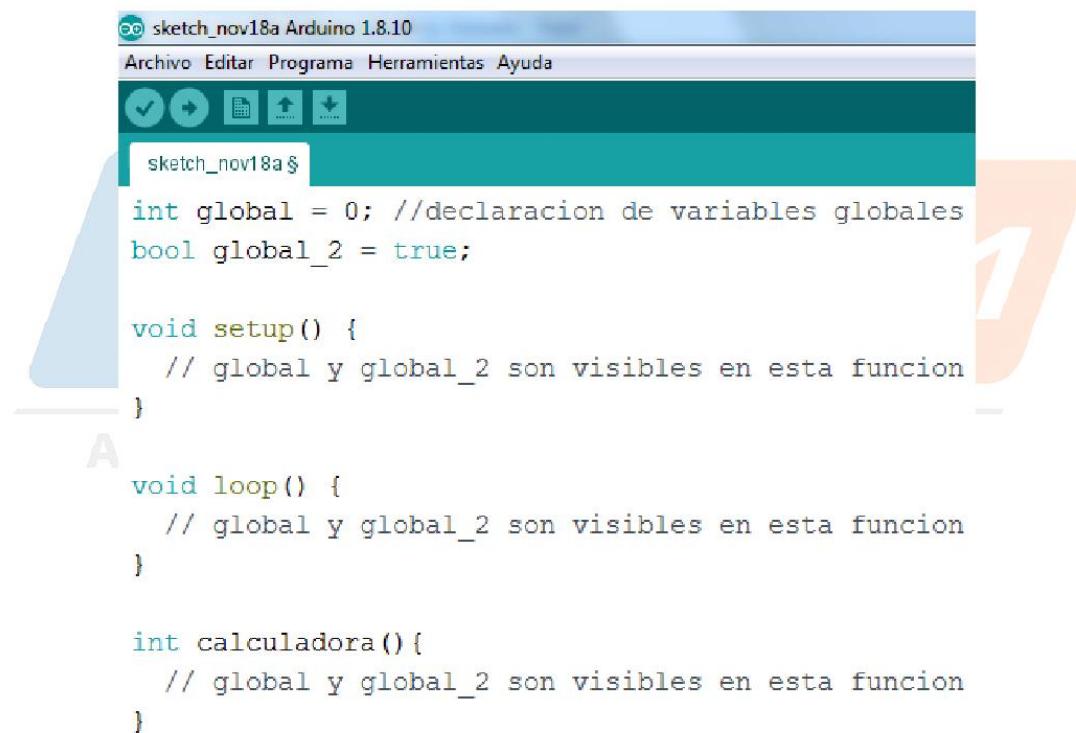


ACADEMIA DE SOFTWARE

Capítulo 2. FUNCIONES. PARTE 2

2.1.- Variables Globales y Locales

Una variable global es aquella que puede ser utilizada por cualquier función en el programa. Para crear una variable global esta debe ser declarada fuera de una función. Es una buena práctica declarar las variables globales al inicio del programa. Cualquier variable declarada fuera de una función es considerada como una variable global.



The screenshot shows the Arduino IDE interface with the title bar "sketch_nov18a Arduino 1.8.10". 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 area contains the following code:

```
int global = 0; //declaracion de variables globales
bool global_2 = true;

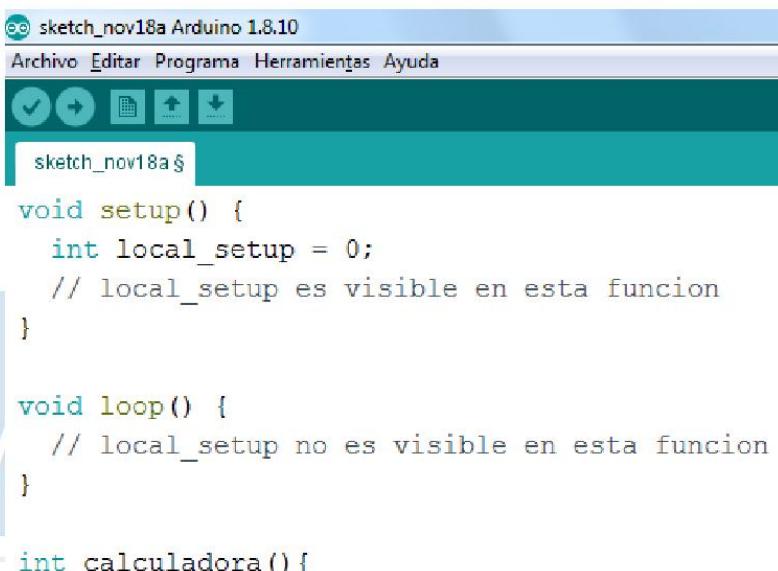
void setup() {
    // global y global_2 son visibles en esta funcion
}

void loop() {
    // global y global_2 son visibles en esta funcion
}

int calculadora(){
    // global y global_2 son visibles en esta funcion
}
```

Las variables locales son aquellas que solamente pueden ser utilizadas dentro de la función donde son declaradas. Esto también incluye los condicionales y ciclos, donde cualquier variable declarada dentro de sus llaves solamente pueden ser utilizadas adentro de ellas.

Cuando el programa empieza a ser más largo y más complejo, las variables locales son una forma útil de asegurarse que solamente una función tiene acceso a sus propias variables. Esto previene errores cuando una función inadvertidamente modifica una variable utilizada por otra función.



```
sketch_nov18a Arduino 1.8.10
Archivo Editar Programa Herramientas Ayuda
sketch_nov18a.ino

void setup() {
    int local_setup = 0;
    // local_setup es visible en esta función
}

void loop() {
    // local_setup no es visible en esta función
}

int calculadora(){
    // local_setup no es visible en esta función
}
```

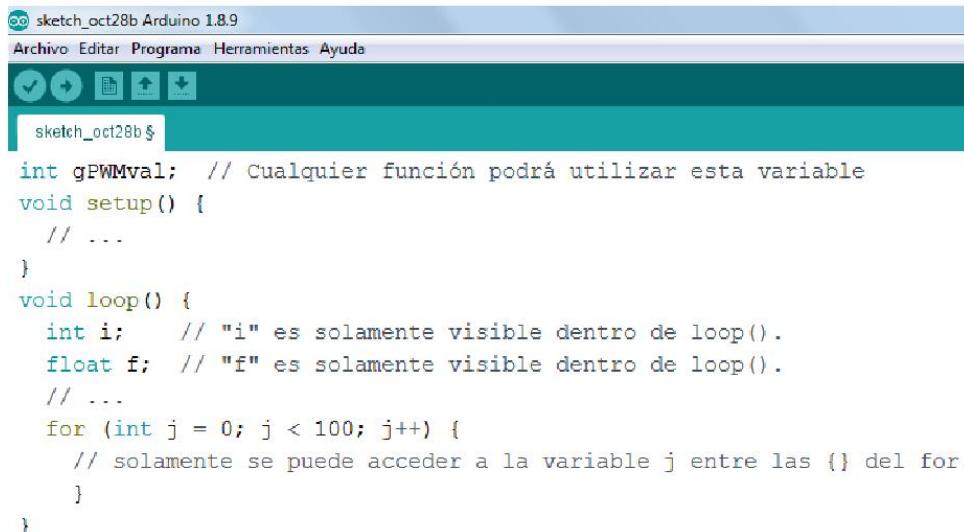
The image shows a screenshot of the Arduino IDE. The title bar says "sketch_nov18a Arduino 1.8.10". The menu bar includes "Archivo", "Editar", "Programa", "Herramientas", and "Ayuda". Below the menu is a toolbar with icons for file operations. The main area shows a code editor with the following content:

```
void setup() {
    int local_setup = 0;
    // local_setup es visible en esta función
}

void loop() {
    // local_setup no es visible en esta función
}

int calculadora(){
    // local_setup no es visible en esta función
}
```

Decorative elements include a blue 'AC' logo on the left and an orange '1' logo on the right.



```
sketch_oct28b Arduino 1.8.9
Archivo Editar Programa Herramientas Ayuda
sketch_oct28b
int gPWMval; // Cualquier función podrá utilizar esta variable
void setup() {
    // ...
}
void loop() {
    int i; // "i" es solamente visible dentro de loop().
    float f; // "f" es solamente visible dentro de loop().
    // ...
    for (int j = 0; j < 100; j++) {
        // solamente se puede acceder a la variable j entre las {} del for.
    }
}
```

2.2.- Funciones Con Parámetros

Las funciones tienen la capacidad de recibir valores al momento de ser llamadas. Estos valores que recibe la función se les denominan parámetros. Con estos valores se pueden realizar operaciones dentro de la función.

Para que una función pueda recibir parámetros, estos valores deben ser declarados dentro de los paréntesis de la función. La función de la imagen recibe dos parámetros, ambos de tipo entero.

Parámetros que recibe la función.

```
void mifuncion(int x, int y){
    int suma = x + y;
    Serial.println(suma);
}
```

Al momento de llamar la función, dentro de los paréntesis se deben pasar los valores de los parámetros. En el siguiente ejemplo, la función mifuncion() recibe por parámetros los valores 9 para cada parámetro y en el serial monitor muestra el resultado de la suma.

The screenshot shows the Arduino IDE interface. On the left, the code for 'sketch_oct24a' is displayed:

```
sketch_oct24a Arduino 1.8.9
Archivo Editar Programa Herramientas Ayuda
sketch_oct24a
void setup() {
    Serial.begin(9600);
}
void loop() {
    mifuncion(9,9);
    delay(1000);
}
void mifuncion(int x, int y) {
    int suma = x + y;
    Serial.println(suma);
}
```

On the right, the Serial Monitor window titled 'COM4' shows the output '18'.

En el siguiente ejemplo la función mifuncion() recibe por parámetros las lecturas de los pines A0 y A1 y muestra la suma de las lecturas en el Serial Monitor.

```

sketch_oct24a Arduino 1.8.9
Archivo Editar Programa Herramientas Ayuda
sketch_oct24a
void setup() {
    Serial.begin(9600);
}
void loop() {
    mifucion(analogRead(0),analogRead(1));
    delay(1000);
}
void mifucion(int x, int y){
    int suma = x + y;
    Serial.println(suma);
}

```

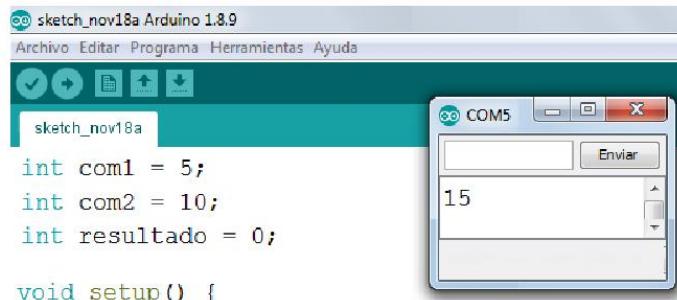
2.3.- Parámetros Por Referencia

Al pasar un parámetro a una función, se puede pasar de 2 formas:

- por valor: se envía un valor a la función. Si el valor cambia, no afecta el valor original.
- por referencia: se envía a la función una referencia de la variable. Si se modifica el valor adentro de la función, afecta al valor original.

En Arduino, por defecto, los parámetros se pasan por valor. Para que un parámetro sea pasado por referencia, debe colocarse el símbolo ampersand (&) antes del nombre del parámetro. Cuando un parámetro se pasa por referencia, lo que hace es pasar un apuntador a la dirección de memoria en la que se localiza la variable, por lo que, al modificar el valor de tal variable dentro de la función, modifica en realidad a la variable pasada por parámetros.

El siguiente ejemplo hace uso de los parámetros por referencia. Al ejecutar la función "mifucion()", el parámetro "resultado" se verá modificado, ya que el parámetro "suma" hace referencia a su posición de memoria.

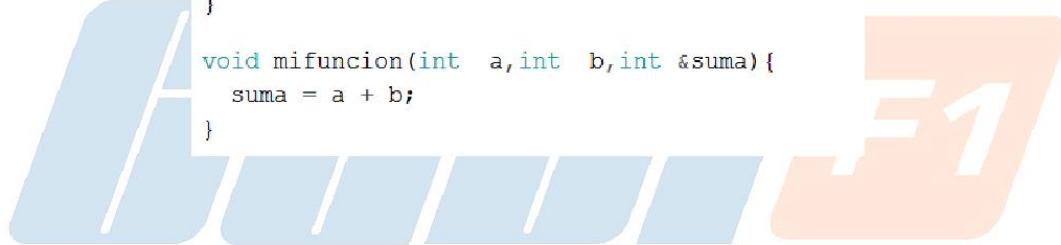


```
sketch_nov18a Arduino 1.8.9
Archivo Editar Programa Herramientas Ayuda
sketch_nov18a
int com1 = 5;
int com2 = 10;
int resultado = 0;

void setup() {
    Serial.begin(9600);
    mifucion(com1,com2,resultado);
}

void loop() {
    Serial.println(resultado);
    while(true);
}

void mifucion(int a,int b,int &summa){
    summa = a + b;
}
```



ACADEMIA DE SOFTWARE

Capítulo 3. RELÉS

3.1.- Introducción

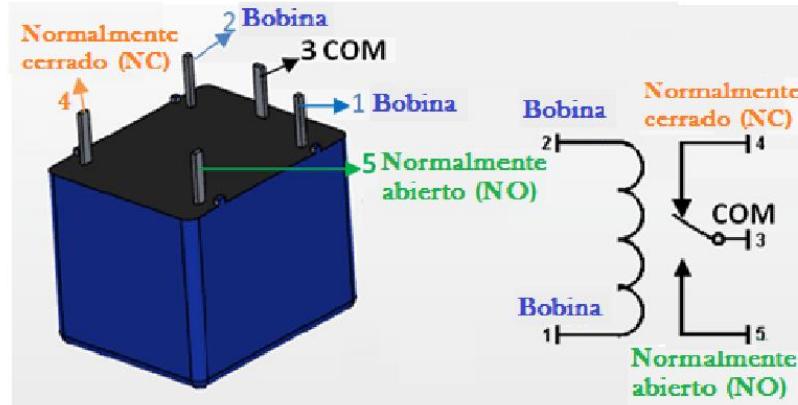
Un relevador, también conocido como relé o relay, es un interruptor magnético, que es accionado al momento que circule una cierta cantidad de corriente eléctrica entre los pines de la bobina. Esto crea un campo magnético que hace que internamente cierre el switch, en caso de que deje de aplicarse corriente, el relé dejará de accionarse. Los relés son utilizados para manejar altas potencias, como por ejemplo, encender un motor de 110V - 220V con una señal de 5V.

Algunas aplicaciones de los relés son:

- el encendido automático del compresor de un aire acondicionado cuando la temperatura del ambiente baja de cierto valor establecido.
- el encendido de una sirena en un sistema de seguridad cuando se detecta un movimiento.
- la apertura de una puerta cuando se lee un valor válido.
- el cambio de visión normal a visión nocturna en una cámara de vigilancia cuando se detecta un cambio en la cantidad de luz en el ambiente.

Un relé generalmente se compone de 5 pines. En los pines 1 y 2 se encuentra la bobina de acción, el pin 3 es el pin común del interruptor, el pin 4 es el normalmente cerrado y el pin 5 es el normalmente abierto. Al circular corriente en la bobina de acción esta genera un campo electromagnético el cual acciona el interruptor.

Mientras la bobina no esté accionada, entre los pines COM y NC habrá continuidad. Cuando la bobina sea accionada se activará el interruptor y los pines que tendrán continuidad serán el COM y NO.



Los Relés operan a ciertos voltajes. Estos valores pueden verse impresos en su carcasa. Indica a qué voltaje funciona correctamente la bobina de acción, en caso de superar el voltaje se podría quemar la bobina y en caso de no llegar al voltaje necesario ésta no se activará.

También existe una corriente máxima que soporta el interruptor que se puede visualizar en la carcasa.



3.2.- Conexión Del Relé al Arduino

Para activar un Relé usando un Arduino, es necesario utilizar componentes adicionales para su correcto funcionamiento, ya que los pines de Arduino por sí solos no tienen la capacidad de suministrar la cantidad de corriente necesaria para activar el interruptor.

Los componentes necesarios para activar el relé son los siguientes:

- * 1 Resistencia.
- * 1 Transistor (BJT).
- * 1 diodo.

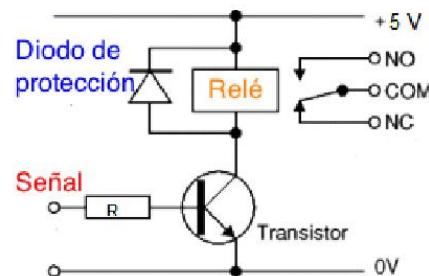
Esto se logra ya que el Transistor (BJT) funcionará como un switch que active el relé.

El diodo se conecta en paralelo a la bobina del relé para evitar dañar componentes sensibles a altos voltajes que se producen al interrumpir el flujo de corriente que circula por la bobina.

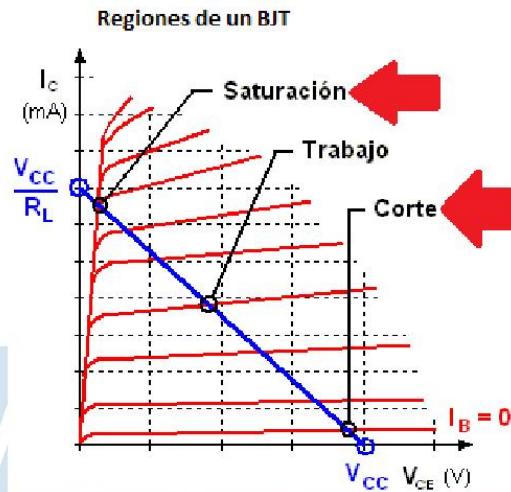
Para cuando el relé esté activo, por la bobina circula corriente, pero para cuando se desactive (se corte el flujo de corriente) la bobina cambia bruscamente la polaridad entre sus terminales para que la corriente siga circulando (por un espacio de tiempo limitado).

El diodo debe soportar el voltaje de alimentación, por Ejemplo:

Si se alimenta el relé de 5V,12V o 24V, el diodo debe soportar tales voltajes más un margen de seguridad.



Para activar el relé, se satura el transistor y para desactivarlo se lleva el transistor a la región de corte. La resistencia se utiliza para llevar el transistor a tales puntos de trabajo.



El siguiente esquemático es de un relé cuyo voltaje de operación es de 5v, VCC (voltaje de colector) se puede conectar directamente a la salida 5V de Arduino, aunque lo recomendable es utilizar una fuente de alimentación externa.

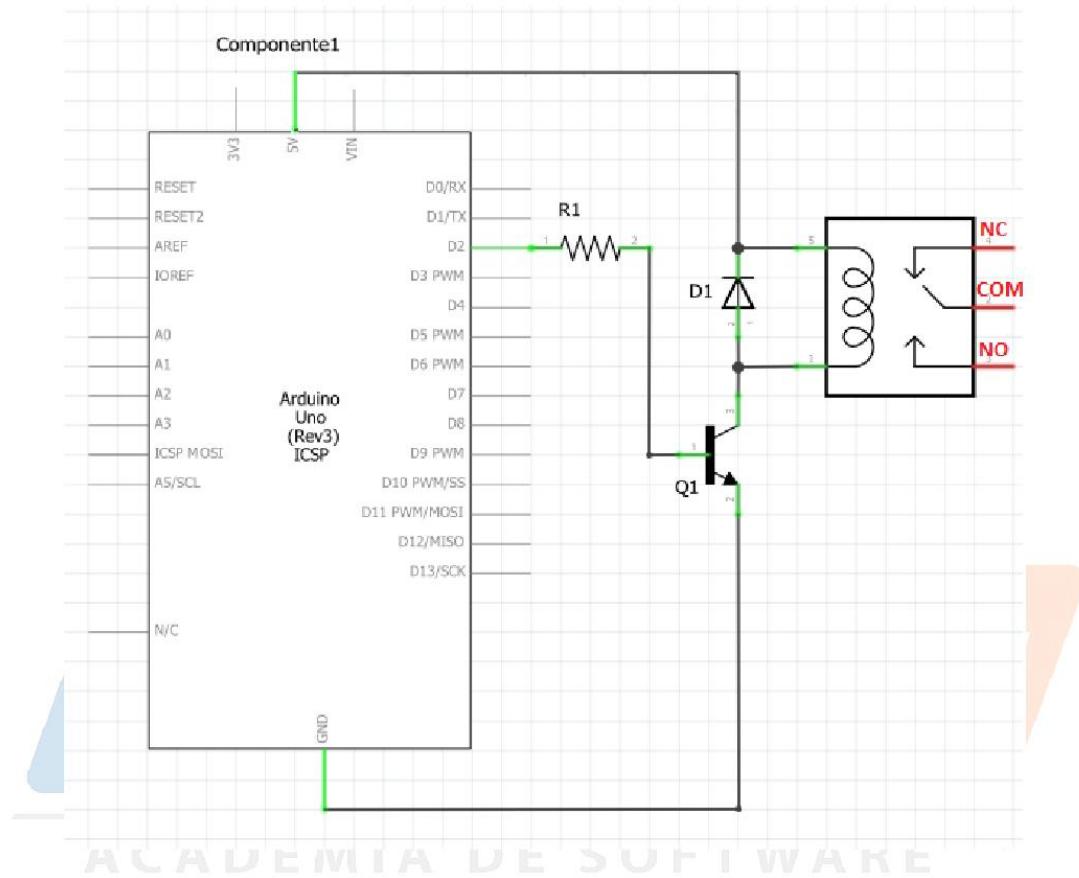


Diagrama de conexión para un relé de 5v alimentado directamente desde Arduino.

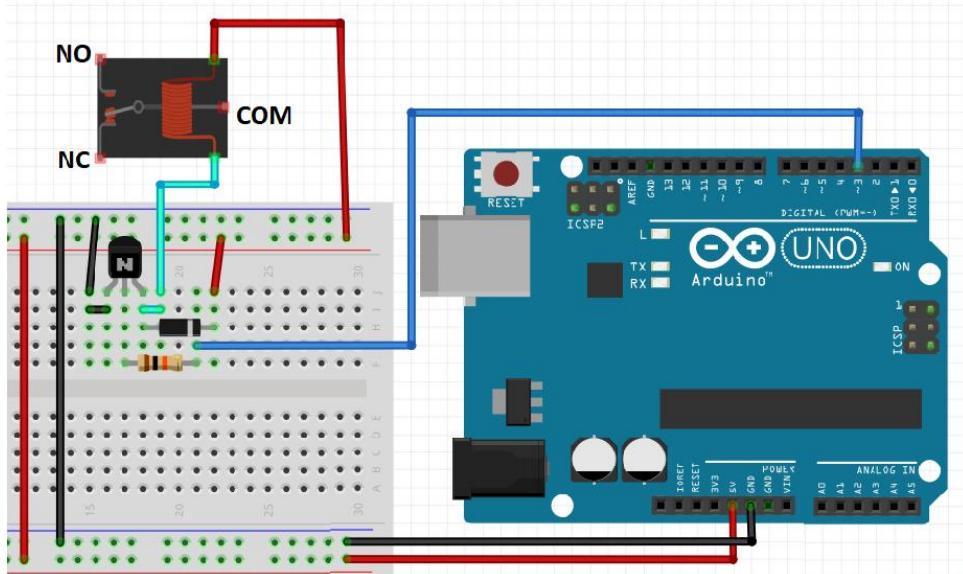
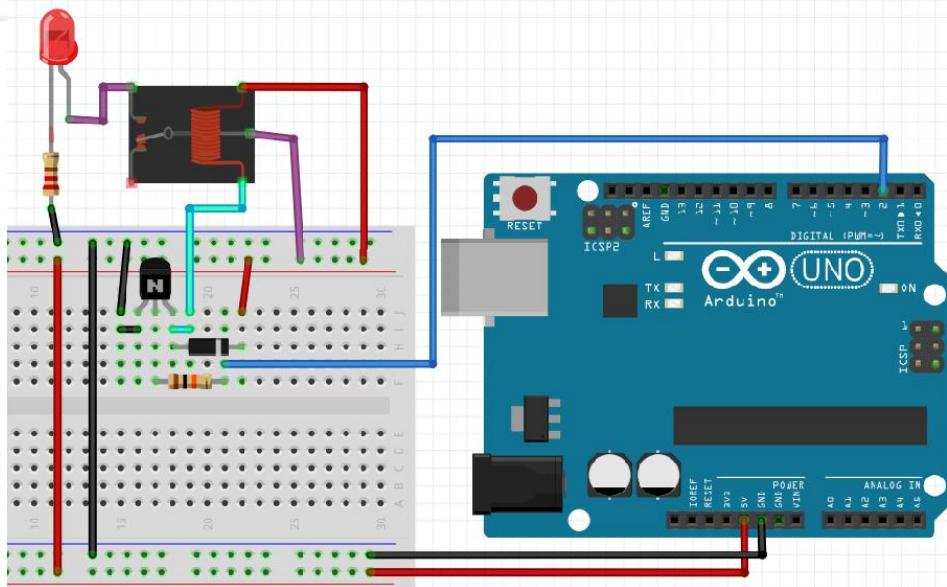


Diagrama de conexión para un relé de 5v alimentado directamente desde Arduino. Como el LED está conectado al NO, cada vez que se active el relé, encenderá el LED.



Esquemático para un relé que opera a 12V, este a diferencia del anterior esquemático, utiliza una fuente de alimentación externa de 12v, para accionar la bobina del relé y la señal del Arduino proviene del pin 13.

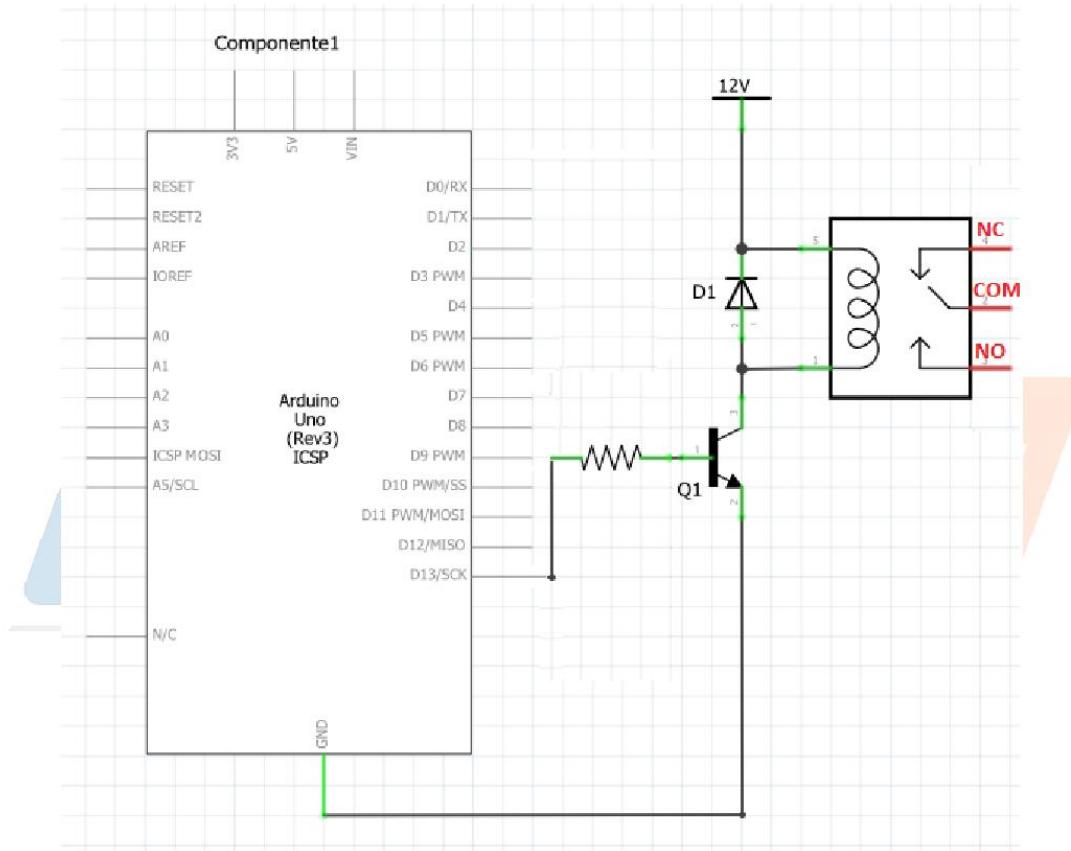
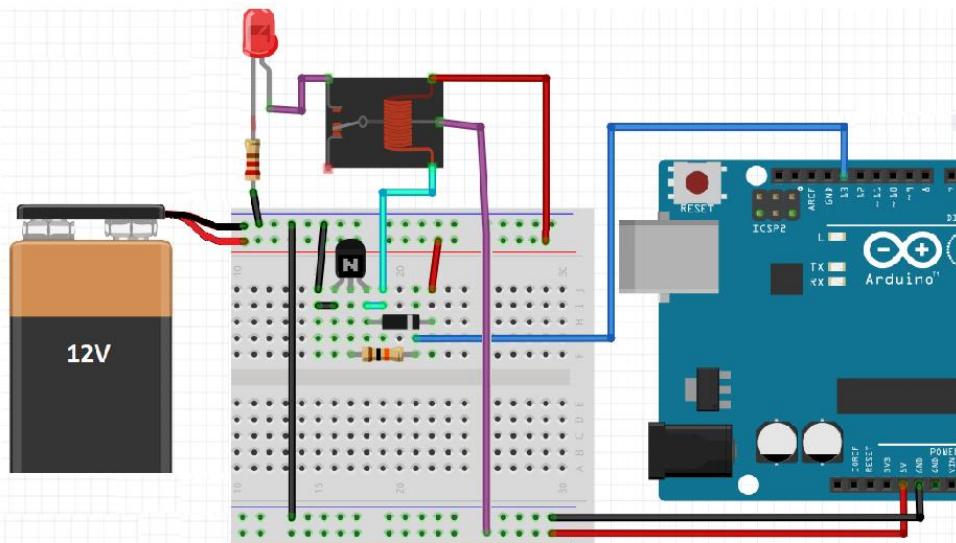


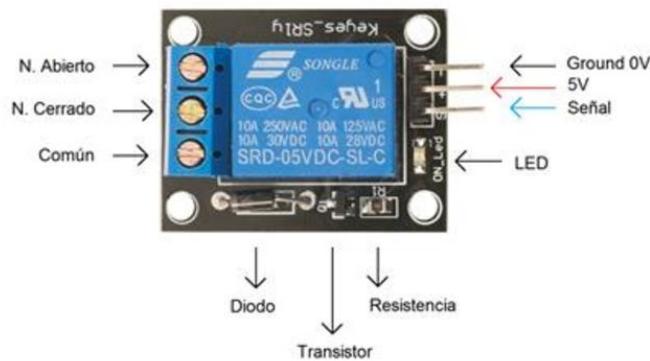
Diagrama de conexión para un relé de 12V:



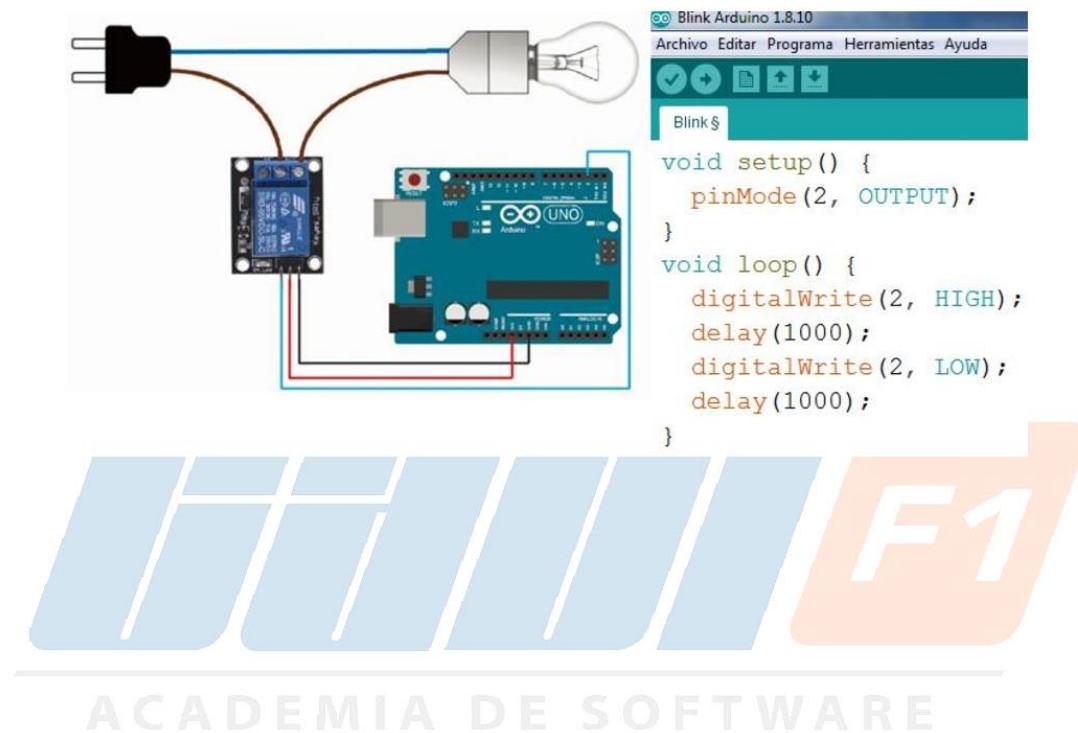
3.3.- Modulo de Relé Para Arduino

Lo recomendado para manejar un Relé con Arduino es utilizar un módulo especialmente diseñado para funcionar en conjunto, además, que facilita la conexión entre los 2 componentes (Arduino y Relé).

En el módulo, el pin “Señal” debe estar conectado al pin proveniente del Arduino.



En el siguiente ejemplo el Relé se activa cada 1 segundo y cada vez que este se active, encenderá el foco.



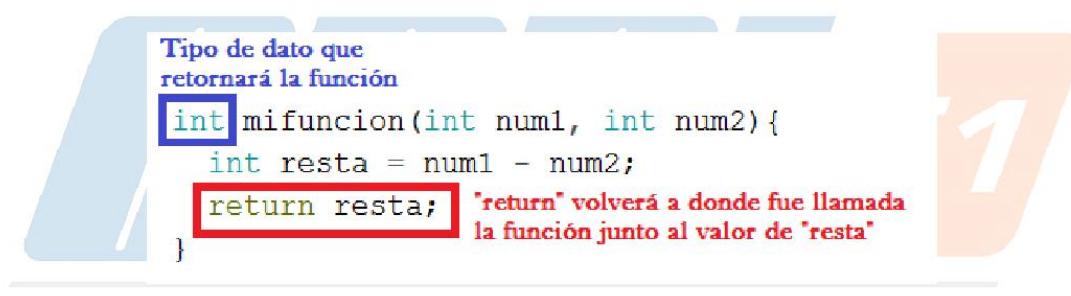
ACADEMIA DE SOFTWARE

Capítulo 4. FUNCIONES. PARTE 3

4.1.- Funciones Con Valor de Retorno

Las funciones con valor de retorno son aquellas que devuelven un valor o un dato durante su ejecución. Este valor puede ser de cualquier tipo de dato, a diferencia de las funciones "void" que no poseen valor de retorno, se debe especificar el tipo de dato que devolverá la función.

Dentro de la función, después de hacer las operaciones para retornar el valor hay que hacer uso de la cláusula "return". De esta manera la función devolverá el resultado que se requiere.



ACADEMIA DE SOFTWARE

Donde esté la cláusula "return" la función detendrá su ejecución y retornará a la instrucción donde fue llamada.

4.2.- Llamado de Funciones

El resultado del valor de retorno de una función se puede manejar directamente desde donde se llame, o se puede guardar en una variable para posteriormente utilizarla. En el siguiente ejemplo se muestra en el serial monitor el resultado de la resta proveniente del valor de retorno de la función.

```

sketch_nov08a Arduino 1.8.10
Archivo Editar Programa Herramientas Ayuda
sketch_nov08a: 
void setup() {
    Serial.begin(9600);
}
void loop() {
    Serial.println(mifucion(23, 6));
    delay(1000);
}
int mifucion(int x, int y){
    int resta = x - y;
    return resta;
}

```

En el siguiente ejemplo se guarda el valor de retorno en la variable resultado y se muestra en el serial monitor. Los valores que recibe la función por parámetros son las lecturas de los pines A0 y A1.

```

sketch_oct24a Arduino 1.8.10
Archivo Editar Programa Herramientas Ayuda
sketch_oct24a: 
int resultado;
void setup(){
    Serial.begin(9600);
}
void loop() {
    resultado = mifucion(analogRead(0),analogRead(1));
    Serial.println(resultado);
    delay(1000);
}

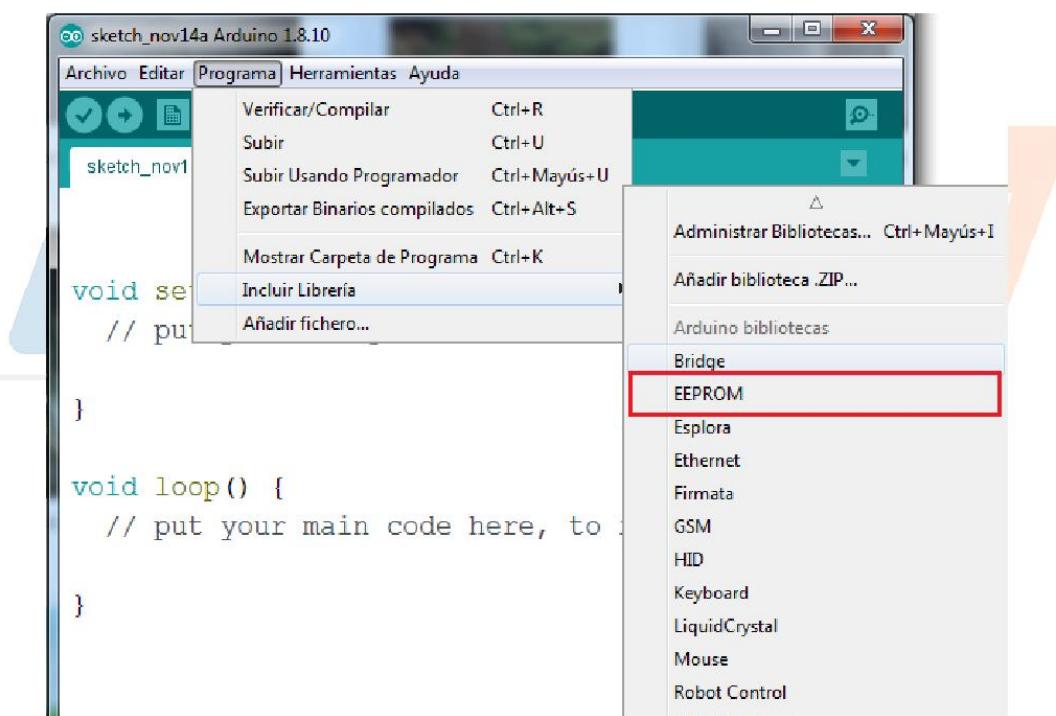
int mifucion(int num1, int num2){
    int resta = num1 - num2;
    return resta;
}

```

Capítulo 5. LIBRERÍAS

5.1.- Librerías

El entorno de Arduino puede ser extendido a través del uso de librerías, como la mayoría de otros lenguajes de programación. Las librerías proveen funcionalidades extra para el uso de sketches, tal como, manejar hardware o manipular información. Para usar una librería en un sketch, se selecciona la opción Programa > Incluir librería (Sketch > Import library).



Al incluir una librería, se agrega al inicio del código una o varias líneas que comienzan con la directiva "#include", seguido del nombre de la librería. En el siguiente ejemplo, se agregó la librería "EEPROM":

The screenshot shows the Arduino IDE interface. The title bar reads "sketch_nov14a Arduino 1.8.10". The menu bar includes "Archivo", "Editar", "Programa", "Herramientas", and "Ayuda". Below the menu is a toolbar with icons for file operations. The main code editor window contains the following code:

```
#include <EEPROM.h>

void setup() {
    // put your setup code here, to run once:

}

void loop() {
    // put your main code here, to run repeatedly:

}
```

The line "#include <EEPROM.h>" is highlighted with a red box. The status bar at the bottom right shows "Arduino Nano, ATmega328P (Old Bootloader) en COM4".

Las librerías estándar en Arduino son aquellas que son desarrolladas por el mismo equipo de Arduino, estas son:

- EEPROM: Incorpora las instrucciones necesarias para gestionar las memorias no volátiles.
- Ethernet: Se emplea para usar la conexión de internet del Arduino Ethernet Shield, Arduino Ethernet Shield 2 y Arduino Leonardo ETH.
- Firmata: Es un protocolo genérico para la comunicación con microcontroladores desde software instalado en un ordenador.

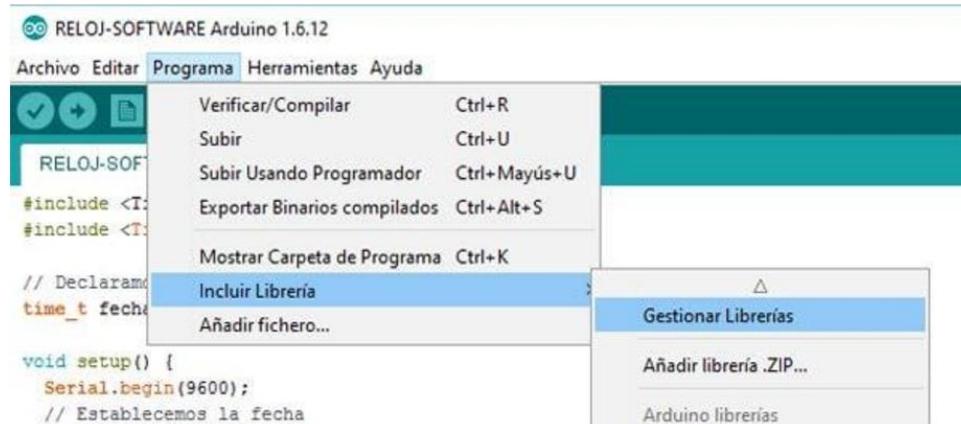
- GSM: Permite a una placa Arduino hacer la mayoría de las operaciones que se pueden hacer con un teléfono GSM.
- LiquidCrystal: Permite a un Arduino controlar LiquidCrystal displays (LCDS).
- SD: Permite leer y guardar contenido en una SD.
- Servo: Implementa la clase del mismo nombre, destinada a facilitar la comunicación de Arduino con servomotores.
- Stepper: Permite controlar motores de paso unipolares o bipolares.
- TFT: Permite al Arduino conectarse con el TFT LCD para dibujar líneas, formas e imágenes, entre otros.
- WiFi: Con el Arduino Wifi Shield, permite a la placa Arduino conectarse a internet.
- Keyboard: Las funciones de esta librería permiten enviar KeyStrokes a una computadora a través del puerto micro USB.
- Mouse: Las funciones de la librería mouse permiten controlar el movimiento de un cursor en una computadora conectada.

En muchos casos se necesitará una librería externa a Arduino (creada por usuarios), las cuales son clasificadas como "no estándar". Estas son usadas para manipular componentes o información, como por ejemplo: para los sensores de temperatura, teclados matriciales e incluso utilizar un botón para diferentes propósitos.

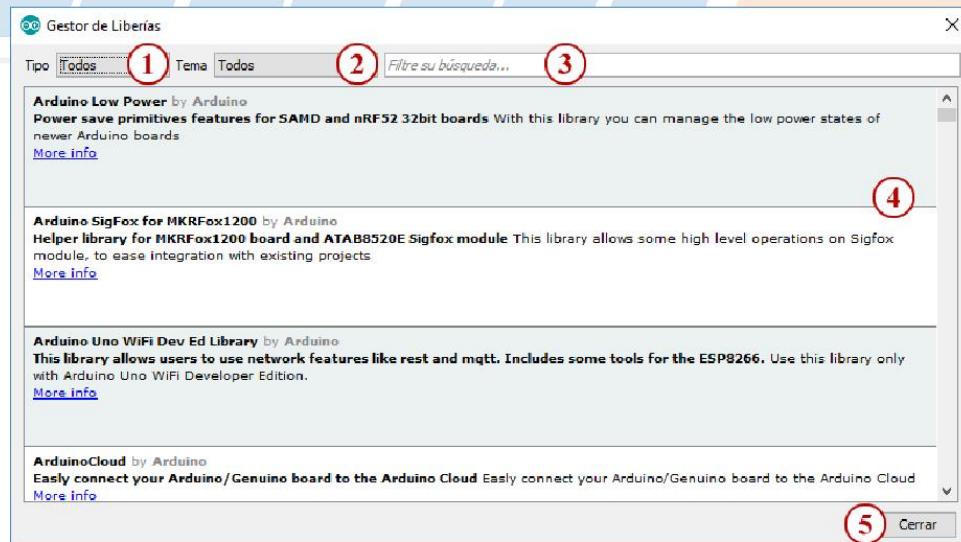
Existen varias formas de instalar una librería en Arduino, a través del gestor de librerías o mediante un archivo .zip

5.2.- Instalar Librería Con el Gestor de Librerías

La forma más sencilla de instalar una librería de Arduino es a través del Gestor de Librerías. Para abrir este gestor sólo hay que ir a Programa>Incluir Librería>Gestionar Librerías.



Esta ventana permite buscar e instalar las librerías que han sido aprobadas por Arduino, pero no son estándar, es decir, librerías de terceros. Se pueden encontrar de diferentes fabricantes de hardware como Adafruit o Sparkfun y otras librerías para diferentes propósitos.



La ventana del gestor de librerías está compuesta por varios elementos:

1. Filtro por tipo: las librerías se pueden filtrar por su tipo, por ejemplo:

- Todos: muestra todas las librerías.
- Actualizable: muestra las librerías que tienen una actualización disponible.
- Instalado: muestra la lista de librerías que están instaladas en el IDE de desarrollo.
- Arduino: muestra las librerías estándar que están instaladas en tu IDE de desarrollo.
- Contribución: muestra el listado de librerías no estándar.
- Retirado: son las librerías que se han retirado o que no se siguen desarrollando. En este tipo puede haber librerías estándar y no estándar.

2. Filtro por tema: también se pueden filtrar las librerías por tema.

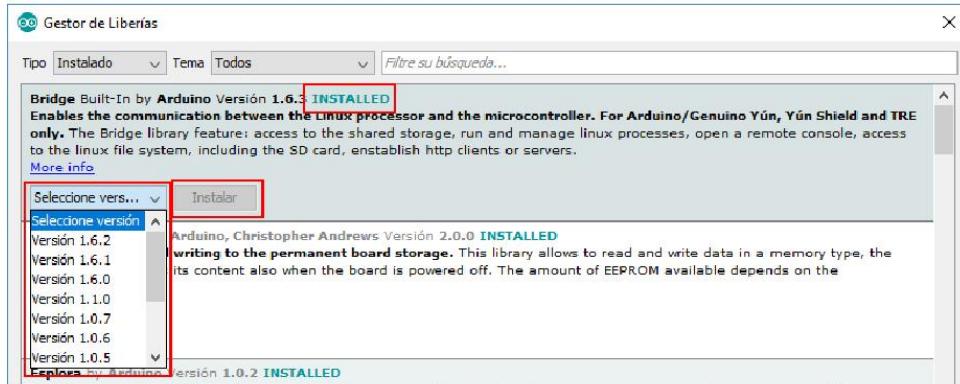
3. Filtro por nombre: se pueden buscar por las librerías por palabra.

4. Listado de librerías: En esta área aparecerá el listado de las librerías según los filtros aplicados.

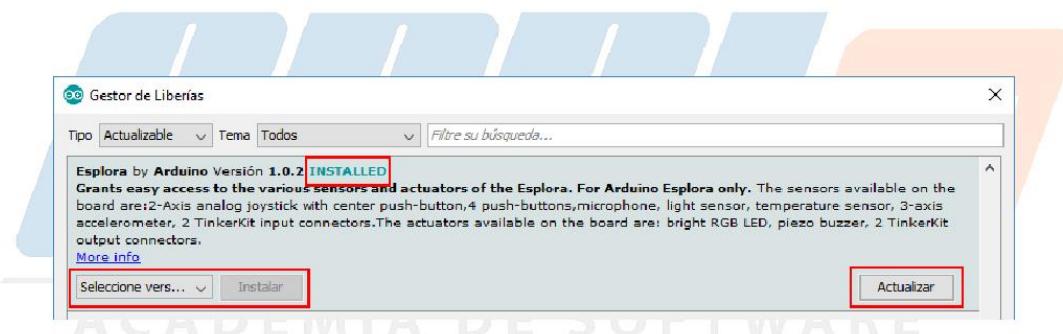
Una librería puede estar en 3 estados: sin instalar, instalada o pendiente de actualización. Una librería sin instalar solamente permite una acción, instalarla. Al hacer click en una librería en este estado, aparecerá un selector para seleccionar la versión y un botón de instalar.



Una librería instalada solo permite cambiar de versión.



Por último, en una librería que está pendiente de actualización aparecerá el botón de actualizar.



5.3.- Instalar Una Librería a Través de un .zip

En muchas ocasiones, en el repositorio oficial de Arduino no están todas las librerías ya que Arduino sigue una política muy estricta para que una librería forme parte de este repositorio.

La mayoría de librerías no estándar pueden ser encontradas en repositorios de github. Por ejemplo, existe una librería que permite conocer la ubicación de una placa de Arduino a través de una conexión wifi (esta librería solo funciona con placas que tienen conexión wifi). La URL para descargar esta librería es <https://github.com/gmag11/WifiLocation>.

En la página de GitHub, para descargar el .zip hay que dar click en el botón Clone or download y luego en la opción "Download ZIP", como lo muestra la siguiente imagen:

gmag11 / ESPWiFiLocation

Code Issues Pull requests Projects Wiki Insights

Google GeoLocation API wrapper for Arduino MKR1000, ESP8266 and ESP32

17 commits 2 branches 2 releases 1 contributor

Branch: master New pull request

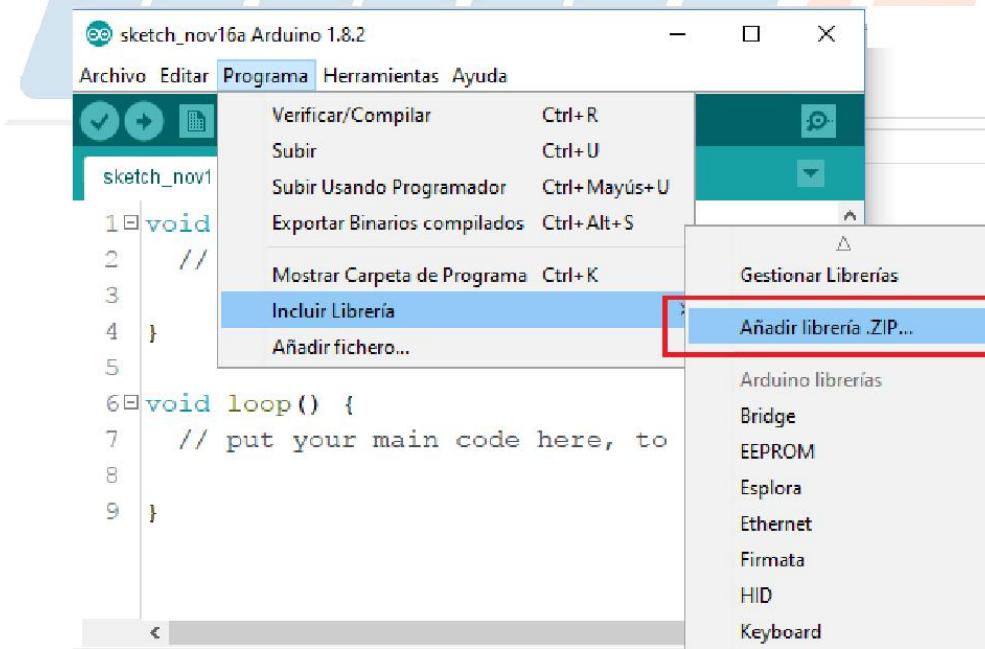
Clone with HTTPS Use SSH
https://github.com/gmag11/ESPWiFiLocation...

Open in Desktop Download ZIP

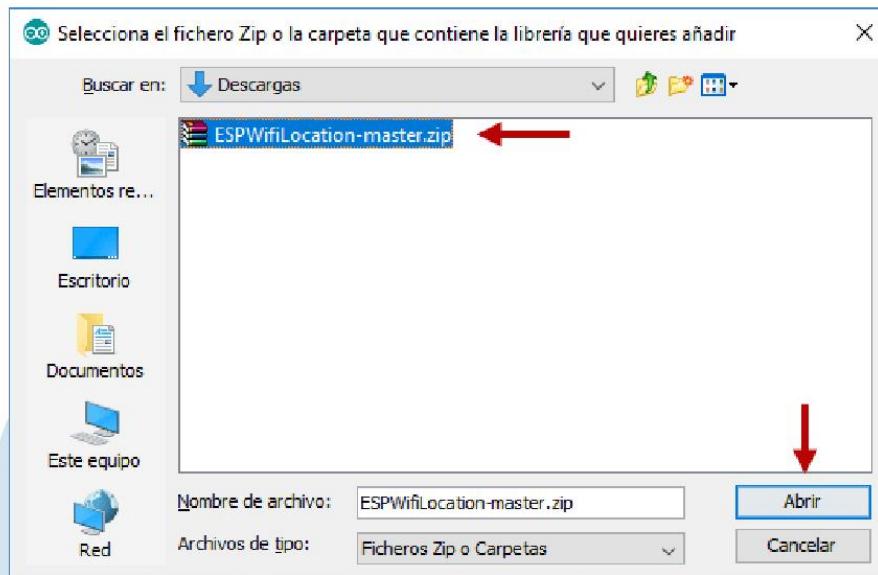
6 months ago 10 months ago 6 months ago

examples/googleLocation Remove debug line
src Add support to ESP32 architecture
LICENSE.md Initial commit
README.md Fix readme.md headers
keywords.txt Initial commit
library.properties Add support to ESP32 architecture

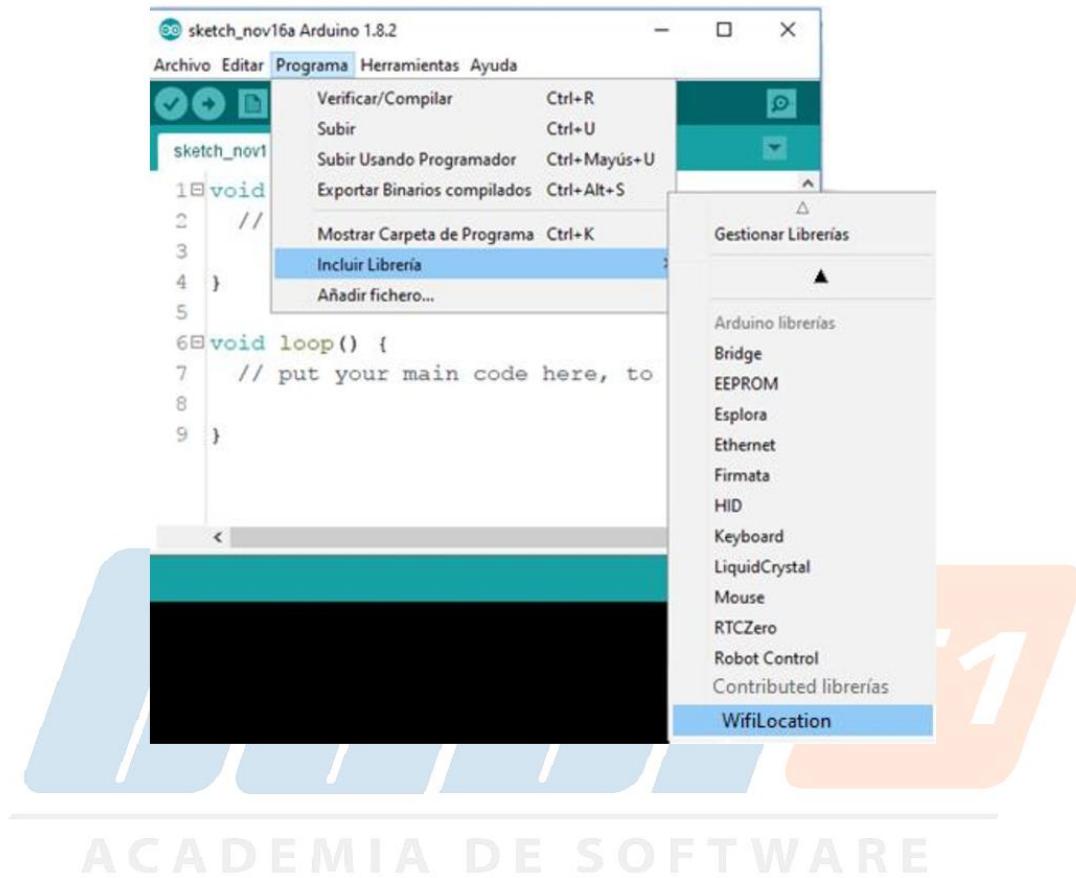
Luego de descargar el archivo .zip con la librería, se debe agregar en el IDE de Arduino. Para lograrlo, se hace click en la opción Programa>Incluir Librería>Añadir librería .ZIP.



Esto abrirá una ventana para buscar el archivo .ZIP previamente descargado para seleccionarlo.



Si todo está correcto, la nueva librería aparecerá en el IDE de Arduino:



Capítulo 6. LCD. PARTE 1

6.1.- Liquid Crystal Display

Un LCD (Liquid Crystal Display) es un display alfanuméricico de matriz de puntos que sirve para mostrar mensajes a través de caracteres como letras, números o símbolos. La placa del display viene equipado con un microcontrolador que se encarga de generar los caracteres, polarizar la pantalla y desplazar el cursor.

También viene equipado con una memoria ROM donde están almacenados los caracteres a través de una matriz de puntos, y una memoria RAM. Estos displays disponen de unos pins para conectar un microcontrolador (en este caso un Arduino) para poder dar instrucciones al display.



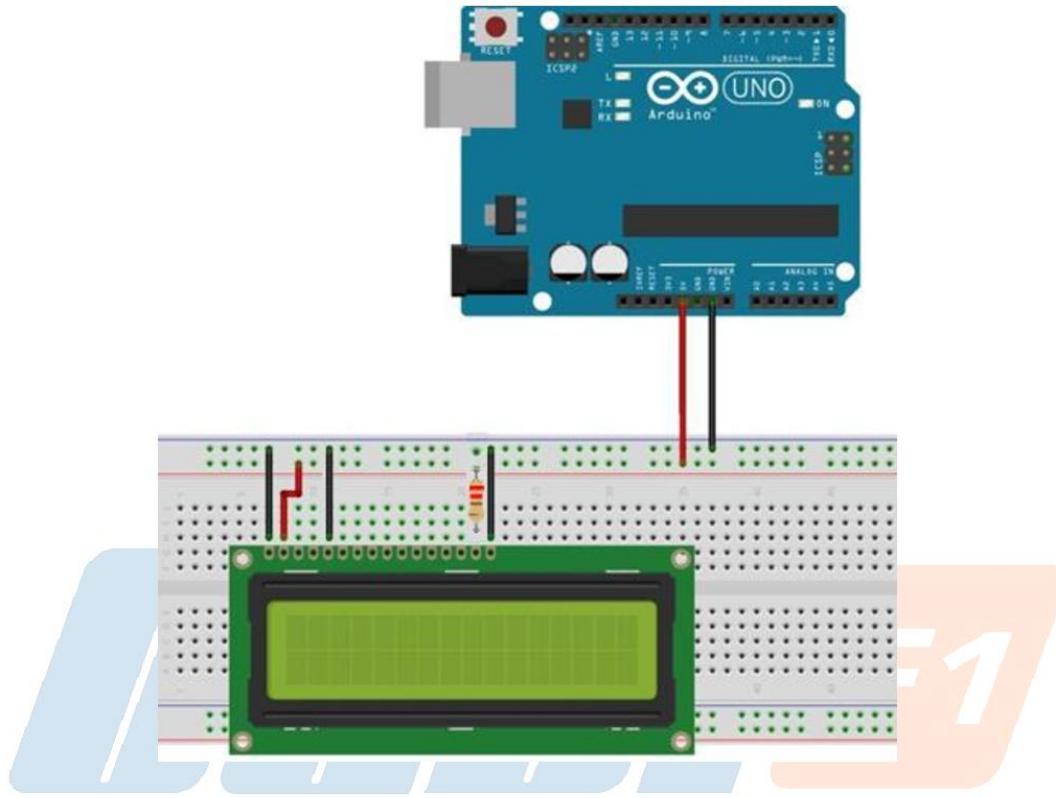
Los pines del LCD son:

- 1: VSS: voltaje negativo para la pantalla.
- 2: VDD: voltaje positivo para la pantalla.

- 3: VE: voltaje de contraste.
- 4: Register Select: corresponde a los registros, estos son registros de comando para cuando RS=0(0V) y registros de datos para cuando RS=1(5V). El registro de comando almacena las instrucciones dadas a la LCD, mientras que el registro de datos almacena los caracteres a mostrar.
- 6: Enable: envía información a los pines de datos cuando se da un pulso HIGH a LOW.
- 7 al 14: data pin. Son los pines de datos por donde la pantalla recibe lo que debe mostrar.
- 15: LED P: positivo de voltaje para el led de la pantalla.
- 16: LED N: negativo de voltaje para el led de la pantalla.



Generalmente para Arduino se utilizan LCD de 16x2 (16 caracteres y 2 filas). Para hacer funcionar la pantalla primero se conecta la alimentación. Como la pantalla es de 5v, se conecta a la alimentación del Arduino. Por seguridad, a la alimentación LED se conecta con una resistencia protectora de 220 Ohm :



6.2.- Contraste Del Lcd

Las pantallas LCD tienen un pin para especificar el nivel de contraste que se desea tenga la pantalla. Por defecto, el contraste de la pantalla está en cero, por lo tanto, no se podrá visualizar nada de lo que se envíe a esta. Por esta razón, se utiliza el pin 3 (VE Contrast) para graduar el nivel de contraste que se desea obtener. Se puede hacer de 2 formas:

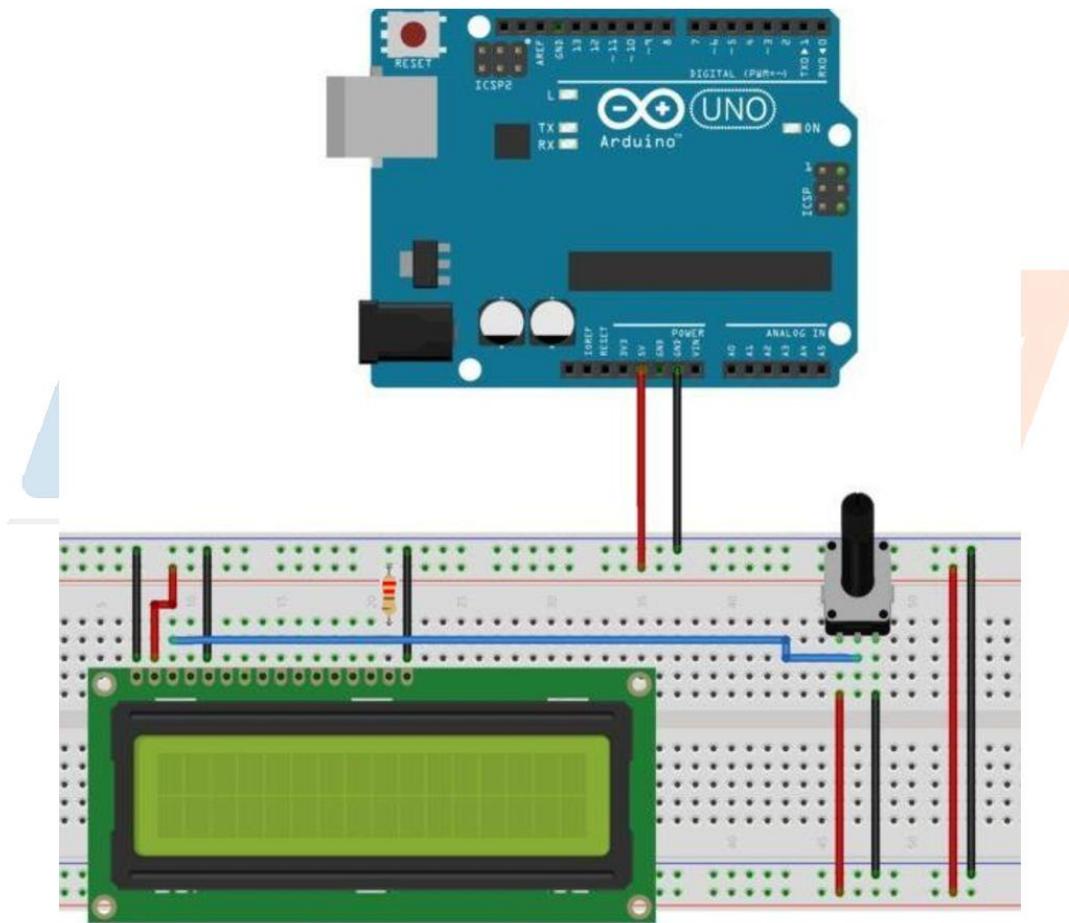
- con hardware: utilizando una resistencia (fija o variable).
 - con software: enviando un valor desde el Arduino.

Si se hace por hardware, se puede hacer de 2 formas:

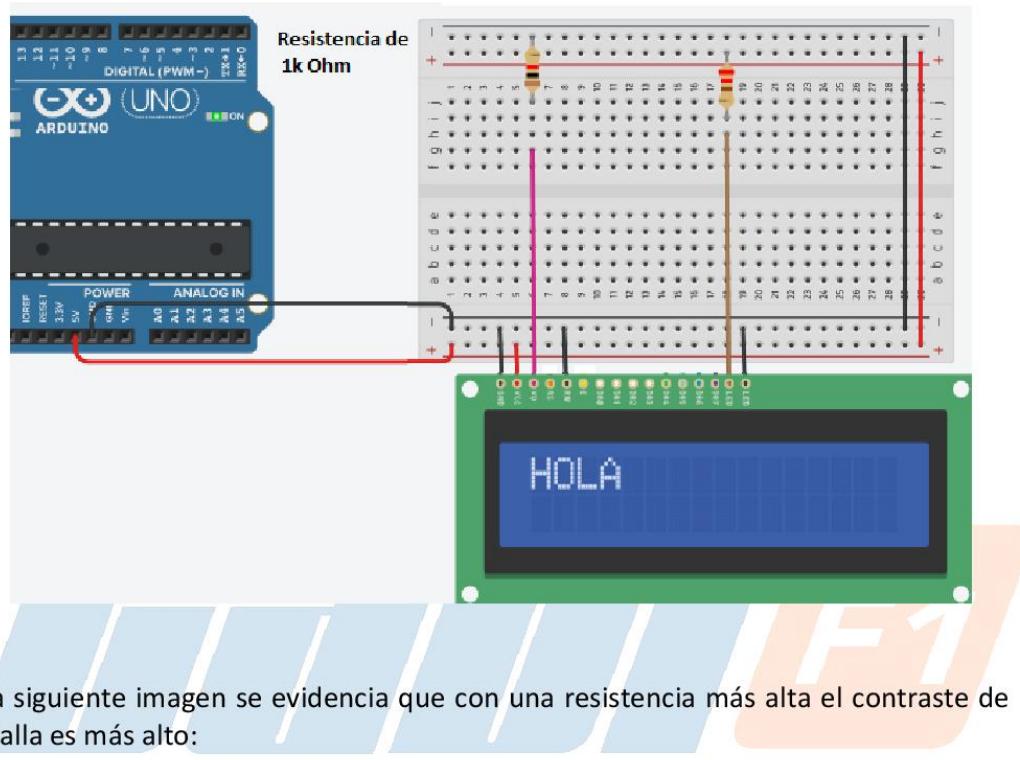
- con una resistencia variable (potenciómetro): el usuario podrá cambiar el valor de contraste a su gusto.

- con una resistencia fija: el nivel de contraste quedará fijo dependiendo del valor de la resistencia.

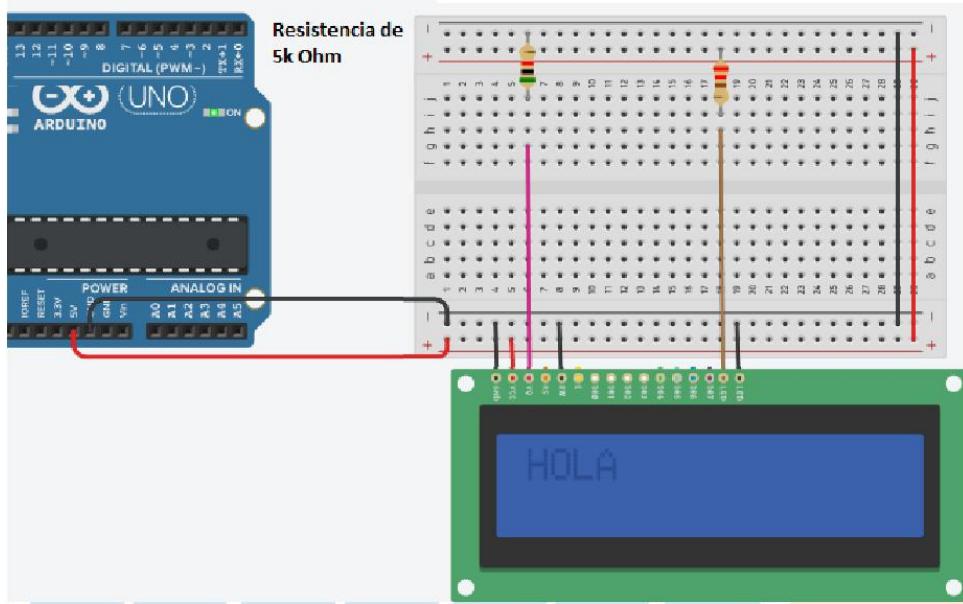
Si se desea especificar el nivel de contraste de forma que pueda ser graduable por el usuario, se utiliza un potenciómetro. Los extremos del potenciómetro van al positivo y negativo de la alimentación, mientras que el centro va al pin Contraste del módulo LCD.



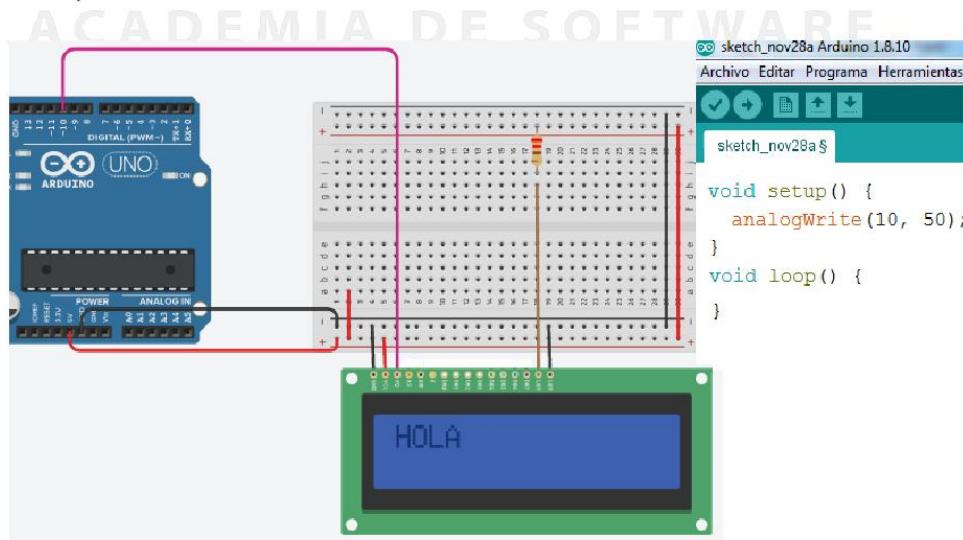
Si se desea especificar el nivel de contraste con una resistencia fija, se conecta la resistencia entre el pin 3 y GND:



En la siguiente imagen se evidencia que con una resistencia más alta el contraste de la pantalla es más alto:

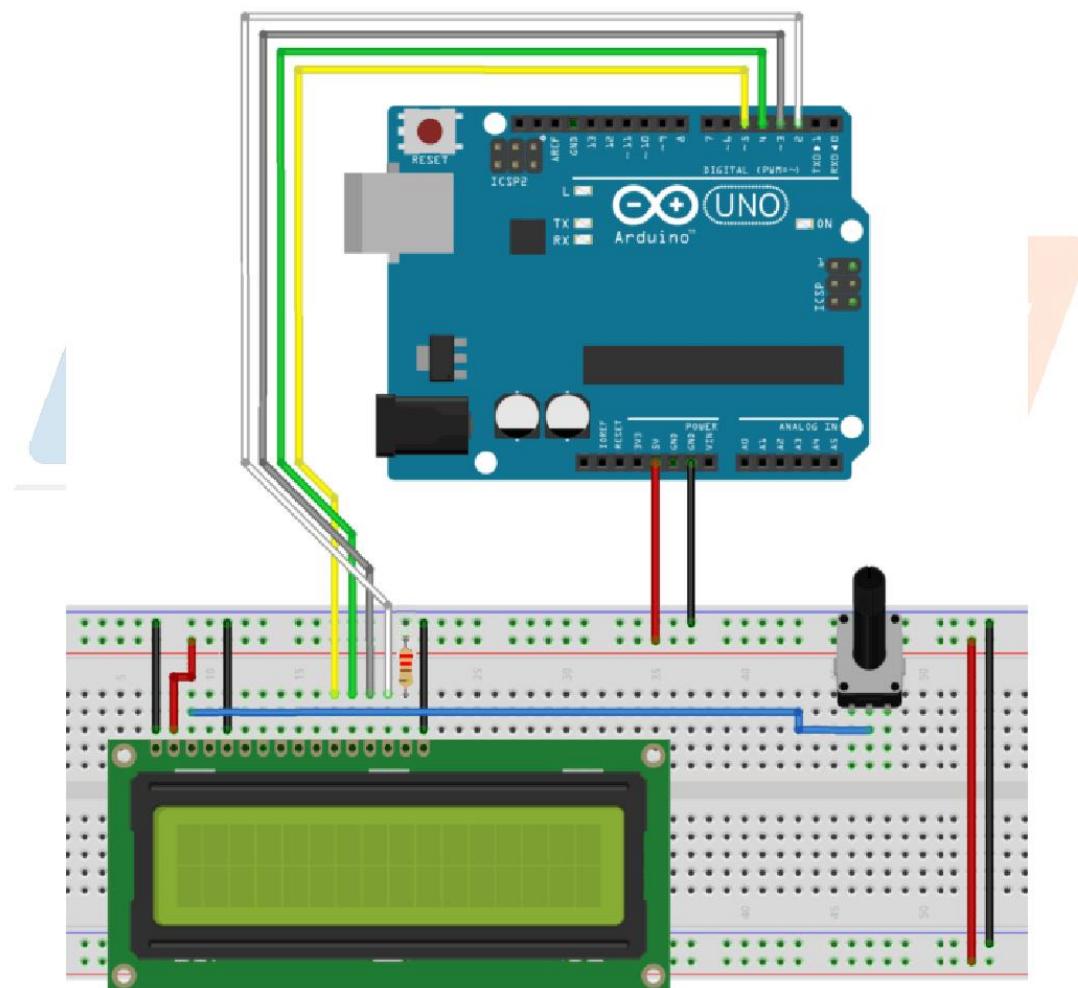


Para establecer el nivel de contraste por software, se envía un valor analógico entre 0 y 255 al pin conectado al pin de contraste de la pantalla. Un valor 0 establece el contraste al valor mínimo y 255 indica que el contraste está al máximo, por lo tanto, no se visualizará nada en la pantalla.

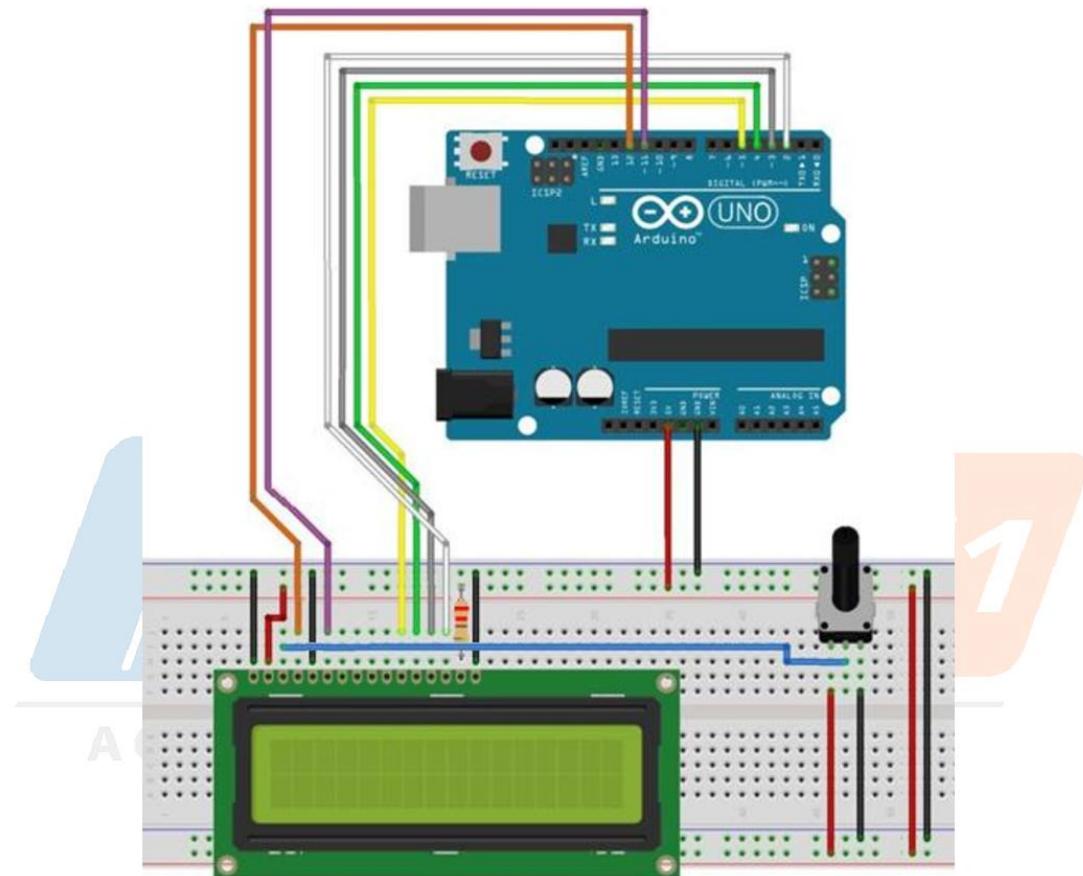


6.3.- Conexión de Datos Del Lcd

Para enviar información a la pantalla, se utilizan los pines D0, D1, D2, D3, D4, D5, D6 y D7 a la LCD. Se pueden utilizar solamente 4 (D7 a D4 para 4 bits) o todos para 8 bits. La diferencia entre utilizar los 4 y 8 bits, es que para 4bits habrá más latencia en el envío de datos, con la ventaja de que se ahorran 4 pines del Arduino que pueden ser útiles para otras tareas. Si se van a usar sólo 4 bits, sólo se conectan los pines del D4 al D7:



Por último, se realiza la conexión de los pines de control RS y E.



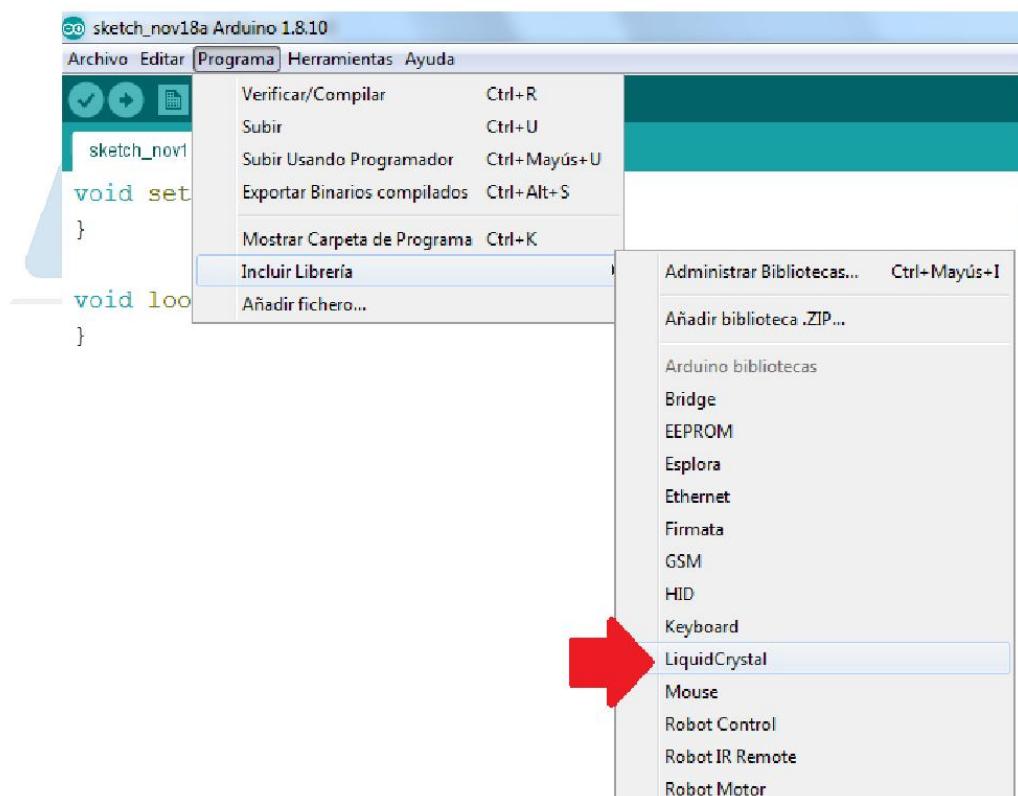
Capítulo 7. LCD. PARTE 2

7.1.- Librería Lcd

Una pantalla LCD necesita una librería para su correcto funcionamiento. Esta librería viene por defecto en el IDE de Arduino.

Se puede agregar al código desde la opción "Incluir Librería" o se puede escribir el siguiente código al inicio del programa:

```
#include <LiquidCrystal.h>
```

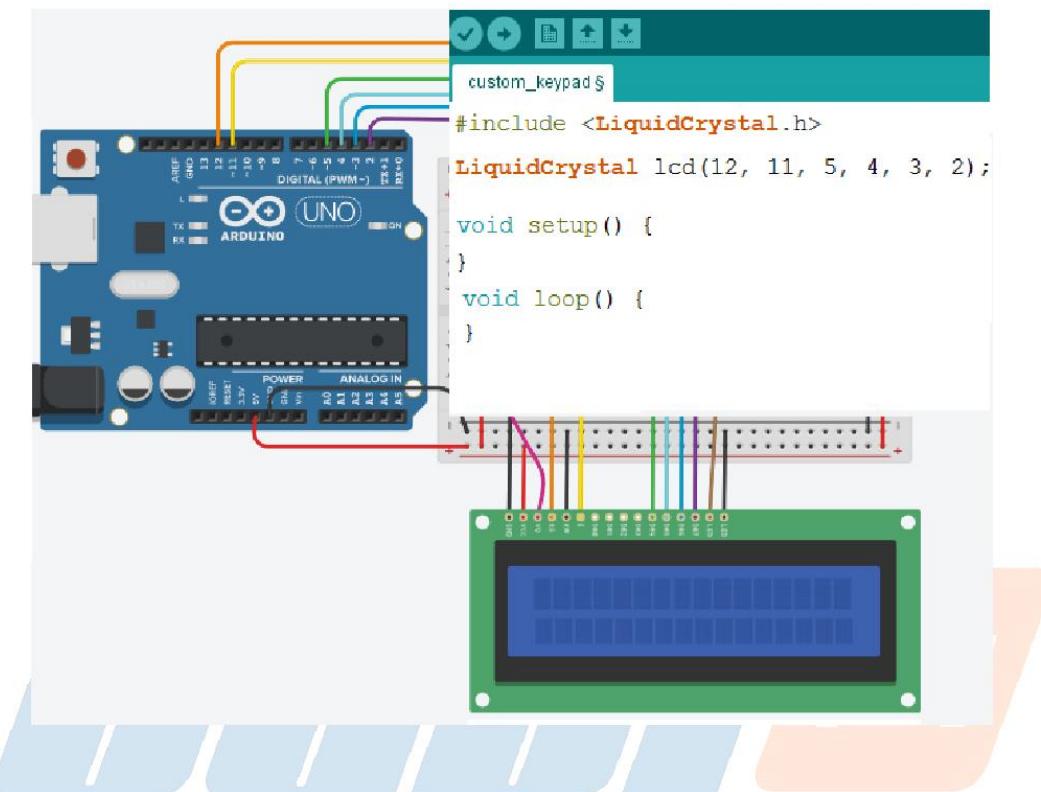


La librería posee los siguientes métodos:

- begin: inicializa la pantalla.
- home: ubica el cursor en el primer carácter de la primera fila sin borrar el display.
- setCursor: ubica el cursor en una ubicación específica.
- write: imprime un texto en la pantalla.
- print: igual que write pero con la posibilidad de enviar directamente números enteros al display, en distintas bases de numeración
- clear: limpia la pantalla.
- autoscroll: si el texto es más largo que la pantalla hace un scroll automáticamente.
- noAutoscroll: si el texto es más largo que la pantalla, no hace scroll automático.
- scrollDisplayRight: desplaza el contenido un carácter a la derecha.
- scrollDisplayLeft: desplaza el contenido un carácter a la izquierda.
- display: se utiliza para volver a encender la LCD después de haber utilizado noDisplay.
- noDisplay: apaga la LCD.

7.2.- Inicializar la Pantalla

Para inicializar la pantalla, se declara una variable global de tipo LiquidCrystal, que es un tipo de dato definido en la librería del mismo nombre. Al declarar la variable, se especifican los pines que se conectaron a la pantalla. Por ejemplo:

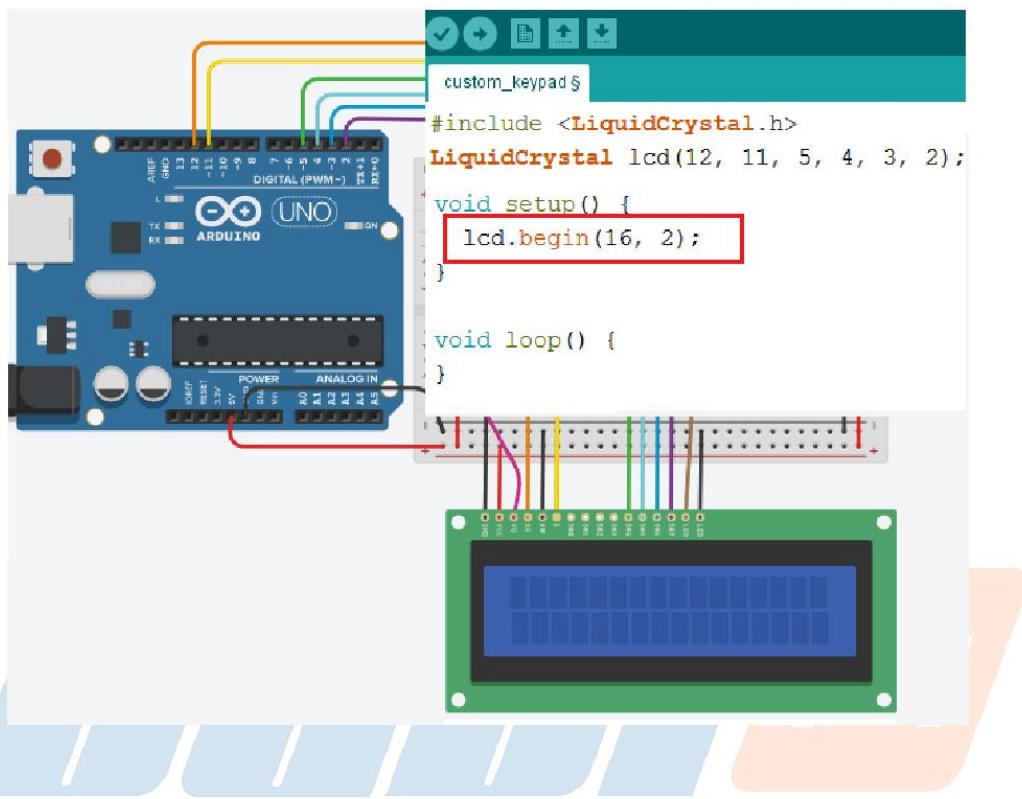


Los dos primeros números (el 11 y el 12) se refieren a los pines conectados a los puntos RS y E del display. Los cuatro últimos números se refieren a los pines de D4 a D7 del bus de datos del display. En general, esta configuración es la más simple y típica. No obstante, admite otras listas de argumentos, por ejemplo:

`LiquidCrystal lcd(Rs, E, D0, D1, D2, D3, D4, D5, D6, D7);`

Se utilizaría si se conectan los 8 pines de datos de la pantalla.

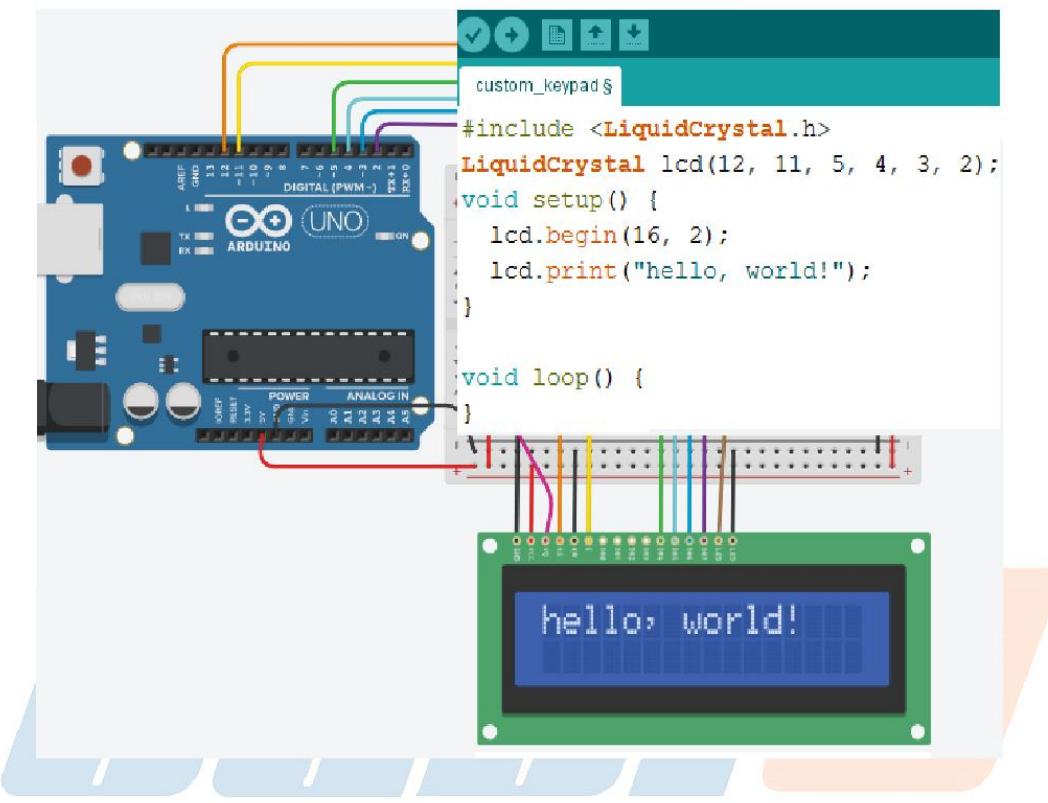
Parte de la configuración es especificar el tamaño de la pantalla. Esto se logra con el método "begin", que se debe llamar en la función "setup". El método recibe 2 parámetros: número de columnas y número de filas. Por ejemplo:



7.3.- Enviando Información

ACADEMIA DE SOFTWARE

El siguiente ejemplo muestra un mensaje simple en la LCD:



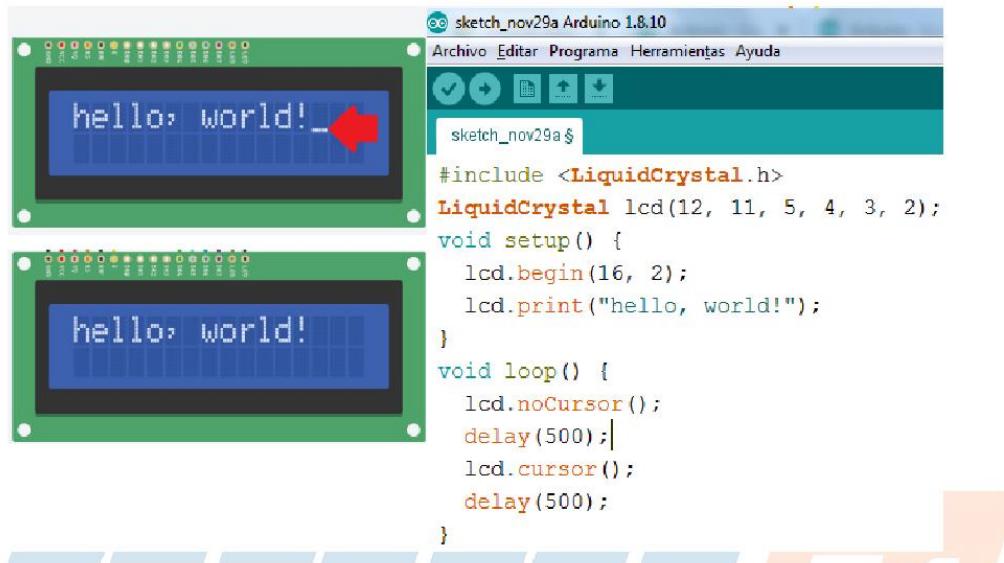
Con el mismo montaje, el siguiente código hará que el mensaje se mueva 5 posiciones a la derecha y 5 posiciones a la izquierda.

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
void setup() {
    lcd.begin(16, 2);
    lcd.print("hello, world!");
}
void loop() {
    for(int i = 0; i < 5; i++){
        lcd.scrollDisplayRight();
    }
    for(int i = 0; i < 5; i++){
        lcd.scrollDisplayLeft();
    }
}
```

El siguiente ejemplo hace uso de los métodos display() y noDisplay() cada 5 segundos.

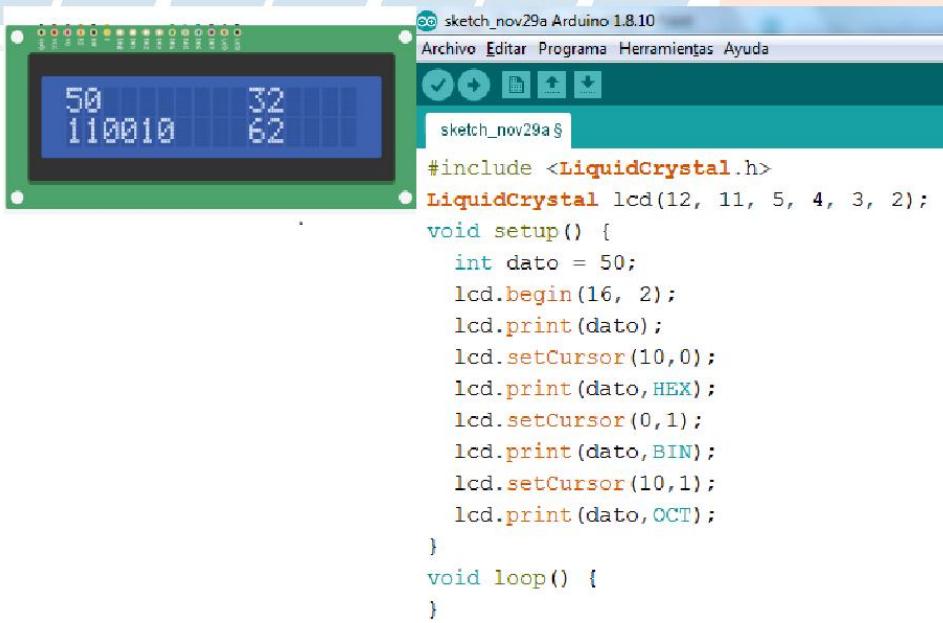
```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
void setup() {
    lcd.begin(16, 2);
    lcd.print("hello, world!");
}
void loop() {
    lcd.noDisplay();
    delay(5000);
    lcd.display();
    delay(5000);
}
```

El siguiente ejemplo muestra el cursor de forma intermitente.



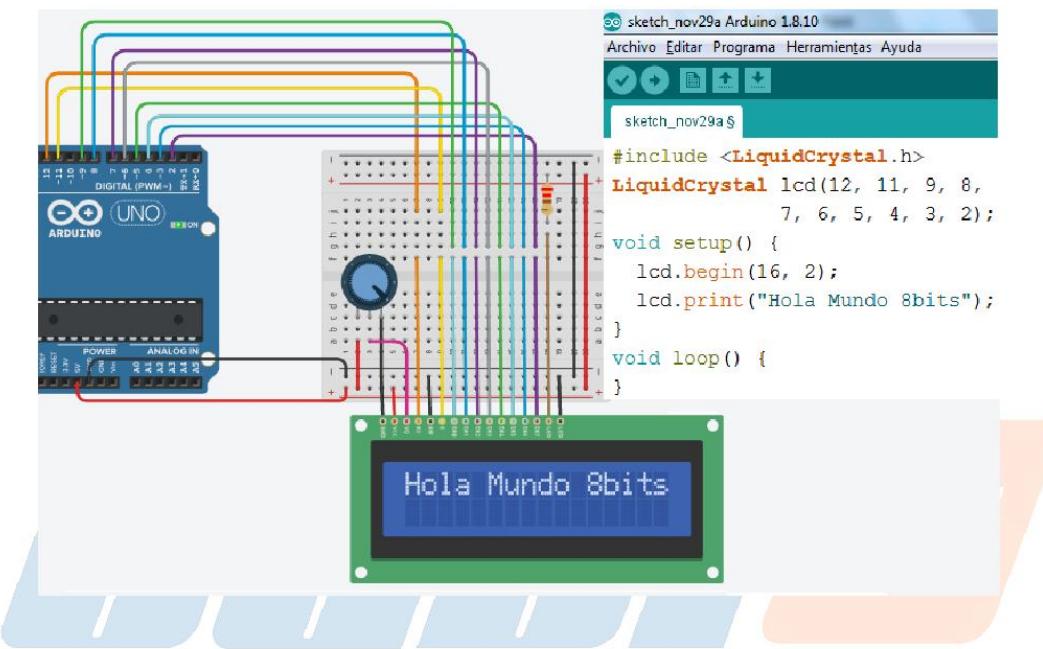
```
sketch_nov29a Arduino 1.8.10
Archivo Editar Programa Herramientas Ayuda
sketch_nov29a $ 
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
void setup() {
  lcd.begin(16, 2);
  lcd.print("hello, world!");
}
void loop() {
  lcd.noCursor();
  delay(500);
  lcd.cursor();
  delay(500);
}
```

En el siguiente ejemplo se imprime una variable. El método print recibe como segundo argumento la base (sistema numéricico) a usar.



```
sketch_nov29a Arduino 1.8.10
Archivo Editar Programa Herramientas Ayuda
sketch_nov29a $ 
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
void setup() {
  int dato = 50;
  lcd.begin(16, 2);
  lcd.print(dato);
  lcd.setCursor(10,0);
  lcd.print(dato,HEX);
  lcd.setCursor(0,1);
  lcd.print(dato,BIN);
  lcd.setCursor(10,1);
  lcd.print(dato,OCT);
}
void loop() { }
```

El siguiente ejemplo muestra el montaje para utilizar los 8 bits de la LCD, al igual que el código de como se inicializa la LCD para 8 bits:



En pantallas de 16x01 la librería LiquidCrystal genera resultados diferentes, ya que cuando se le envía un mensaje a la pantalla esta solamente mostrará los primeros 8 caracteres, por ejemplo: si se envía lcd.print("1234567890"), en la pantalla solamente se visualizarán los números del 1 al 8. Para resolver este problema, se imprimen los 8 primeros caracteres desde la posición 0 y los siguientes caracteres se imprimen desde la posición 40 utilizando el método "setCursor".



```
sketch_deco2a Arduino 1.8.10
Archivo Editar Programa Herramientas Ayuda
sketch_deco2a.h
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
void setup() {
    lcd.begin(16,2); //o lcd.begin(8,2);
    //Muestra los primeros 8 caracteres
    lcd.print("0123456");
    lcd.setCursor(40,0);
    //Muestra los 8 caracteres restantes
    lcd.print("WXYZ[¥]^");
}
```

Esto es debido a que los caracteres después de la posición 8 están ubicados a partir del espacio de memoria 40 de la LCD y hay que imprimir en tales espacios de memoria para visualizar el mensaje completo.



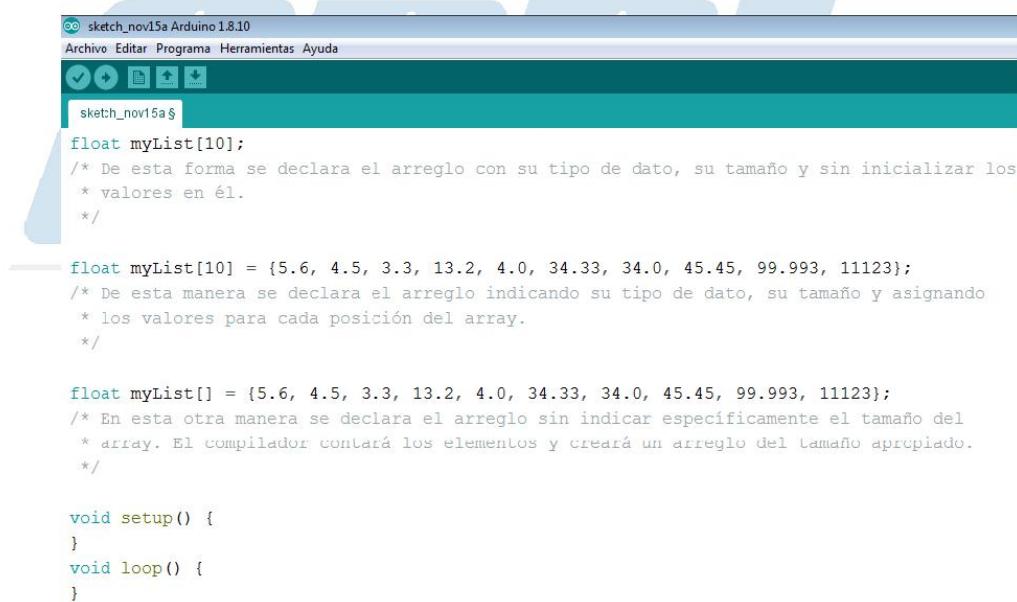
Capítulo 8. ARREGLOS. PARTE 1

8.1.- Arreglos

Un arreglo o un array es una estructura de datos que permite tener una colección de valores bajo un mismo nombre. Un arreglo se puede declarar de 3 formas:

- inicializando su tamaño específicamente.
- inicializando sus valores.
- inicializando tanto el tamaño como los valores.

El siguiente es un ejemplo de como declarar e inicializar arreglos:



```

sketch_nov15a Arduino 1.8.10
Archivo Editar Programa Herramientas Ayuda
sketch_nov15a §
float myList[10];
/* De esta forma se declara el arreglo con su tipo de dato, su tamaño y sin inicializar los
 * valores en él.
 */

float myList[10] = {5.6, 4.5, 3.3, 13.2, 4.0, 34.33, 34.0, 45.45, 99.993, 11123};
/* De esta manera se declara el arreglo indicando su tipo de dato, su tamaño y asignando
 * los valores para cada posición del array.
 */

float myList[] = {5.6, 4.5, 3.3, 13.2, 4.0, 34.33, 34.0, 45.45, 99.993, 11123};
/* En esta otra manera se declara el arreglo sin indicar específicamente el tamaño del
 * array. El compilador contará los elementos y creará un arreglo del tamaño apropiado.
 */

void setup() {
}
void loop() {
}

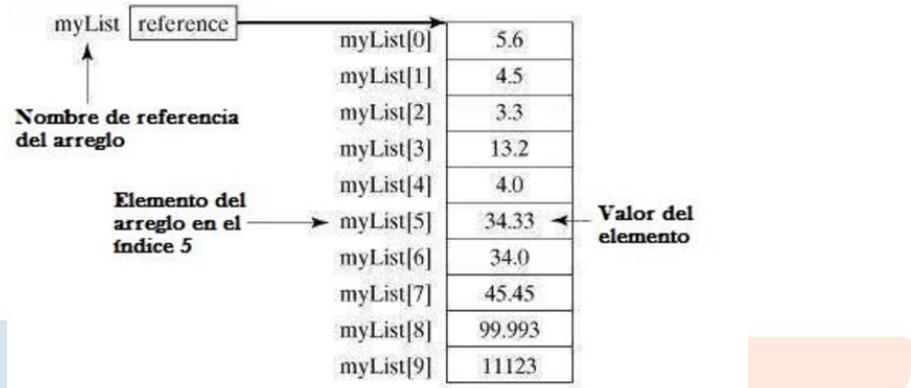
```

8.2.- Acceso a Elementos

Para acceder los valores almacenados en un arreglo, ya sea para asignarles o para averiguar el valor almacenado, hay que hacer uso de un índice, el cual indica la posición

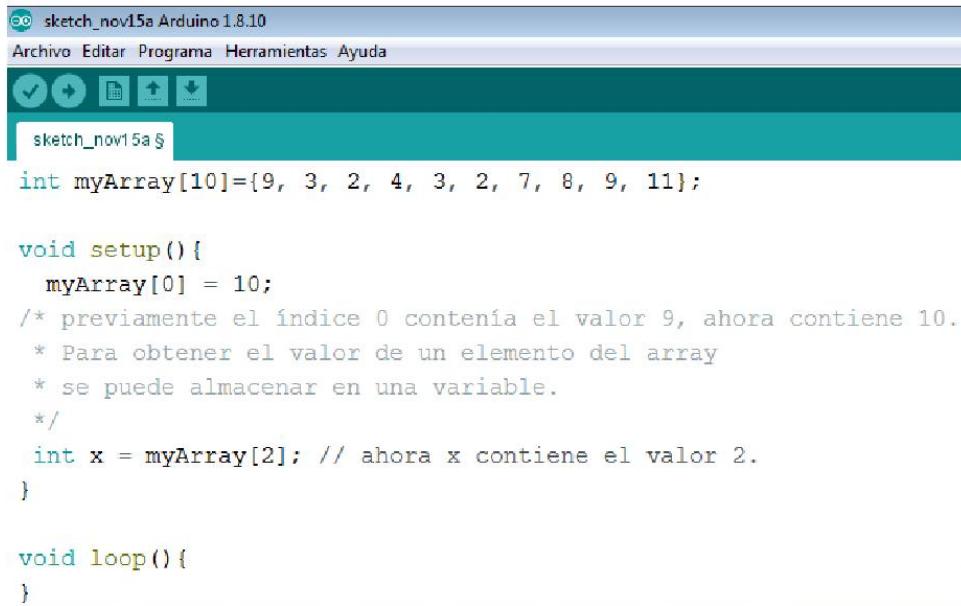
dentro del arreglo donde esta se encuentra el elemento. El primer elemento de un arreglo está en el índice o posición 0. De esta manera, da la imagen de ejemplo:

myList[0] es igual a 5.6 y myList[4] es igual a 4.0



El siguiente ejemplo muestra como asignar valor a una posición del arreglo y cómo asignar a una variable el valor almacenado en otra posición del mismo arreglo:

ACADEMIA DE SOFTWARE



```

sketch_nov15a Arduino 1.8.10
Archivo Editar Programa Herramientas Ayuda
sketch_nov15a §

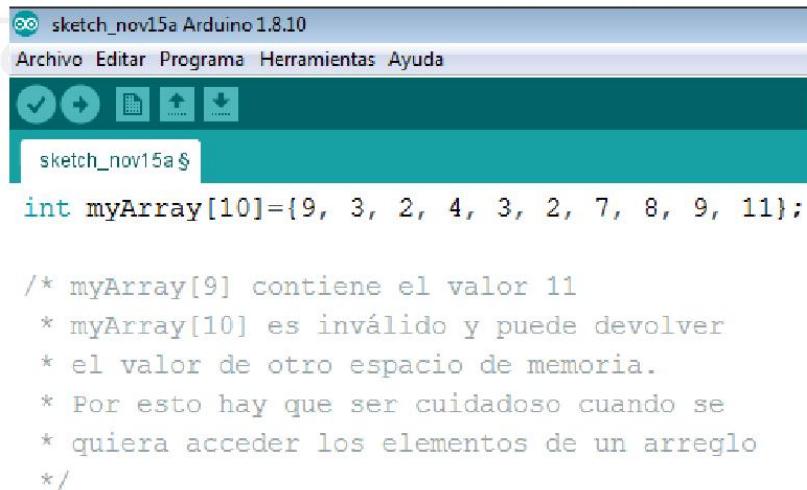
int myArray[10]={9, 3, 2, 4, 3, 2, 7, 8, 9, 11};

void setup() {
    myArray[0] = 10;
    /* previamente el índice 0 contenía el valor 9, ahora contiene 10.
     * Para obtener el valor de un elemento del array
     * se puede almacenar en una variable.
    */
    int x = myArray[2]; // ahora x contiene el valor 2.
}

void loop() {
}

```

En un arreglo que contenga 10 elementos, el último elemento estará en la posición 9.



```

sketch_nov15a Arduino 1.8.10
Archivo Editar Programa Herramientas Ayuda
sketch_nov15a §

int myArray[10]={9, 3, 2, 4, 3, 2, 7, 8, 9, 11};

/* myArray[9] contiene el valor 11
 * myArray[10] es inválido y puede devolver
 * el valor de otro espacio de memoria.
 * Por esto hay que ser cuidadoso cuando se
 * quiera acceder los elementos de un arreglo
*/

```

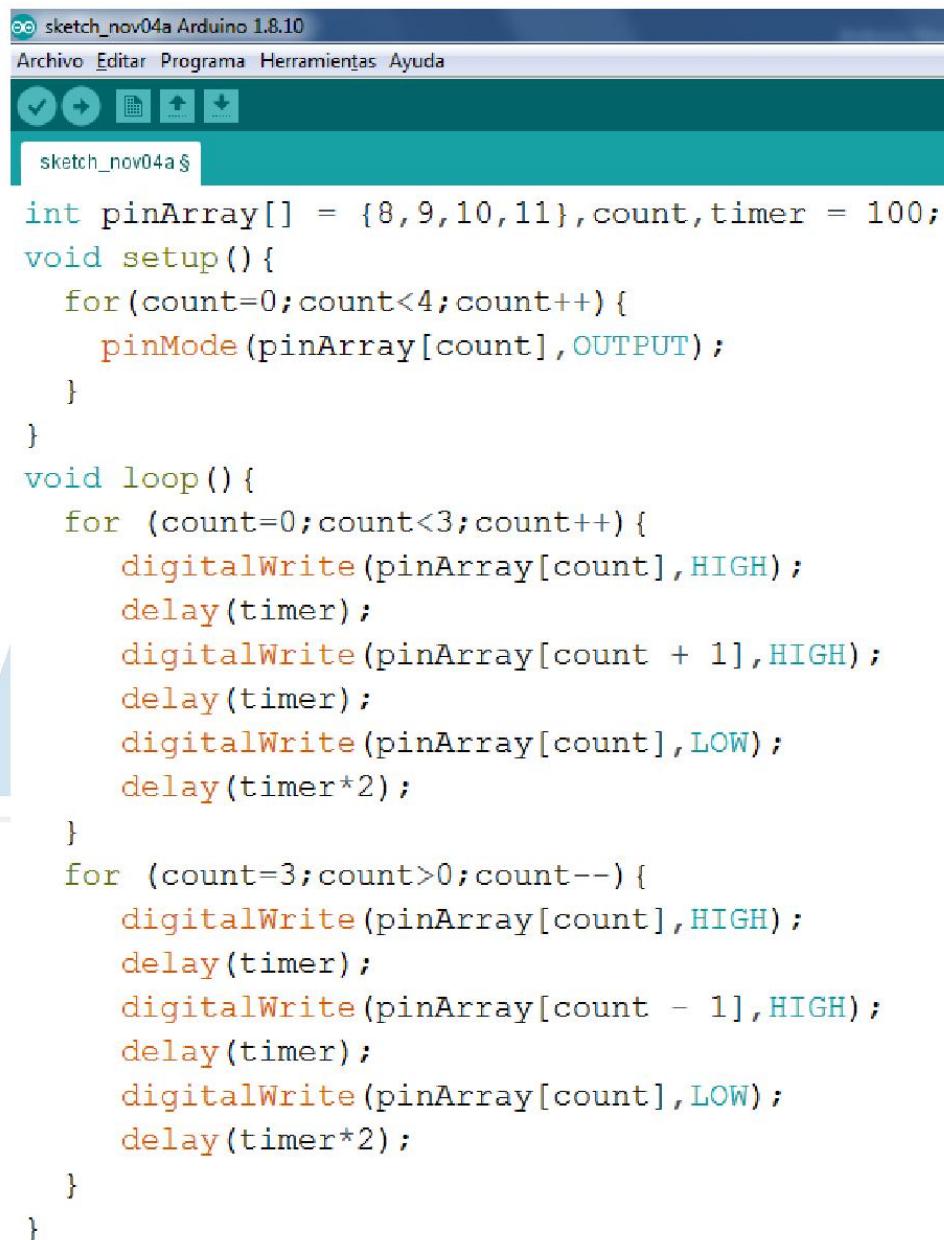
8.3.- Recorrer Arreglos

A menudo los arreglos son manipulados dentro de ciclos for, donde el contador del ciclo for es usado como el índice para cada elemento del arreglo. Por ejemplo, para imprimir los elementos de un arreglo en el Serial Monitor.

```
for(int i = 0; i < 5; i++){
    Serial.println(myArray[i]);
}
```

El siguiente ejemplo guarda el número de los pines dentro de un arreglo para posteriormente acceder a ellos a través de un ciclo for, los LEDs se encenderán de tal manera que parecerán luces de navidad.

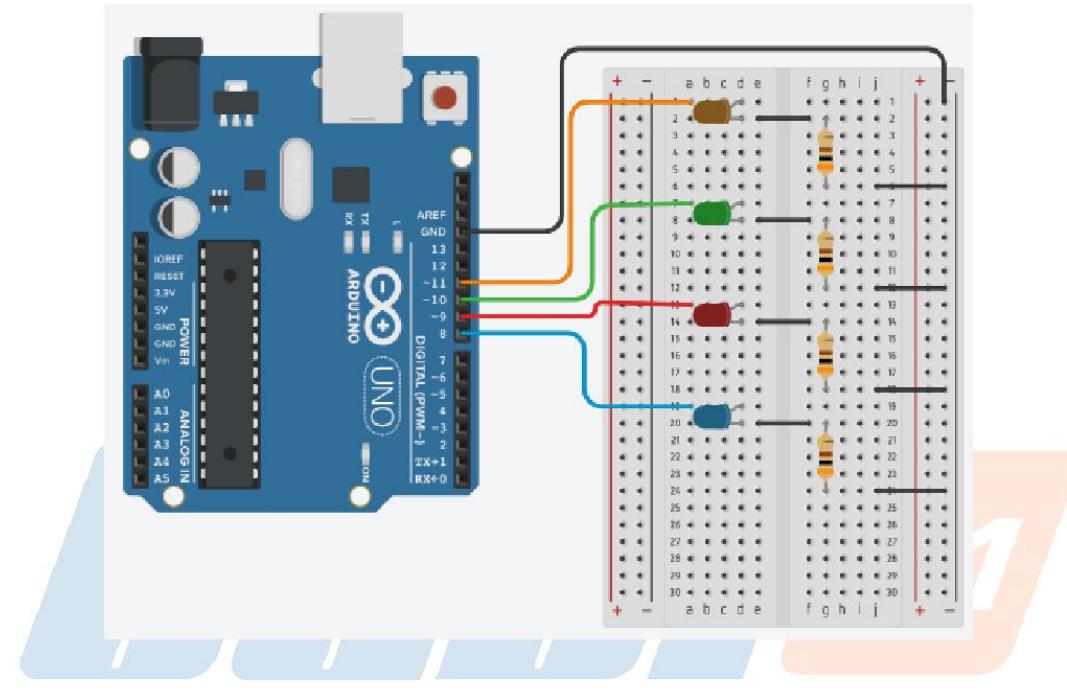




```
sketch_nov04a Arduino 1.8.10
Archivo Editar Programa Herramientas Ayuda
sketch_nov04a §

int pinArray[] = {8,9,10,11},count,timer = 100;
void setup() {
    for(count=0;count<4;count++) {
        pinMode(pinArray[count], OUTPUT);
    }
}
void loop() {
    for (count=0;count<3;count++) {
        digitalWrite(pinArray[count], HIGH);
        delay(timer);
        digitalWrite(pinArray[count + 1], HIGH);
        delay(timer);
        digitalWrite(pinArray[count], LOW);
        delay(timer*2);
    }
    for (count=3;count>0;count--) {
        digitalWrite(pinArray[count], HIGH);
        delay(timer);
        digitalWrite(pinArray[count - 1], HIGH);
        delay(timer);
        digitalWrite(pinArray[count], LOW);
        delay(timer*2);
    }
}
```

La siguiente imagen muestra el diagrama de conexión:

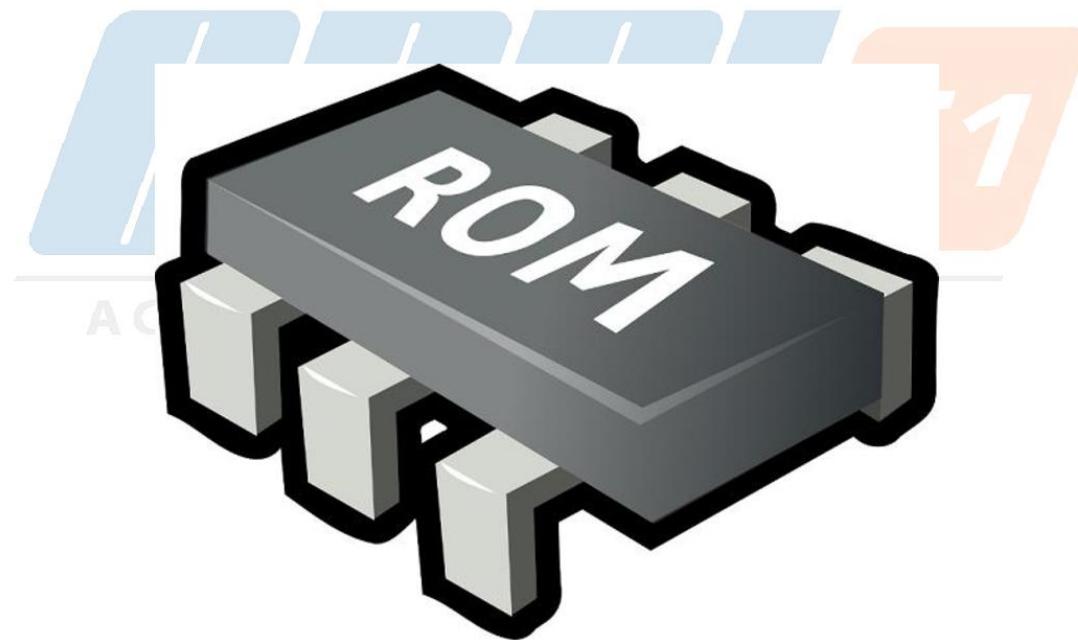


Capítulo 9. MEMORIA EEPROM

9.1.- Introducción

La memoria EEPROM (Electrically Erasable Programmable Memory) es una memoria que no pertenece a la memoria de programación (ROM, donde se almacena el programa), ni tampoco a la memoria de datos (RAM, donde se almacenan los valores de las variables), almacenar los valores incluso cuando el micro controlador esté apagado.

También poseen una vida limitada para borrar y reprogramar, vida que aproximadamente alcanza un millón de operaciones. Como es frecuentemente reprogramada mientras el micro controlador está en uso, la vida útil de estas memorias es una consideración de diseño importante.



Gracias a la memoria EEPROM, Arduino puede mantener los datos que se deseen guardar luego de apagarlo o reiniciarlo, para luego disponer de ellos al retomar la operación. En Arduino, el tamaño de esta memoria dependerá del microcontrolador de la placa:

- 512 bytes para el ATmega168P y ATmega8.
- 1024 bytes para el ATmega328P.
- 4KBytes para el ATmega1280 y ATmega2560.

De fábrica, todas las posiciones de memoria tienen escrito el valor por defecto de 255 o 0xFF.

9.2.- Leer la Memoria Eeprom

Para acceder a la memoria EEPROM se debe hacer uso la librería que lleva su nombre, esta librería está disponible de manera estándar en el IDE de Arduino. La librería es EEPROM.h y esta se debe incluir al principio del programa.

Funciones de lectura de la biblioteca EEPROM:

La función EEPROM.read() permite leer un byte en la posición de memoria que se indique por su parámetro.

Sintaxis:

```
EEPROM.read(direccion);
```

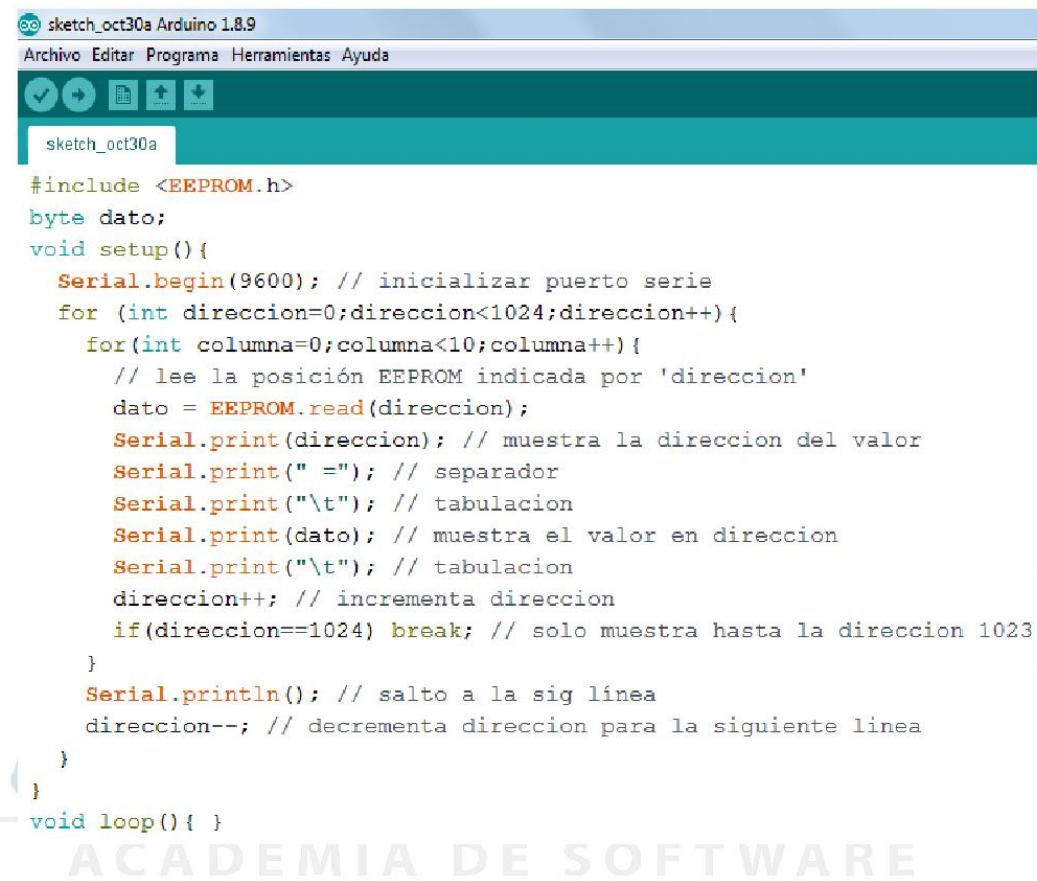
Parámetros:

dirección: la posición de memoria de 0 a 1023 (para la memoria de 1024 bytes).

Valor de retorno:

El valor almacenado en la posición de memoria indicada por parámetro.

El siguiente ejemplo muestra en el Serial Monitor el valor que posee cada posición de memoria de la EEPROM.

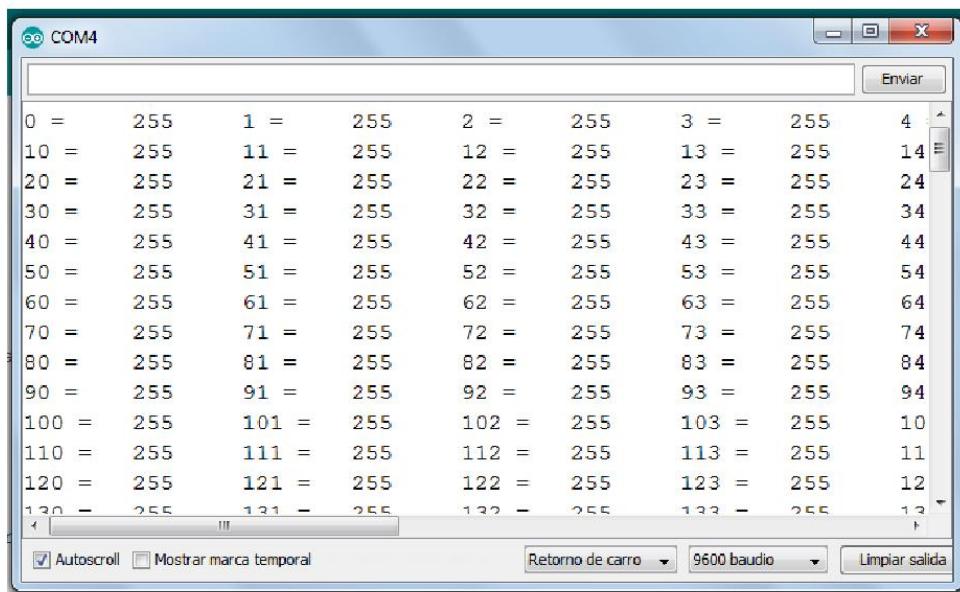


```
sketch_oct30a Arduino 1.8.9
Archivo Editar Programa Herramientas Ayuda
sketch_oct30a

#include <EEPROM.h>
byte dato;
void setup() {
    Serial.begin(9600); // inicializar puerto serie
    for (int direccion=0;direccion<1024;direccion++){
        for(int columna=0;columna<10;columna++){
            // lee la posición EEPROM indicada por 'direccion'
            dato = EEPROM.read(direccion);
            Serial.print(direccion); // muestra la dirección del valor
            Serial.print(" =");
            Serial.print("\t"); // tabulación
            Serial.print(dato); // muestra el valor en dirección
            Serial.print("\t");
            direccion++; // incrementa dirección
            if(direccion==1024) break; // solo muestra hasta la dirección 1023
        }
        Serial.println(); // salto a la siguiente línea
        direccion--; // decrementa dirección para la siguiente línea
    }
}
void loop() { }
```

ACADEMIA DE SOFTWARE

Como se puede observar, de fábrica, cada posición de memoria de la EEPROM posee el valor 255 o 0xFF (Hexadecimal)



La función EEPROM.get(): permite leer cualquier tipo de dato de la EEPROM.

Esta función a diferencia de read() lee una determinada cantidad de bytes dependiendo del tipo de dato que se le indique por parámetros.

Sintaxis:

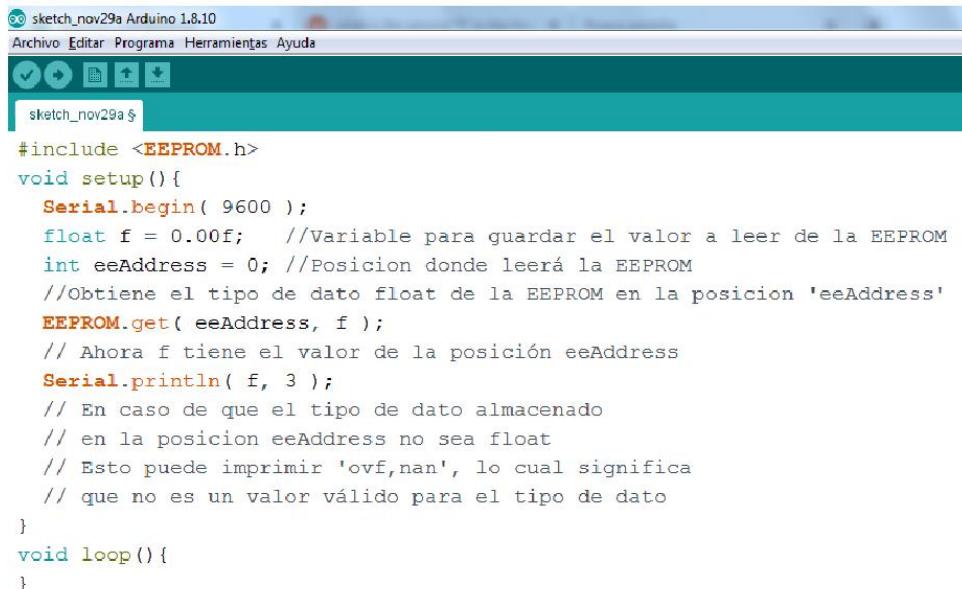
```
EEPROM.get(direccion,data)
```

Parámetros:

dirección: posición de memoria (de 0 hasta 1023 para memorias de 1024 bytes)

data: variable del mismo tipo de dato del valor que se quiere leer.

El siguiente ejemplo obtiene un valor tipo float previamente almacenado en la EEPROM.



```

sketch_nov29a Arduino 1.8.10
Archivo Editar Programa Herramientas Ayuda
sketch_nov29a
#include <EEPROM.h>
void setup() {
    Serial.begin( 9600 );
    float f = 0.00f; //Variable para guardar el valor a leer de la EEPROM
    int eeAddress = 0; //Posicion donde leerá la EEPROM
    //Obtiene el tipo de dato float de la EEPROM en la posicion 'eeAddress'
    EEPROM.get( eeAddress, f );
    // Ahora f tiene el valor de la posición eeAddress
    Serial.println( f, 3 );
    // En caso de que el tipo de dato almacenado
    // en la posicion eeAddress no sea float
    // Esto puede imprimir 'ovf,nan', lo cual significa
    // que no es un valor válido para el tipo de dato
}
void loop() {
}

```

9.3.- Escribir la Memoria Eeprom

La función EEPROM.write() almacena el valor de un byte en la posición de memoria indicada por parámetros. La función posee dos parámetros:

-El primero es la dirección o posición de memoria donde se desee almacenar el valor del byte.

-El segundo es el valor que se desea almacenar en la EEPROM, este debe ser un valor comprendido entre 0 y 255.

Sintaxis:

EEPROM.write(direccion,valor);

Parámetros:

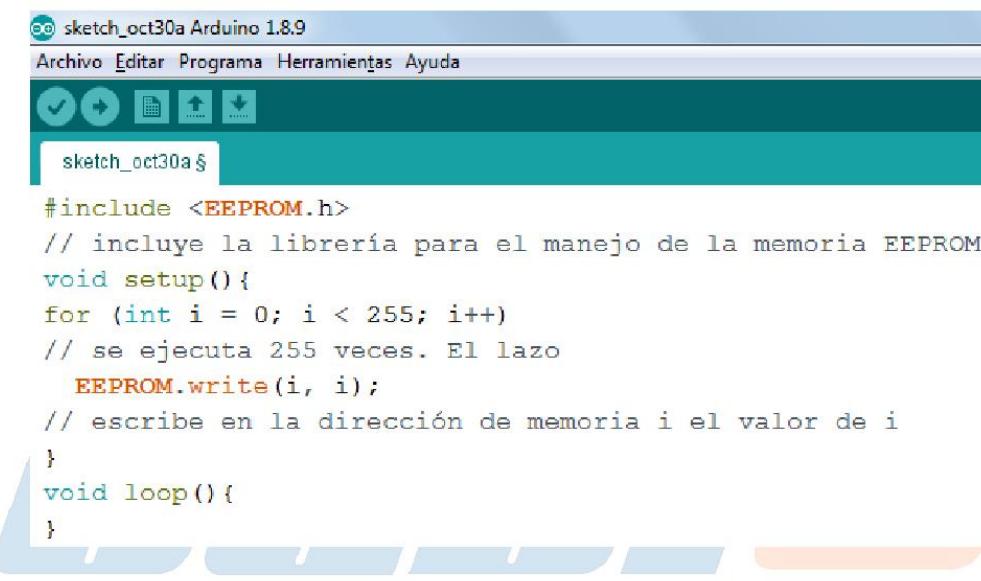
dirección: dirección de la memoria (0 a 1023).

valor: el valor a almacenar en la memoria.

Valor de retorno:

No posee.

El siguiente ejemplo con el uso de un ciclo for, se almacenará en la memoria EEPROM el valor de i en la posición i, el ciclo se ejecutará 255 veces.



The screenshot shows the Arduino IDE interface with the following details:

- Title Bar:** sketch_oct30a Arduino 1.8.9
- Menu Bar:** Archivo Editar Programa Herramientas Ayuda
- Toolbars:** Standard toolbar with icons for file operations.
- Code Editor:** The code is displayed in the main editor area:

```
#include <EEPROM.h>
// incluye la librería para el manejo de la memoria EEPROM
void setup(){
    for (int i = 0; i < 255; i++)
        // se ejecuta 255 veces. El lazo
        EEPROM.write(i, i);
    // escribe en la dirección de memoria i el valor de i
}
void loop(){
}
```

La función EEPROM.update() almacena un byte en la memoria EEPROM. El valor es solamente escrito solo si es diferente al valor que esta previamente almacenado en esa posición.

Sintaxis:

EEPROM.update(direccion,valor)

Parámetros:

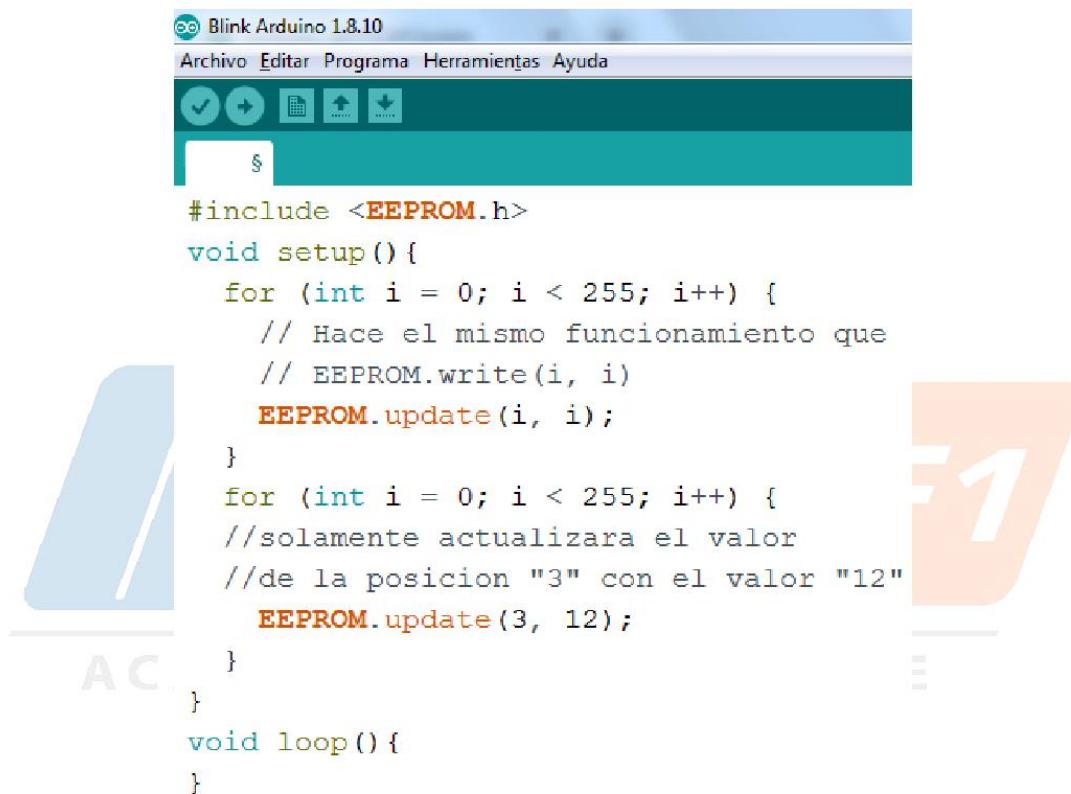
dirección: la posición de memoria (0 a 1023 para memorias de 1024 bytes).

valor: el valor a almacenar en la memoria.

Valor de retorno:

No posee.

El siguiente ejemplo, muy parecido al anterior, escribe el valor i en la posición i (solamente si el valor a almacenar es diferente al que está ya almacenado). El siguiente ciclo, aunque se repita 255 veces, solamente actualizará el valor de la posición 3 una única vez.



```
#include <EEPROM.h>
void setup() {
    for (int i = 0; i < 255; i++) {
        // Hace el mismo funcionamiento que
        // EEPROM.write(i, i)
        EEPROM.update(i, i);
    }
    for (int i = 0; i < 255; i++) {
        //solamente actualizara el valor
        //de la posicion "3" con el valor "12"
        EEPROM.update(3, 12);
    }
}
void loop() {
```

La función EEPROM.put() escribe cualquier tipo de dato en la EEPROM. A diferencia de write(), esta función permite almacenar más de 1 byte en la memoria, esta cantidad de bytes dependerá del tipo de dato de la variable que se le indique por parámetros.

Sintaxis:

EEPROM.put(direccion,valor)

Parámetros:

dirección: la posición de la memoria (de 0 a 1023 para memorias de 1024 bytes).

valor: el valor a escribir en la memoria.

El siguiente ejemplo escribe en la EEPROM el valor de la variable f

```
#include <EEPROM.h>
void setup() {
    Serial.begin(9600);
    float f = 123.456f; //variable a guardar en la EEPROM.
    int eeAddress = 0; //Posicion donde se almacenará la variable.
    //Una simple llamada para guardar en la EEPROM.
    EEPROM.put(eeAddress, f);
    Serial.println("Tipo de dato float guardado!");
}
void loop() {
```

9.4.- Operador Eeprom

El operador EEPROM[] permite usar el identificador "EEPROM" como un arreglo, tanto para leer valores como para escribir o asignar valores en una posición específica. Las posiciones o celdas del EEPROM pueden ser directamente leídas o escritas usando este método.

Sintaxis

EEPROM[address]

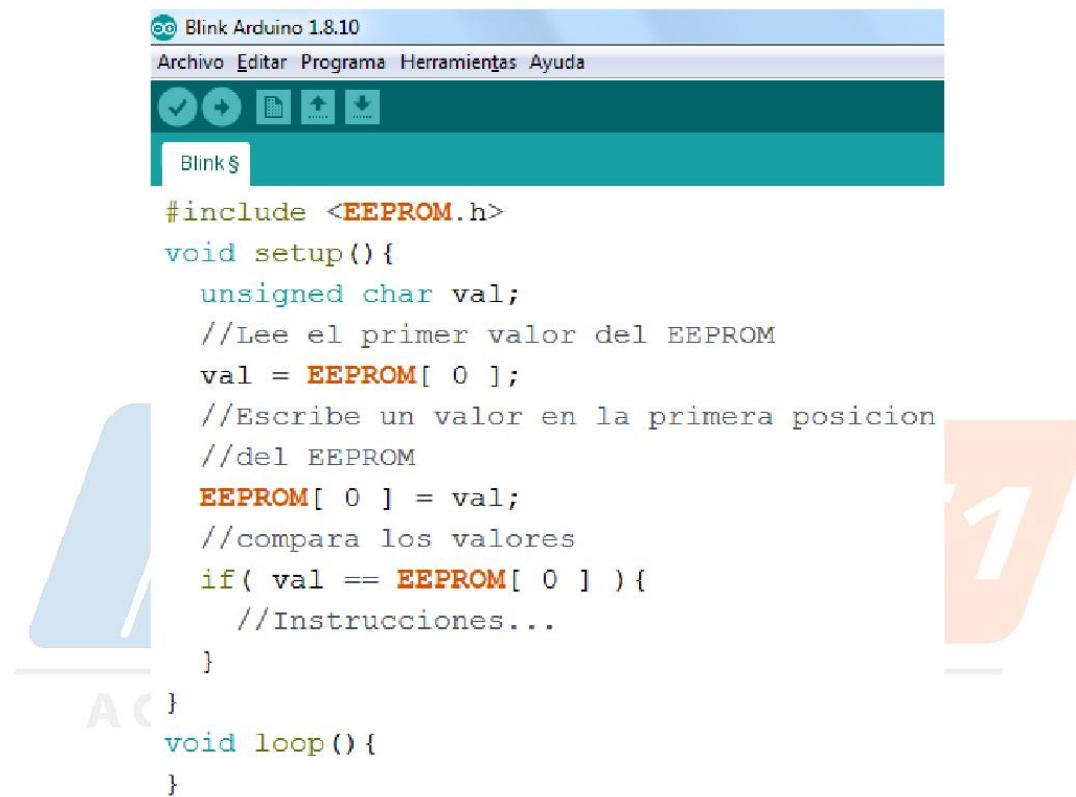
Parámetros:

address: la posición a leer o escribir (de 0 a 1023 para memorias de 1024 bytes).

Valor de retorno

Una referencia a la celda EEPROM

El siguiente ejemplo muestra como escribir y obtener el valor de la EEPROM usando el operador EEPROM[].



```
#include <EEPROM.h>
void setup() {
    unsigned char val;
    //Lee el primer valor del EEPROM
    val = EEPROM[ 0 ];
    //Escribe un valor en la primera posicion
    //del EEPROM
    EEPROM[ 0 ] = val;
    //compara los valores
    if( val == EEPROM[ 0 ] ){
        //Instrucciones...
    }
}
void loop() {
```

Capítulo 10. INTERRUPCIONES

10.1.- Introducción

Las interrupciones son un mecanismo muy valioso en los microprocesadores. Hasta ahora, la única forma de detectar el cambio de estado en una entrada del Arduino, se ha utilizado el método de consultar en el loop repetidamente el valor de la entrada con un intervalo de tiempo (delay) entre consultas. Este mecanismo es llamado “polling” y tiene 3 desventajas:

- Supone un continuo consumo del procesador y de energía, al tener que preguntar continuamente por el estado de la entrada.
- Si la acción necesita ser atendida inmediatamente, por ejemplo, una alerta de colisión, se debe esperar hasta el punto del programa donde se realiza la consulta.
- Si el pulso es muy corto, o si el procesador está ocupado haciendo otra tarea mientras se produce, es posible que se salte el disparo y nunca llegue a interpretarlo.

Para resolver este tipo de problemas, los microprocesadores incorporan el concepto de interrupciones. Una interrupción es un mecanismo que permite asociar una función (denominada como "callback") a la ocurrencia de un determinado evento.

Esta función "callback" asociada al evento se denomina ISR (Interrupt Service Routine). Cuando ocurre el evento, el procesador “sale” (o interrumpe lo que esté haciendo) inmediatamente del flujo normal del programa y ejecuta la función ISR asociada, ignorando por completo cualquier otra tarea. Al finalizar la función ISR asociada, el procesador vuelve al flujo principal, en el mismo punto donde había sido interrumpido.

Las interrupciones permiten realizar acciones que no serían posibles sin el uso de estas.

En general se disponen de dos tipos de interrupciones:

- las interrupciones de hardware: que corresponden a eventos ocurridos en ciertos pines físicos.
- las interrupciones de timers: denominadas interrupciones de software (se estudiarán en el nivel 3).

10.2.- Interrupciones en Arduino

Dentro de las interrupciones de hardware, Arduino es capaz de detectar los siguientes eventos:

- RISING: ocurre cuando hay un cambio de estado de subida de LOW a HIGH en el pin asociado.
- FALLING: ocurre cuando hay un cambio de estado de bajada de HIGH a LOW en el pin asociado.
- CHANGING: ocurre cuando el pin cambia de estado, ya sea de subida o de bajada (rising + falling) en el pin asociado.
- LOW: se ejecuta continuamente mientras el pin asociado está en estado LOW.

Los pines susceptibles de generar interrupciones son los pines digitales, aunque no todos. Los pines a los cuales se pueden asociar interrupciones varían en función del modelo de Arduino.

En la siguiente imagen se muestran la cantidad de interrupciones y los pines a los cuales se les puede asociar interrupciones en cada modelo de Arduino:

PLACA	INT0	INT1	INT2	INT3	INT4	INT5
Uno, Nano, mini Pro	D2	D3				
Mega2560	D2	D3	D21	D20	D19	D18
Leonardo, Micro	D3	D2	D0	D1	D7	

Para definir una interrupción en Arduino, en el setup del programa se utiliza la función:

```
attachInterrupt(interrupt, ISR, mode);
```

Parámetros:

interrupt: es el número de la interrupción (0 para INT0, 1 para INT1,...,5 para INT5).
 ISR: es la función de callback asociada.
 mode: son las opciones (eventos) disponibles (Falling, Rising, Change y LOW).

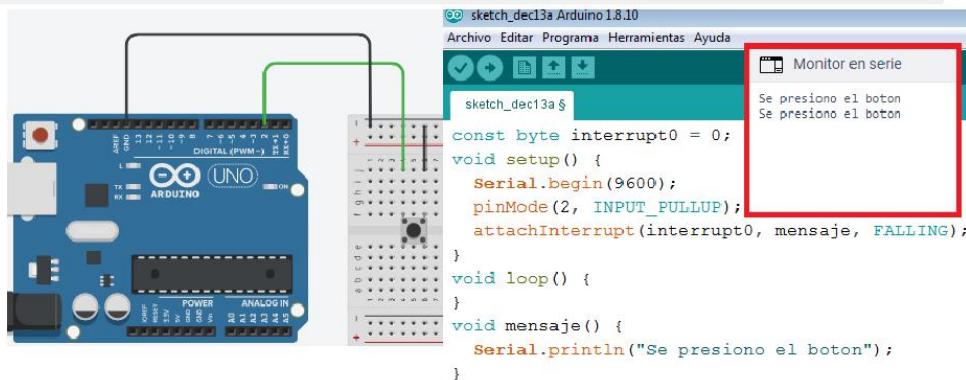
Es importante saber qué pin está asociado a la interrupción indicada por parámetros, ya que ese es el pin donde se tomará el evento para ejecutar dicha interrupción.

La función asociada a una interrupción se denomina ISR (Interruption Service Routines) y, por definición, tiene que ser una función que no recibe nada y no devuelve nada.

Dos ISR no pueden ejecutarse de forma simultánea. En caso de dispararse otra interrupción mientras se ejecuta una ISR, la función ISR se ejecutará después de la otra.

Al diseñar una ISR se debe mantener el objetivo de que tenga el menor tiempo de ejecución posible, ya que mientras esté ejecutándose, el resto de funciones del programa se encuentran detenidas.

En el siguiente ejemplo, cuando el pin 2 (asociado a la interrupción 0) cambie de estado a LOW, se activará la interrupción y llamará la ISR (blink) que mostrará un mensaje en el serial monitor:



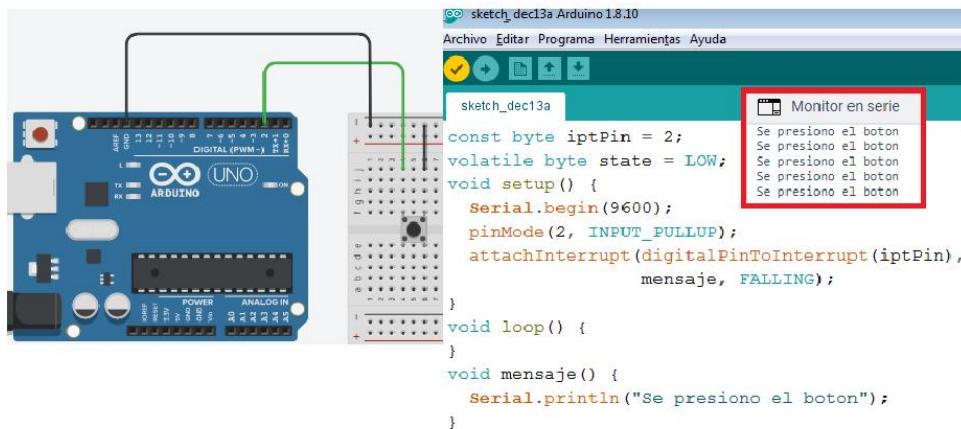
10.3.- Digitalpintointerrupt

Al configurar interrupciones, el programador debe conocer el número de la interrupción asociada al PIN que se está usando (o viceversa), de otro modo, no funcionará. Otro problema latente, es que de un modelo de Arduino a otro puede que para cierto número de interrupción no esté asociado el mismo número de PIN que donde se programó originalmente.

Por estas 2 razones, es mejor usar la función `digitalPinToInterruption()`, que dado un número de PIN retorna el número de interrupción asociado a éste. De esta forma se favorece el cambio de modelo de placa sin tener que modificar el código. Los pines susceptibles a interrupciones dependen del modelo de la placa:

PLACA	PINES SUSCEPTIBLES A INTERRUPCIONES
Uno, Nano, Mini	2, 3
Uno WiFi Rev.2	Todos los pines digitales
Mega, Mega2560, MegaADK	2, 3, 18, 19, 20, 21
Micro y Leonardo	0, 1, 2, 3, 7
Zero	Todos los pines digitales, excepto el 4
Due	Todos los pines digitales

El siguiente ejemplo hace uso de `digitalPinToInterruption(pin)` y el resultado es el mismo que el ejemplo anterior, con la diferencia, se beneficia la compatibilidad del programa con otras placas donde puede ejecutarse el mismo código:



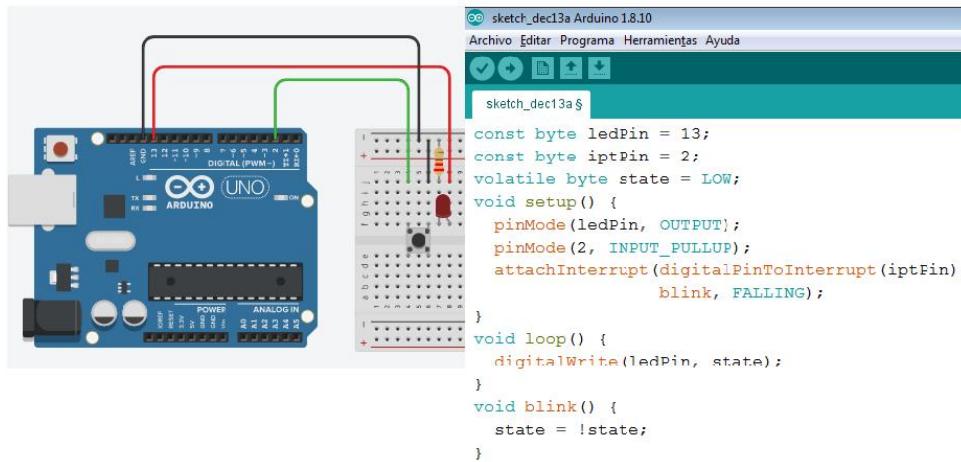
10.4.- Consideraciones Con Las Interrupciones

Para modificar una variable global dentro una ISR, la variable se debe declarar como "volatile".

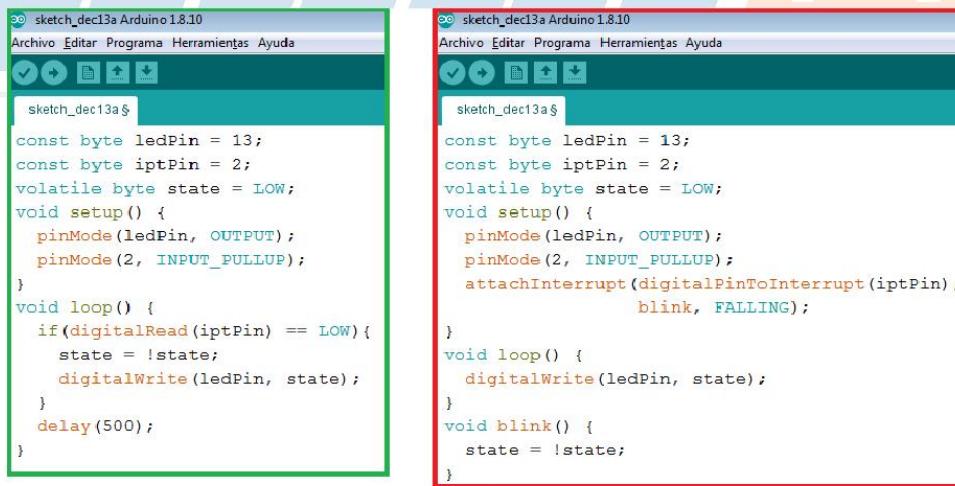
El modificador "volatile" indica al compilador que la variable tiene que ser consultada siempre antes de ser usada, dado que puede haber sido modificada de forma ajena al flujo normal del programa (lo que precisamente, hace una interrupción).

Al indicar una variable como volatile el compilador desactiva ciertas optimizaciones, lo que supone una pérdida de eficiencia. Por lo tanto, es recomendable declarar como volátil sólo las variables que sean utilizadas en interrupciones.

El siguiente ejemplo se declara una variable volatile, que se modifica en la ISR y es usada en el loop() del programa para encender o apagar el LED cada vez que se presione el botón:



En la siguiente imagen se aprecia la diferencia del código si se utiliza el método polling (en el recuadro verde) o las interrupciones (en el recuadro rojo) para el encendido y apagado de un LED cada vez que se presione un botón.



Las interrupciones tienen efectos en la medición del tiempo de Arduino, tanto fuera como dentro de la ISR, porque Arduino emplea interrupciones de tipo Timer para actualizar la medición del tiempo.

Durante la ejecución de una interrupción Arduino no actualiza el valor de la función millis() ni micros(), es decir, el tiempo de la ISR no se contabiliza y Arduino tiene un desfase en la medición del tiempo.

Si un programa tiene muchas interrupciones y estas suponen un alto tiempo de ejecución, la medida de tiempo de Arduino puede quedar distorsionada respecto a la realidad.

Dentro de la ISR, el resto de las interrupciones están desactivadas. Esto supone que:

- La función millis() no actualiza su valor, por lo que no se puede utilizar para medir el tiempo dentro de la ISR.
 - La función delay() no funciona, ya que basa su funcionamiento en la función millis().
 - La función micros() actualiza su valor dentro de la ISR, pero empieza a dar mediciones de tiempo inexactas pasado el rango de 500us.
- En consecuencia, la función delayMicroseconds() funciona en ese rango de tiempo, aunque se debe evitar su uso ya que no se debería introducir esperas dentro de una ISR.

Otras funciones para la gestión de interrupciones son:

- detachInterrupt(interrupt): anula una interrupción particular, especificada en el parámetro "interrupt".
- noInterrupts(): desactiva indefinidamente la ejecución de todas las interrupciones hasta que se vuelvan a activar.
- Interrupts(): reactiva las interrupciones si fueran desactivadas.

Capítulo 11. ARREGLOS. PARTE 2

11.1.- Matrices

Las matrices o arreglos multidimensionales son una estructura muy similar a los arreglos. Una matriz es una estructura de datos conformada por filas y columnas, que permite almacenar un conjunto de datos del mismo tipo.

Si una matriz tiene una única fila y columna no puede considerarse como tal.

The diagram shows a matrix structure with 'Filas' (Rows) and 'Columnas' (Columns). The matrix is labeled 'mat'. The matrix data is:

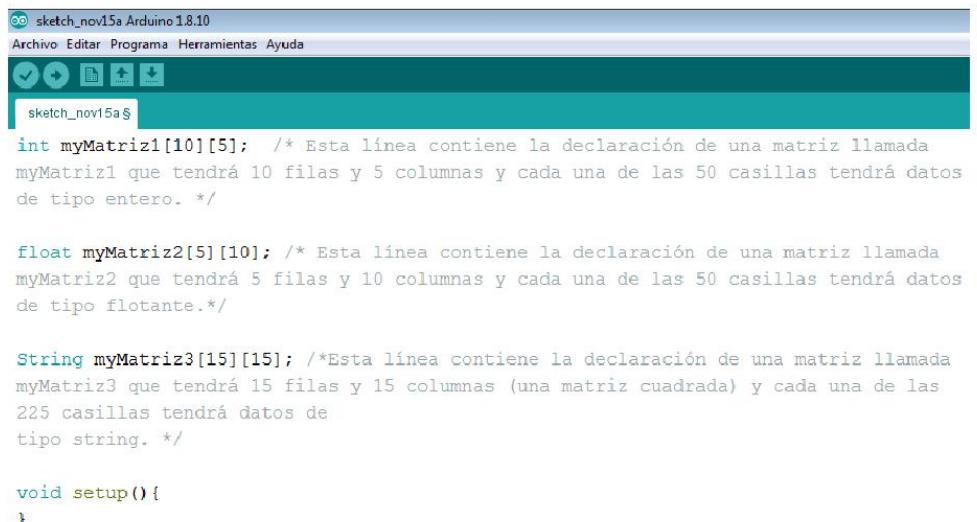
Filas	50	5	27	400	7
0	67	90	6	97	
30	14	23	251	490	

11.2.- Declaración de Matrices

Declarar una matriz es muy similar a la de un arreglo, se deben seguir las mismas normas, solamente hay un cambio en la sintaxis, la cual es agregarle el número de columnas.

```
TipoDeDatos nombreMatriz[filas][columnas];
```

Al crear una matriz es necesario saber la cantidad de información (filas y columnas) que esta contendrá.



```

sketch_nov15a Arduino 1.8.10
Archivo Editar Programa Herramientas Ayuda
sketch_nov15a $ 

int myMatriz1[10][5]; /* Esta linea contiene la declaración de una matriz llamada
myMatriz1 que tendrá 10 filas y 5 columnas y cada una de las 50 casillas tendrá datos
de tipo entero. */

float myMatriz2[5][10]; /* Esta linea contiene la declaración de una matriz llamada
myMatriz2 que tendrá 5 filas y 10 columnas y cada una de las 50 casillas tendrá datos
de tipo flotante.*/

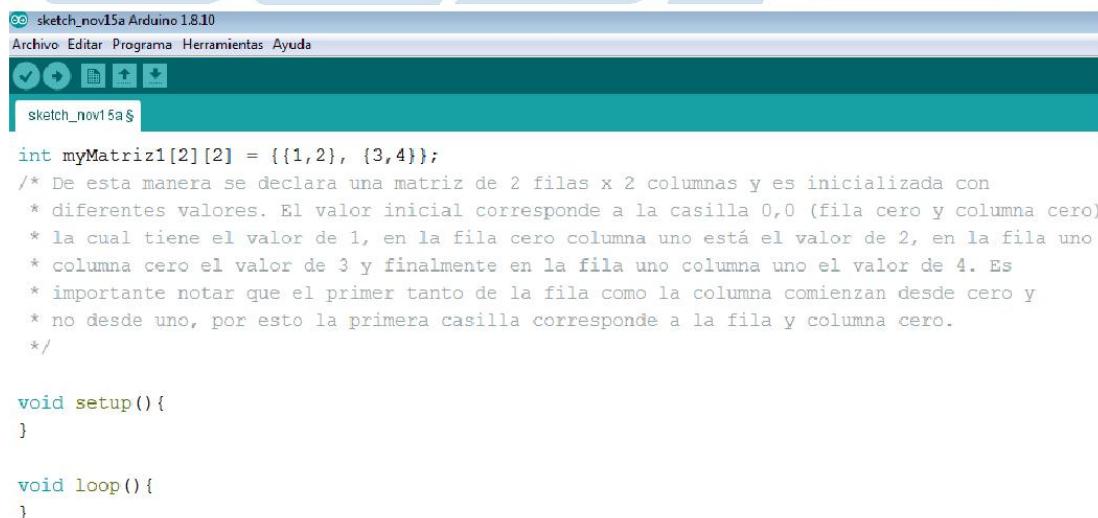
String myMatriz3[15][15]; /*Esta linea contiene la declaración de una matriz llamada
myMatriz3 que tendrá 15 filas y 15 columnas (una matriz cuadrada) y cada una de las
225 casillas tendrá datos de
tipo string. */

void setup() {
}

void loop() {
}

```

Al igual que con los arreglos, las matrices se pueden declarar e inicializar:



```

sketch_nov15a Arduino 1.8.10
Archivo Editar Programa Herramientas Ayuda
sketch_nov15a $ 

int myMatriz1[2][2] = {{1,2}, {3,4}};
/* De esta manera se declara una matriz de 2 filas x 2 columnas y es inicializada con
* diferentes valores. El valor inicial corresponde a la casilla 0,0 (fila cero y columna cero)
* la cual tiene el valor de 1, en la fila cero columna uno está el valor de 2, en la fila uno
* columna cero el valor de 3 y finalmente en la fila uno columna uno el valor de 4. Es
* importante notar que el primer tanto de la fila como la columna comienzan desde cero y
* no desde uno, por esto la primera casilla corresponde a la fila y columna cero.
*/

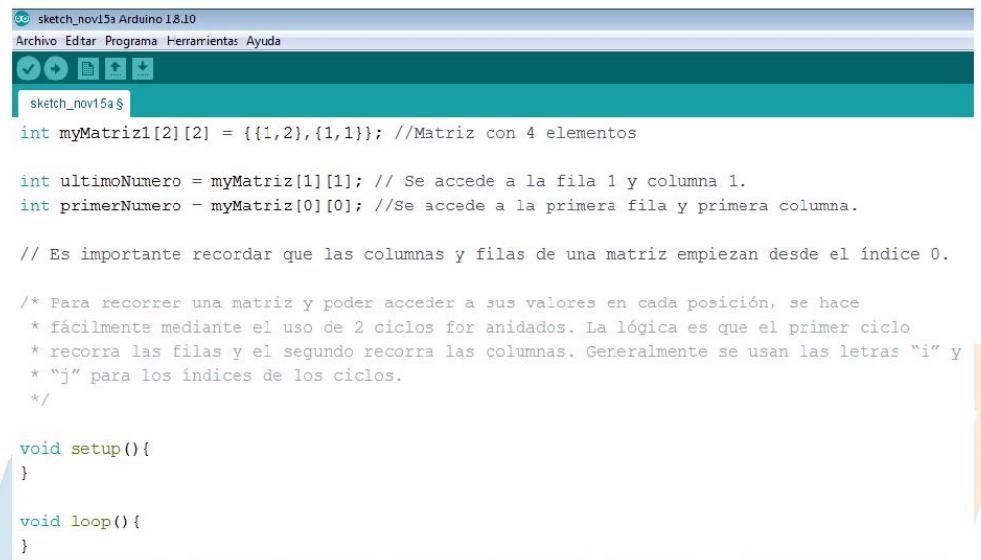
void setup() {
}

void loop(){
}

```

11.3.- Uso de Una Matriz

Para acceder al valor de una casilla específica hay que indicar a través de corchetes la posición de la matriz a la cual se desea acceder.



The screenshot shows the Arduino IDE interface with the following code:

```
sketch_nov15a Arduino 1.8.10
Archivo Editar Programa Herramientas Ayuda
sketch_nov15a
int myMatriz1[2][2] = {{1,2},{1,1}}; //Matriz con 4 elementos

int ultimoNumero = myMatriz[1][1]; // Se accede a la fila 1 y columna 1.
int primerNumero = myMatriz[0][0]; //Se accede a la primera fila y primera columna.

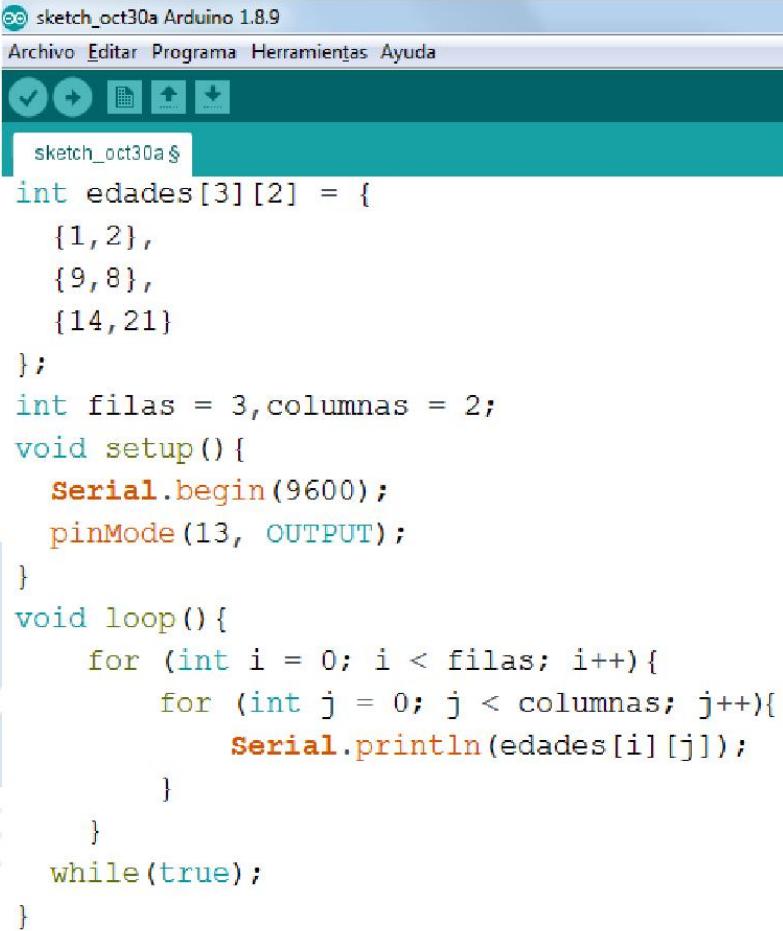
// Es importante recordar que las columnas y filas de una matriz empiezan desde el índice 0.

/* Para recorrer una matriz y poder acceder a sus valores en cada posición, se hace
 * fácilmente mediante el uso de 2 ciclos for anidados. La lógica es que el primer ciclo
 * recorra las filas y el segundo recorra las columnas. Generalmente se usan las letras "i" y
 * "j" para los índices de los ciclos.
 */

void setup() {
}

void loop() {
```

El siguiente ejemplo recorre toda la matriz y muestra el valor de cada casilla en el Serial Monitor.

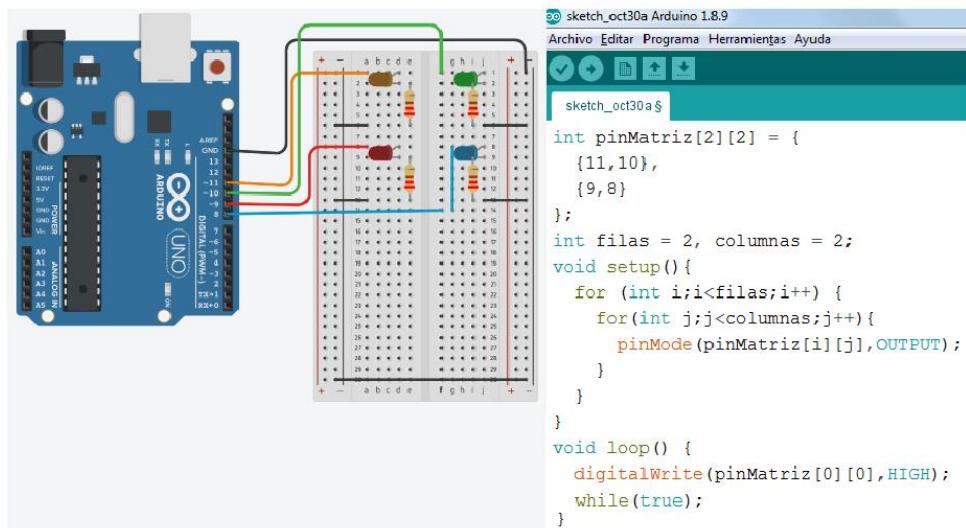


The screenshot shows the Arduino IDE interface with the following details:

- Title Bar:** sketch_oct30a Arduino 1.8.9
- Menu Bar:** Archivo Editar Programa Herramientas Ayuda
- Toolbar:** Includes icons for Open, Save, Print, Upload, and Refresh.
- Code Editor:** Displays the following C++ code:

```
int edades[3][2] = {  
    {1,2},  
    {9,8},  
    {14,21}  
};  
int filas = 3, columnas = 2;  
void setup(){  
    Serial.begin(9600);  
    pinMode(13, OUTPUT);  
}  
void loop(){  
    for (int i = 0; i < filas; i++){  
        for (int j = 0; j < columnas; j++){  
            Serial.println(edades[i][j]);  
        }  
    }  
    while(true);  
}
```

El siguiente ejemplo posee una matriz de leds 2x2, el programa enciende el LED amarillo ubicado en la primera fila y primera columna de la matriz.



ACADEMIA DE SOFTWARE

Capítulo 12. TECLADOS MATRICIALES

12.1.- Introducción

Un teclado matricial es un dispositivo que agrupa varios pulsadores y permite controlarlos empleando un número de conductores inferior al que se necesitaría al usarlos de forma individual (no se necesita un pin para cada botón del teclado matricial). Estos dispositivos agrupan los pulsadores en filas y columnas formando una matriz, disposición que da lugar a su nombre.

Es comúnmente encontrar este tipo de teclado en los electrodomésticos y en sistemas de seguridad o alarmas.



Los teclados matriciales son frecuentes en la electrónica e informática. Incluso los teclados de las computadoras son teclados matriciales. Una de las desventajas de usar un teclado matricial es que pueden causar problemas cuando se pulsa más de una tecla simultáneamente.

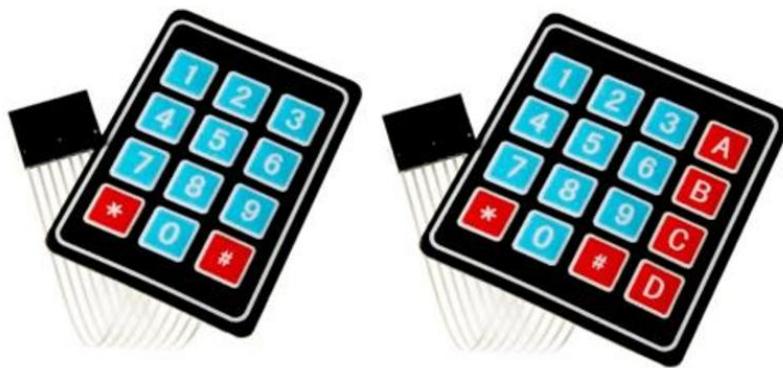
Este es uno de los motivos por el que los teclados de computadora usan una disposición no rectangular, agrupando ciertas teclas en circuitos diferentes (Ctrl, Alt, Shift...).



12.2.- Conexión

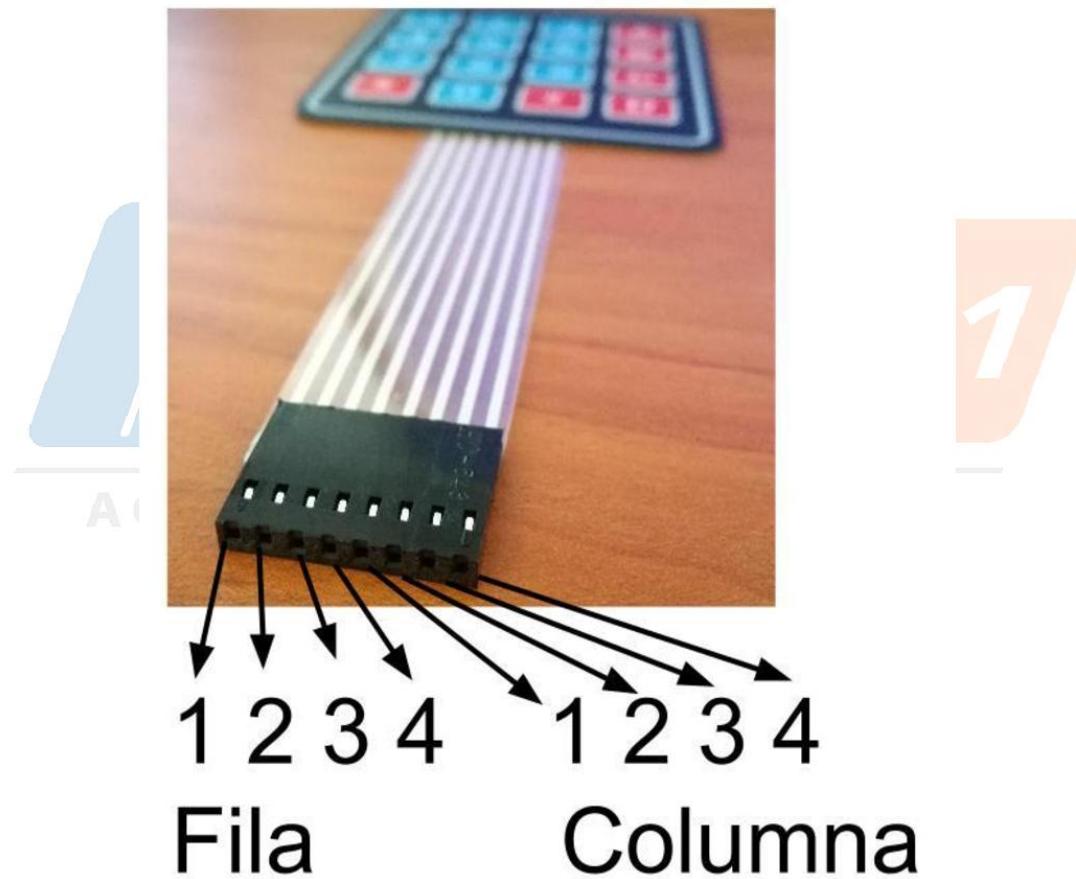
En el campo de Arduino, existen múltiples modelos de teclados matriciales en distintos soportes (rígidos o flexibles) y con distintos números de teclas, siendo habituales configuraciones de 3x3, 3x4, y 4x4.

Se pueden emplear teclados matriciales en los proyectos de electrónica y robótica, por ejemplo, para cambiar el modo de funcionamiento de un montaje, para solicitar una contraseña, teclas de dirección para controlar un brazo robótico o un vehículo, o proporcionar instrucciones a un robot.



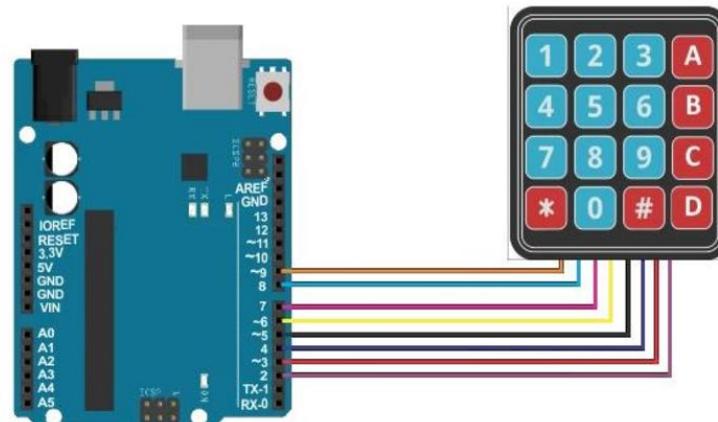
Los primeros pines de izquierda a derecha corresponden a las filas del teclado y los siguientes pines corresponden a las columnas, por lo que para un teclado 4x4 se puede visualizar el pinout en la imagen.

Para facilitar el uso de teclados matriciales en Arduino, se debe utilizar la librería Keypad.h la cual está disponible en el siguiente repositorio <https://github.com/Chris--A/Keypad>.



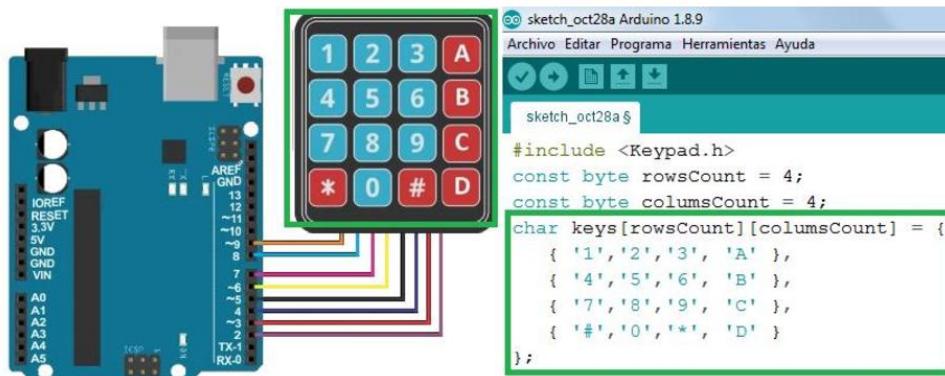
Para conectar el teclado matricial, los primeros 4 pines del teclado (filas) se conectan desde el pin digital 9 hasta el pin digital 6 del Arduino. Los 4 pines restantes del teclado

corresponden a las columnas, estos pines se conectan desde el pin digital 5 hasta el pin digital 2.

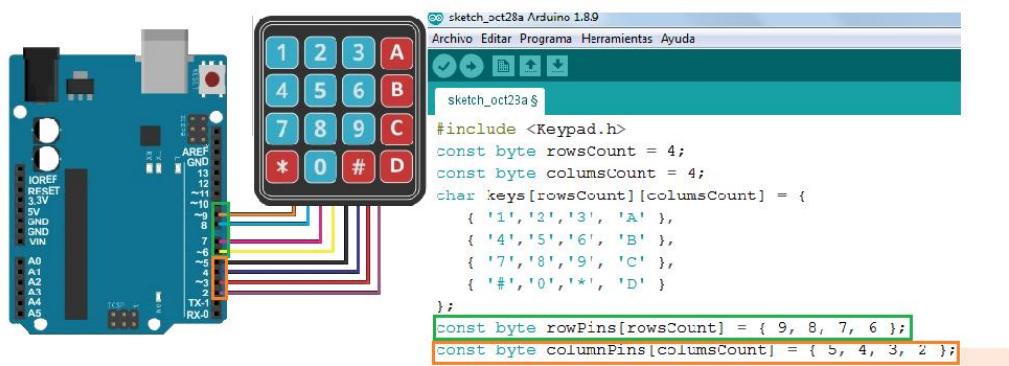


12.3.- Programación

En el código, se incluye la librería previamente descargada. Se crean 2 constantes de valor 4 que representan las filas y las columnas del teclado. Se crea una matriz de tipo char para representar las letras del teclado, que debe ser inicializada al momento de ser declarada con valores que deben coincidir con los del teclado. El tamaño de la matriz se define usando las constantes que representan las filas y las columnas.



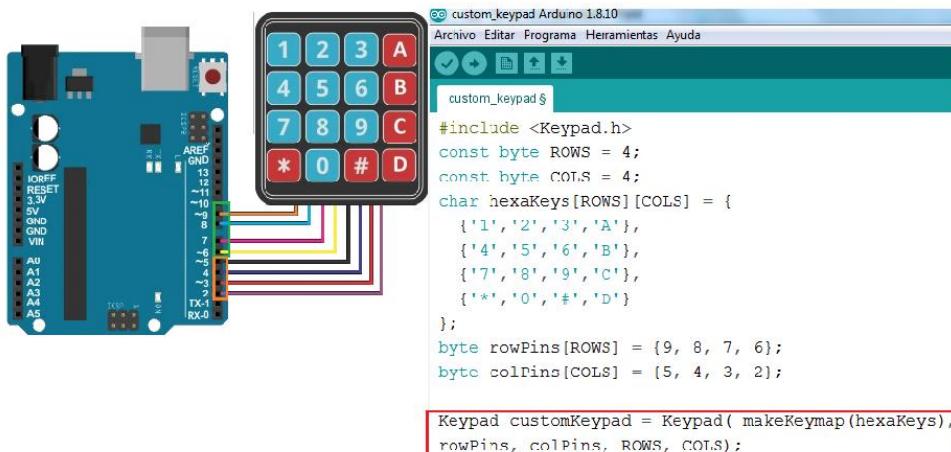
Se crean 2 constantes adicionales, la primera constante es un arreglo donde indica los pines del Arduino donde están conectadas las filas del teclado matricial. La segunda constante representa los pines donde están conectados las columnas del teclado matricial.



La función Keypad configura las letras, pulsadores y pines asociados al teclado. Esta función recibe como parámetros:

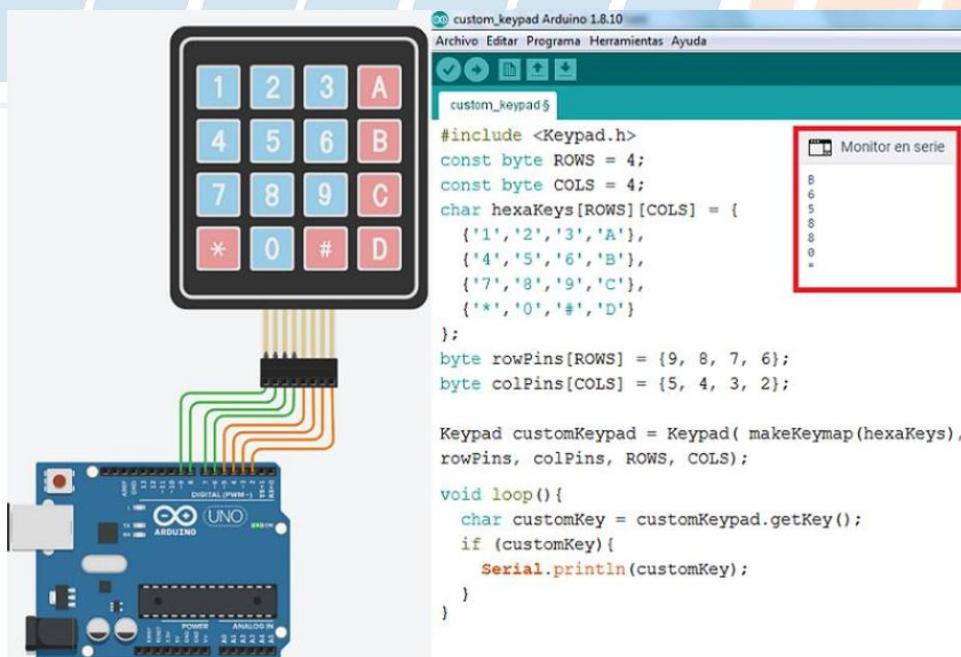
- makeKeymap(hexaKeys): Mapea los caracteres de la matriz y los asocia a los pulsadores.
- rowPins: El arreglo de los pines conectado a las filas del teclado.
- colPins: El arreglo de los pines conectado a las columnas del teclado.
- ROWS: La cantidad de filas que posee el teclado.
- COLS: La cantidad de columnas que posee el teclado.

La siguiente imagen es un ejemplo de la inicialización de un teclado:



El método que se utiliza para obtener la letra que esté siendo presionada es getKey().

En el siguiente ejemplo se muestra en el Serial Monitor la tecla pulsada por el usuario en el teclado matricial.



Capítulo 13. SENSORES. PARTE 1

13.1.- Introducción

Los campos que abarca Arduino son muy extensos, al ser un proyecto open source muchas personas desarrollan sensores, librerías y proyectos. Entre los sensores que se pueden utilizar en un proyecto de Arduino están:

- Ultrasónicos.
- Infrarrojos.
- Detector de movimiento.
- Sensores de humedad y temperatura.
- Sensor de humo.
- Sensor de lluvia.
- Sensores de inclinación.



Un sensor de temperatura es un componente capaz de percibir la temperatura y/o humedad del exterior y lo transforma en una señal digital o analógica, que luego es enviada a la placa Arduino.

Existen diferentes tipos de sensores de temperatura, donde la precisión que estos ofrecen es la principal característica con los que se diferencian unos de otros. Otra característica importante por la que se distinguen es la temperatura máxima y mínima que admiten.

13.2.- Sensor de Temperatura Tmp36

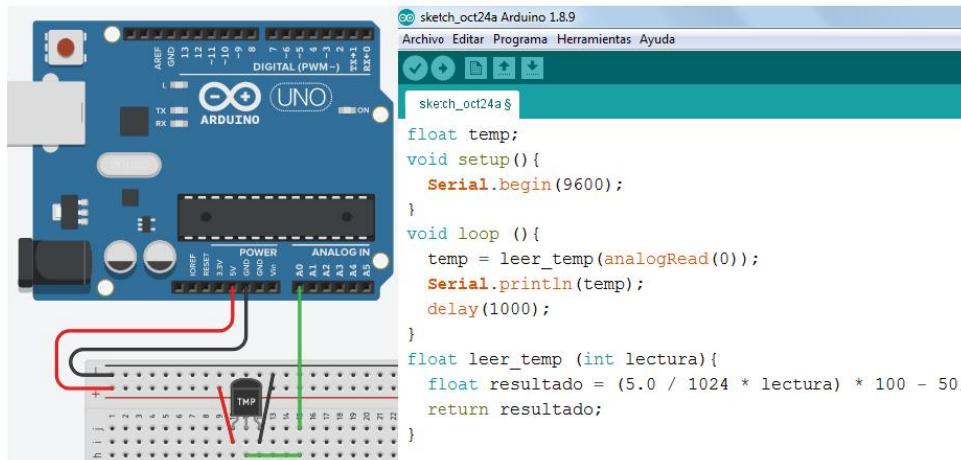
El TMP36 es un sensor analógico. Por lo tanto, su valor de tensión es proporcional a la temperatura que se encuentre el exterior. Una de sus características es que funciona entre -50°C y 125°C.

Para saber cómo funcionan los componentes electrónicos hay que leer su datasheet. El fabricante del TMP36 indica que la tensión será de 10mV por cada grado de temperatura, por lo tanto, para una lectura dada, el valor de la temperatura medida será (ver formula):

Donde $5/1024 * \text{lectura}$ son los valores de Voltaje para cada lectura dada, $*100$ es la conversión de Voltios a milivoltios y -50 ya que el fabricante indica que para cuando la lectura sea 0, la temperatura será de -50°C.

$$\text{TempC} = \frac{5}{1024} * \text{lectura} * 100 - 50$$

El siguiente ejemplo muestra el diagrama y código para leer la temperatura usando el sensor TMP36:



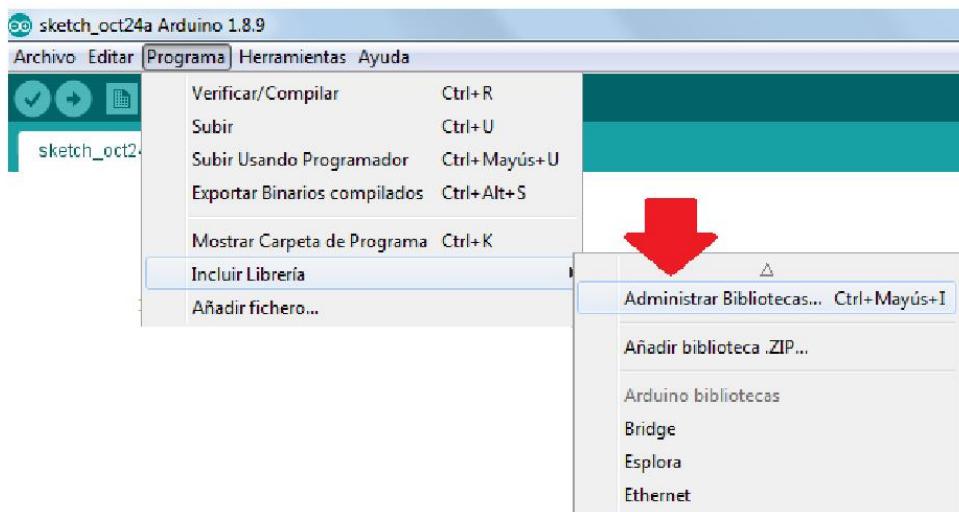
13.3.- Sensor de Temperatura Dht22

El sensor DHT22 es un sensor de temperatura digital, no solamente recoge la temperatura, sino que además recoge la humedad del ambiente. La señal es enviada a Arduino a través de una señal digital de 16 bits. Las temperaturas que soporta rondan los -40°C y los 80°C.

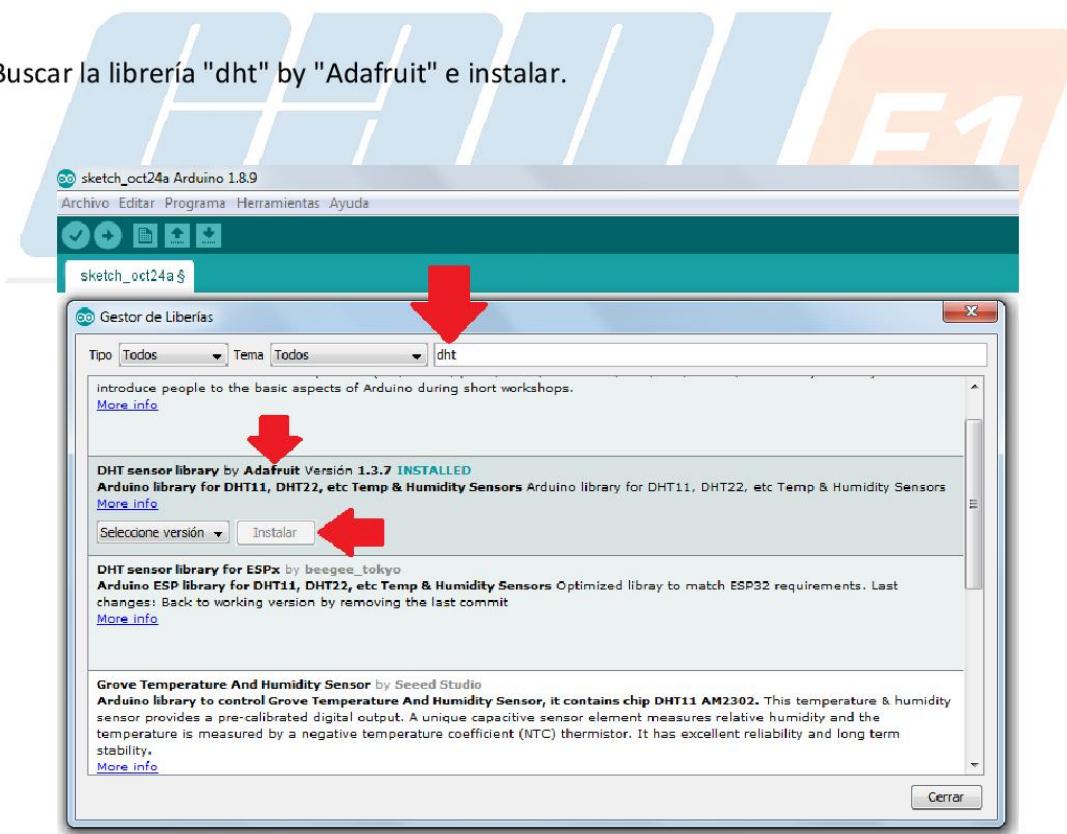
El DHT22 usa su propio sistema de comunicación bidireccional mediante un único conductor, empleando señales temporizadas.

En cada envío de medición el sensor envía un total de 40bits, en 4ms. Estos 40 bits corresponden con 2 Bytes para la medición de humedad, 2 Bytes para la medición de temperatura, más un Byte final para la comprobación de errores.

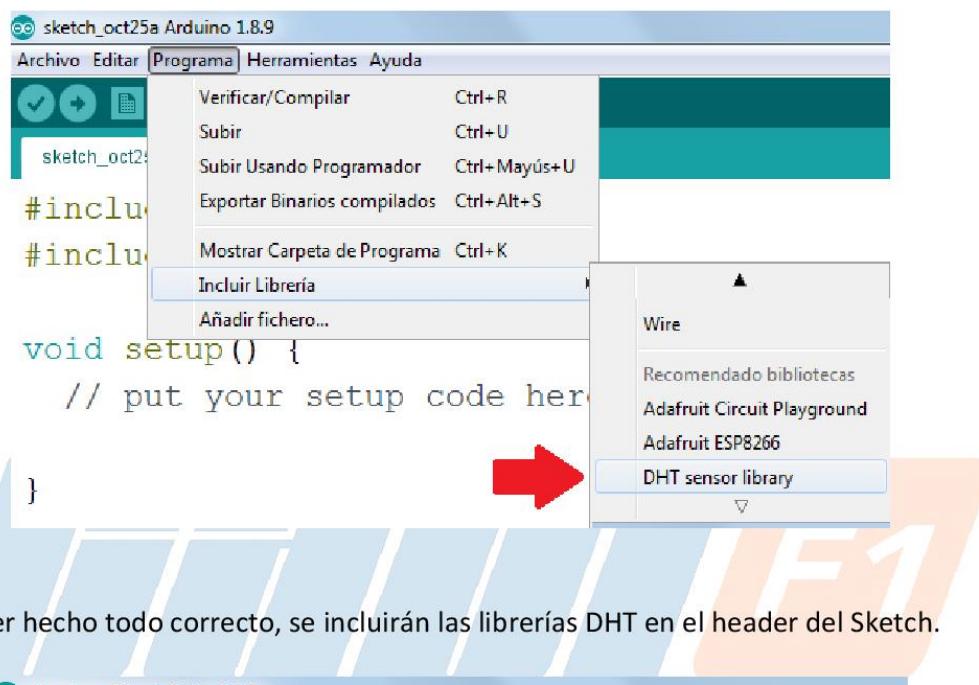
Este sensor al abarcar más funciones y al ser digital es más complejo el cálculo para obtener su lectura, por lo tanto, para facilitar todo esto se hace uso de la librería DHT.h, para ello necesitamos la librería de la siguiente manera: Programa -> incluir librería -> Administrar bibliotecas.



Buscar la librería "dht" by "Adafruit" e instalar.



Se incluye la librería DHT en el Sketch de Arduino.

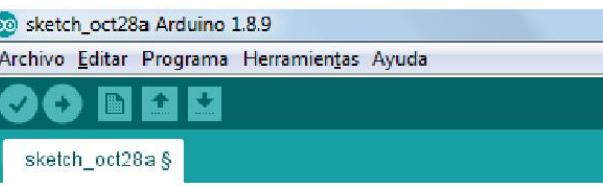


De haber hecho todo correcto, se incluirán las librerías DHT en el header del Sketch.

```
#include <DHT.h>
#include <DHT_U.h>
void setup() {
    // put your setup code here, to run once:
}

void loop() {
    // put your main code here, to run repeatedly:
}
```

Se definen las variables donde se almacenarán las lecturas de temperatura y humedad. Posteriormente se imprimen las lecturas en el Serial Monitor cada 2 segundos.



```
#include <DHT.h>
#include <DHT_U.h>

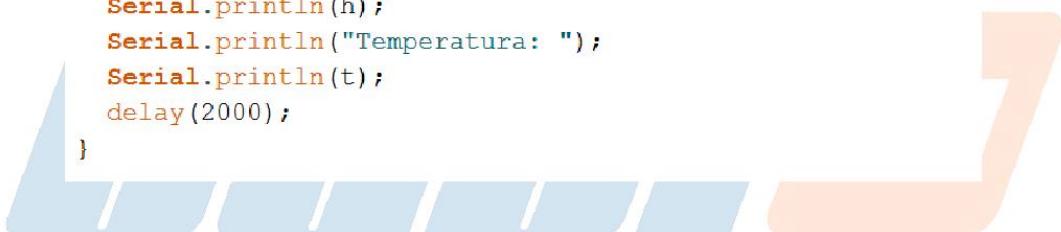
DHT dht(2, DHT22); // Se define una variable DHT con el pin a donde está conectado el sensor y el tipo de sensor DHT22

void setup() {
    Serial.begin(9600);
    dht.begin(); // Se inicia el sensor
}

void loop() {
}
```



Este código sirve tanto para el sensor DHT11 y DHT22, solo hay que especificar el tipo de sensor cuando se cree la variable DHT.

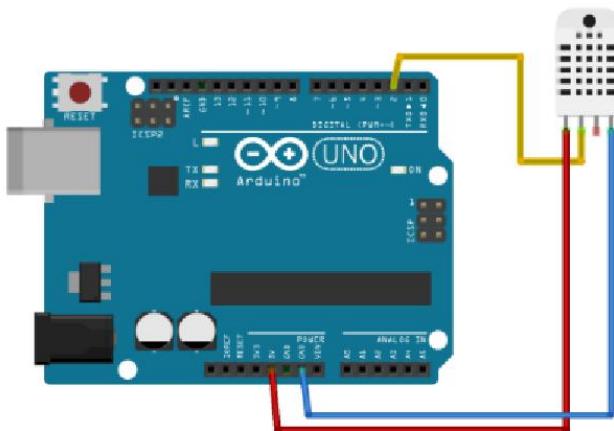


```
sketch_oct28a Arduino 1.8.9
Archivo Editar Programa Herramientas Ayuda
sketch_oct28a §

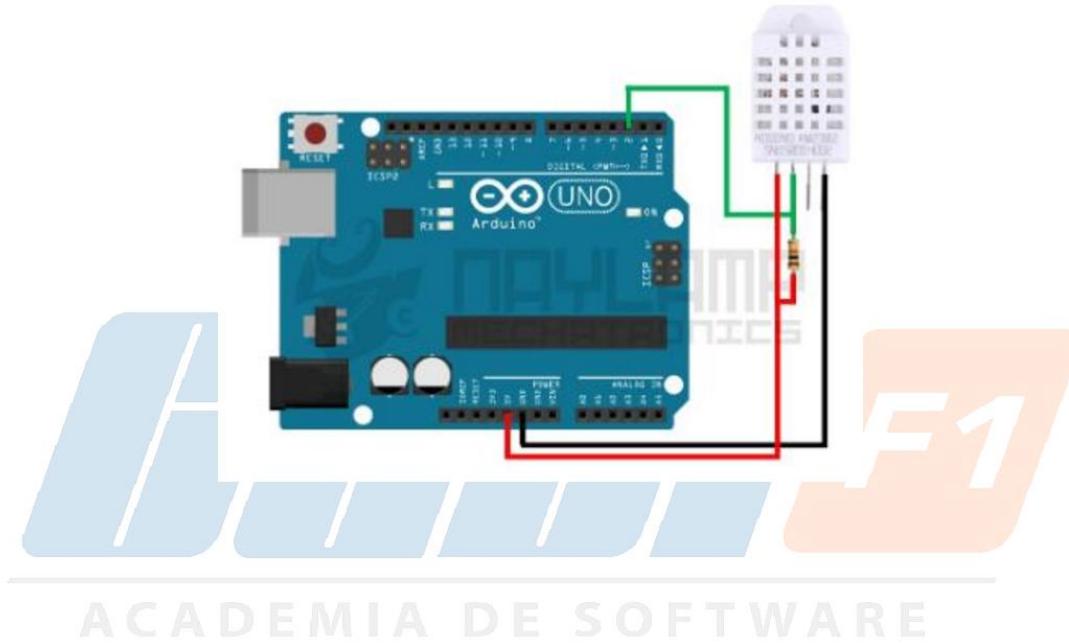
#include <DHT.h>
#include <DHT_U.h>
DHT dht(2,DHT22);
void setup() {
    Serial.begin(9600);
    dht.begin();
}
void loop() {
    float h = dht.readHumidity(); //Se lee la humedad
    float t = dht.readTemperature(); //Se lee la temperatura
    Serial.println("Humedad: ");
    Serial.println(h);
    Serial.println("Temperatura: ");
    Serial.println(t);
    delay(2000);
}
```

Diagrama de conexión del sensor

ACADEMIA DE SOFTWARE



Se puede hacer uso de una resistencia PULLUP en caso de obtener lecturas erróneas, aunque algunas veces no es necesario. La resistencia PULLUP puede ser de un valor entre 4.7K y 10K. Si se desea trabajar con lógica de 3.3v solo hay que cambiar la alimentación a dicho voltaje al igual que la resistencia PULLUP debe ir a 3.3V.



Capítulo 14. SENSORES. PARTE 2

14.1.- Sensor de Movimiento

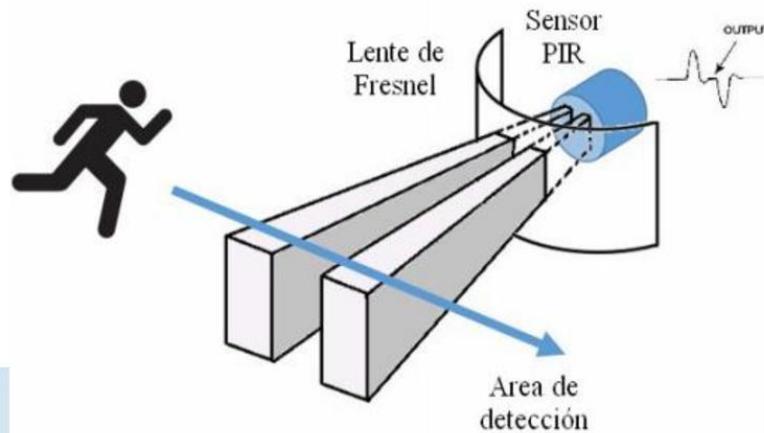
Todos los objetos con una temperatura por encima de 0°C emiten calor. Por lo general, esta radiación es invisible para el ojo humano, ya que irradia en longitudes de ondas infrarrojas, pero puede ser detectado por dispositivos electrónicos diseñados para tal propósito.

Los detectores PIR (Passive Infra Red) o Pasivo Infrarrojo, reaccionan sólo ante determinadas fuentes de energía tales como el calor del cuerpo humano o animales. Básicamente reciben la variación de las radiaciones infrarrojas del medio ambiente que cubre. Es llamado pasivo debido a que no emite radiaciones, sino que las recibe.

Una de sus aplicaciones principales de los sensores PIR es la detección de movimientos para las alarmas de seguridad, porque estos captan la diferencia entre el calor emitido por el cuerpo humano y el espacio alrededor.



En los sensores de movimiento, el sensor PIR consta de 2 elementos detectores separados, siendo la señal diferencial entre ambos la que permite activar la alarma de movimiento.

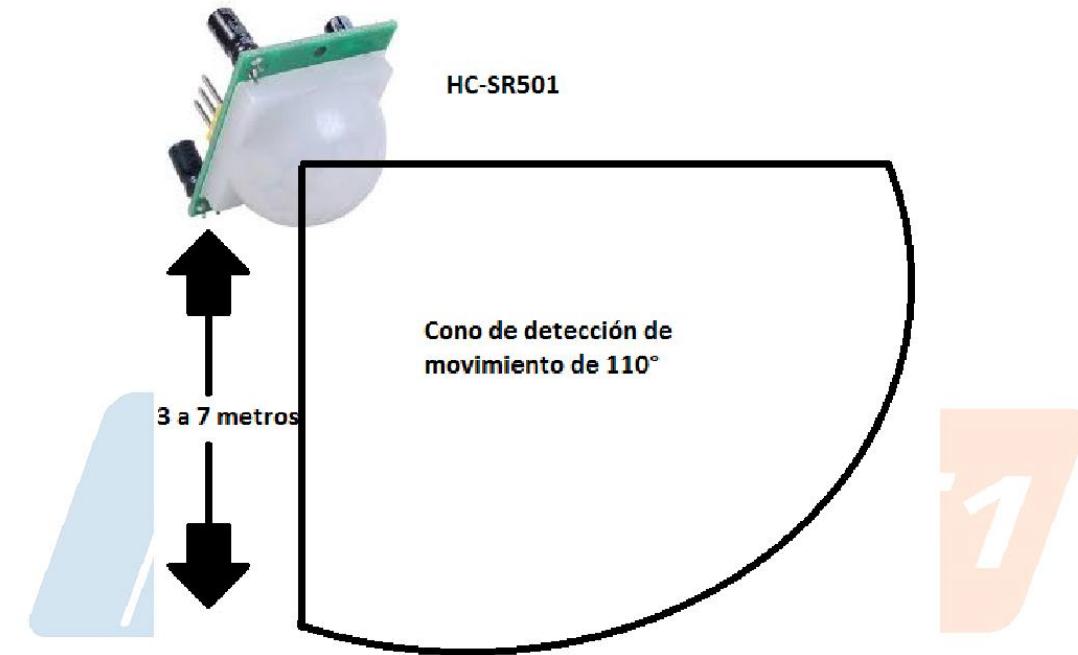


14.2.- Conexión

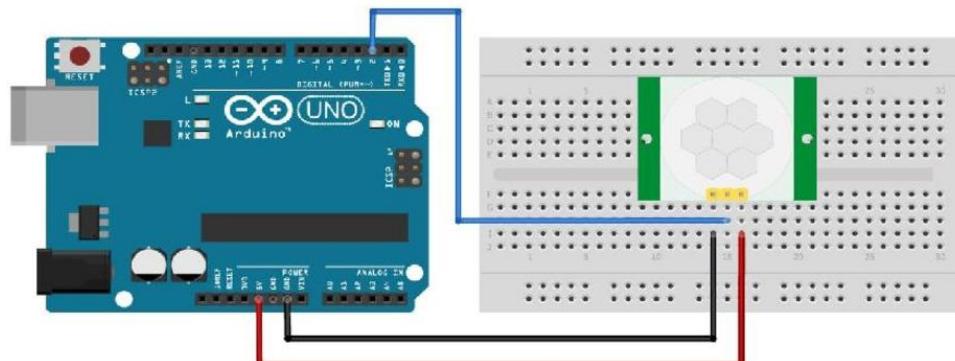
Uno de los modelos más comerciales de sensores es el PIR HC-SR501. El rango de detección de movimiento es ajustable y generalmente funciona con alcances de 3 hasta 7 metros y con aperturas de 90° a 110°. El montaje del PIR puede realizarse tanto en el piso, muro o en el techo, según convenga a la aplicación.



La distancia y velocidad de repuesta se pueden variar con los potenciómetros que posee el sensor.



En la siguiente imagen se muestra el diagrama de conexión del sensor PIR con el Arduino. Cabe destacar que dependiendo del fabricante los pines GND, VCC y DATA pueden variar, por lo que, es importante leer el datasheet antes de montar el sensor para evitar conexiones erróneas que puedan dañar el componente.



14.3.- Programación

En el siguiente ejemplo, cada vez que detecte movimiento el PIR, encenderá el LED y enviará un mensaje por el Serial Monitor de que se detectó movimiento. (Los pines de este sensor PIR es diferente al anterior).

