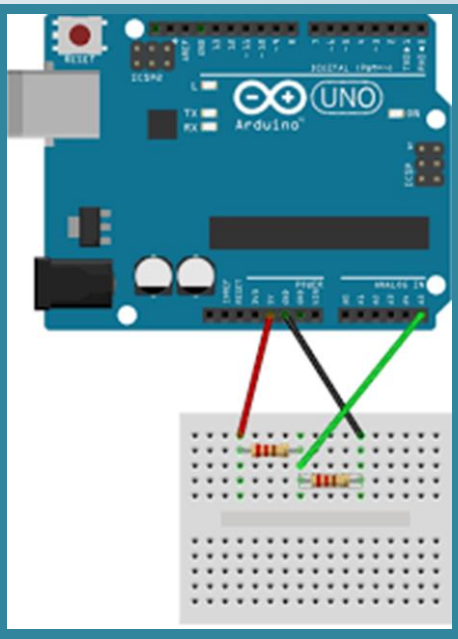
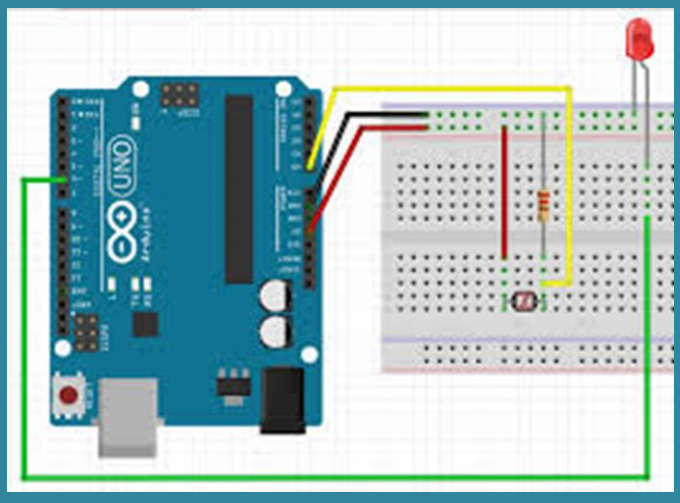
# LENGUAJE DE PROGRAMACIÓN DE ARDUINO

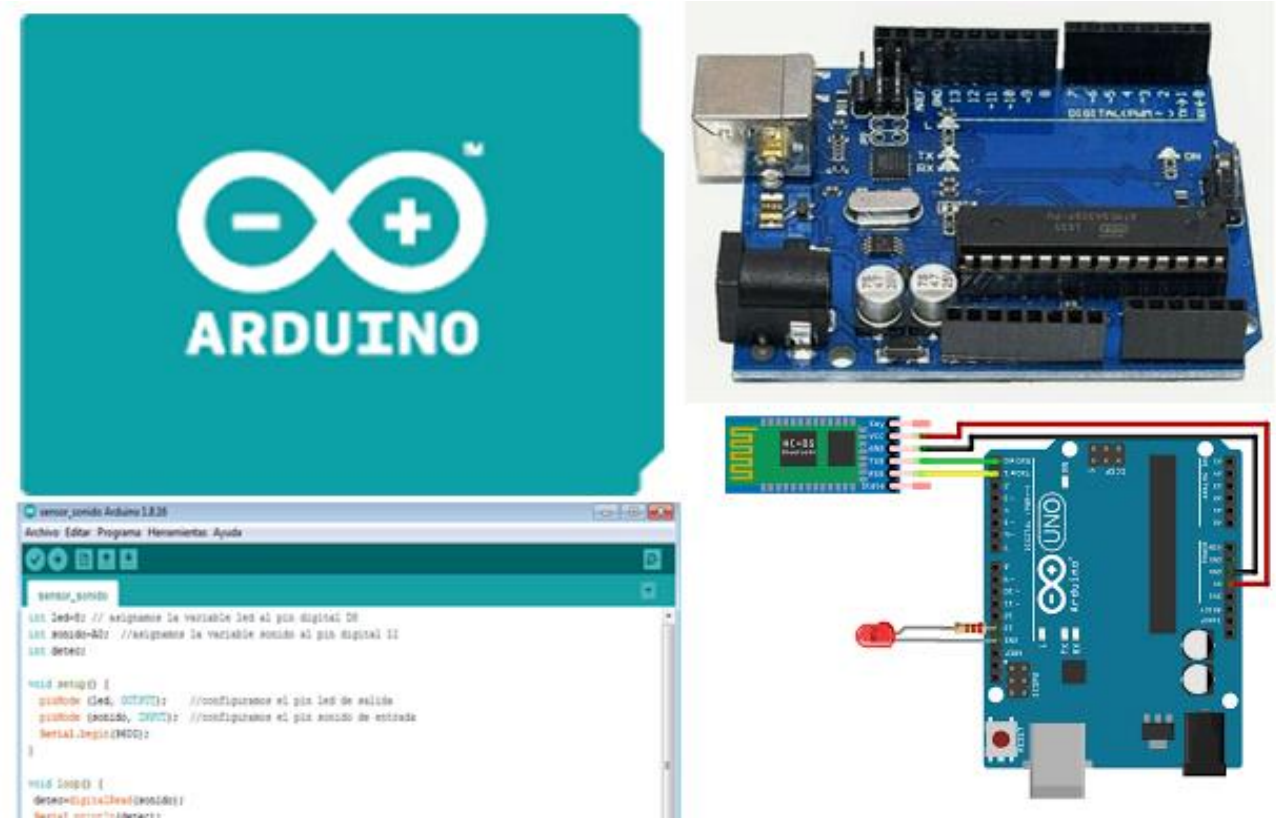


José M. Castillo Castillo



Versión 1.03. 2023

# Lenguaje de Programación de Arduino



# INDICE DE CONTENIDO

1. INTRODUCCIÓN_____(4)
2. ESTRUCTURA DE CÓDIGO Y SINTAXIS_____(6)
3. TIPOS DE DATOS Y VARIABLES_____(16)
4. CONSTANTES _____(26)
5. FUNCIONES_____(28)
6. ARITMÉTICA Y LÓGICA _____(31)
7. INSTRUCCIONES DE CONTROL_____(37)
8. ASIGNACIÓN DE ENTRADAS Y SALIDAS_____(46)
9. SEÑALES PWM EN ARDUINO_____(53)
10. GESTIÓN DEL TIEMPO _____(57)
11. FUNCIONES DE SONIDOS_____(59)
12. INTERRUPCIONES HARDWARE_____(63)
13. ALEATORIEDAD _____(67)
14. USANDO LAS BIBLIOTECAS_____(69)
15. VISUALIZACION DE VARIABLES POR EL MONITOR SERIE _____(74)
16. PROGRAMACIONES Y CIRCUITOS ELÉCTRICOS_____(88)

# 1. INTRODUCCIÓN

Arduino emplea un lenguaje de programación especial, donde podemos decir que se basa en la sintaxis de otros lenguajes de programación, como pueden ser C y C++. Aún y así, este lenguaje que utiliza Arduino, posee sus características propias, orientado a una fácil programación de los sensores y dispositivos externos. Para ello es importante conocer que la programación es la forma que tenemos de transmitir a un microcontrolador aquello que deseamos que haga para nosotros.

El microcontrolador es, por decirlo suavemente, estúpido, incapaz de pensar e improvisar, y solo capaz de seguir instrucciones precisas. La comunicación con el microcontrolador se realiza por escrito en un lenguaje de programación, el lenguaje es muy claro, estricto y tiene una sintaxis propia y unos estándares de diseño. Y si un error de sintaxis conduce a un error en la compilación del código o al funcionamiento incorrecto del dispositivo que ha flasheado, entonces el diseño del código no sirve para la conveniencia del programador, así como para aquellos que intentarán lidiar con su código.

Estos programas serán líneas escritas en un lenguaje del tipo ensamblador o lenguaje especial que sólo entiende el microcontrolador y la persona que lo está escribiendo. A este idioma se le denomina **lenguaje de programación.**

Hay muchos y variados tipos de lenguajes de programación, y cada uno de ellos tiene una característica que lo convierte en la mejor opción, según para lo que se desee programar.

Existen dos tipos de lenguaje de programación:

- **Lenguaje de bajo nivel**
- **Lenguaje de alto nivel.**

El **lenguaje de bajo** nivel son aquellos lenguajes de programación que están muy cerca del verdadero lenguaje que utilizan los ordenadores, es decir, el código binario 1 y 0.

Un lenguaje de bajo nivel no significa que sea un lenguaje fácil o de poca importancia, al contrario, son bastante difíciles de trabajar, e interactúan de forma directa con el microcontrolador. Un ejemplo de un lenguaje de bajo nivel es el **ASSEMBLER**.

El **lenguaje de alto** nivel son aquellos lenguajes de programación que son más parecidos a los idiomas que utilizan las personas. Recurren a instrucciones con igual significado y escritura que las palabras de los idiomas que hablamos, ejemplo: **while**, **input**, **output**, **do**, etc.

El **código máquina** binario se carga directamente en el propio microcontrolador, que parece un conjunto caótico de letras y números. Este código se puede obtener de cualquier lenguaje de programación, todo depende del entorno de desarrollo y de un intérprete.

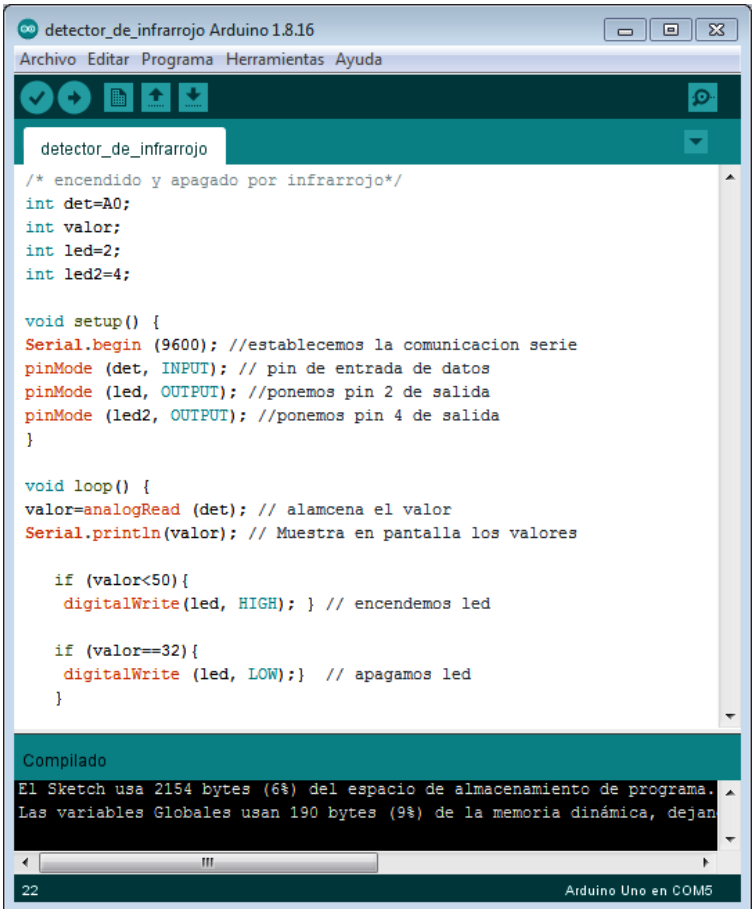
En la programación de Arduino, el entorno de desarrollo oficial es **Arduino IDE**, donde la programación se realiza en C++, uno de los lenguajes más populares y poderosos. Los propios desarrolladores llaman al lenguaje Arduino **Wiring**, ya que la biblioteca estándar **arduino.h** usa funciones y herramientas del marco Wiring. Pero el lenguaje, el lenguaje del cual se toma la sintaxis, es C++, por lo tanto, en paralelo con el estudio de funciones estándar, es recomendable estudiar cualquier libro de referencia sobre las programación C++. En él puedes encontrar mucha más información sobre el idioma que en todas las lecciones de Arduino combinadas (estamos hablando del idioma y la sintaxis, y no de las funciones de Wiring). Además de C, existen entornos de desarrollo que le permiten escribir en Java, por ejemplo Espruino WEB IDE, o B4R– en Basic. O XOD: tendrás que programar con bloques visuales. Pero francamente hablando, tal para mí, solo consideraremos C.

El lenguaje de programación de Arduino se puede dividir en tres grandes partes: **estructura**, **valores** (variables y constantes), y **funciones**.

## 2. ESTRUCTURA DE CÓDIGO Y SINTAXIS

Como se ha comentado anteriormente. Arduino emplea para su programación un lenguaje en C y C++, a través de su interfaz de Arduino, pero con una estructura y unos comandos especiales que facilitan mucho la programación e interacción con los diferentes sensores, actuadores y dispositivos que al él se conectan.

El entorno de desarrollo integrado **IDE** de Arduino es el área de edición del código donde se escribirá el código del software que se requiere. Aquí se crea el **sketch,** el programa escrito en código fuente. En éste área de trabajo es sencilla visualmente y tiene ciertas características que hace diferenciar las instrucciones cuando se está escribiendo. Esto es que las instrucciones aparecen con diferentes colores del texto: variables, funciones, entradas y salidas, controles de flujo, etc., siempre que la instrucción, código o función estén correctamente escrita, de lo contrario si se encuentra apagada, sin color, producirá error de compilación.




Una sección importante es el área de mensajes del **IDE** de Arduino, en donde el compilador le informa de posibles errores en el código. Además, en esta área, la placa Arduino puede enviar información acerca de su estado, según como sea programado.

La estructura de programa en Arduino puede llegar a ser muy diferente en función de la complejidad de la aplicación que queramos crear, pero como en la mayoría de lenguajes de programación ésta estructura está formada por **funciones**, **sentencias**, **bucles** y otros elementos que conforman la estructura del programa.

La estructura básica del lenguaje de programación de Arduino es bastante simple y está formada de **tres partes** principales. Estas tres partes son necesarias y encierran bloques que contienen declaraciones, configuraciones e instrucciones.

La **primera parte** del programa, se deben poner las bibliotecas específicas y la asignación de variables y constantes que se utilizarán en el programa. Este bloque se ejecuta una sóla vez y bajo cualquiera de los siguientes eventos:

- Encendido de la placa o alimentación del microcontrolador
- Después de un reset
- Después de cargar un programa desde el PC.

La **segunda parte** es la encargada de recoger la configuración del hardware de todos los dispositivos que se van a utilizar, todas las declaraciones se hacen en este bloque, y se ejecuta después de la sección anterior y por una sola vez, para ello se declara e introduce los datos al iniciar el programa, esta función es **void setup()**.

La **tercera parte** es la que está encargada de contener el programa con las instrucciones que se ejecutará indefinidamente, esta función es **void loop()** (de ahí el termino **loop – bucle**). Los comandos reales a realizar y el proceso de cálculo están codificados dentro de esta función. El **loop** significa que el código del programa fuente “sketch” se ejecutará infinitamente.

```
variables;      //asignación de variables

void  setup()   //bloque declaración de las variables y configuracion
{
instrucciones;
}
void  loop()    //bucle de la programación
{
instrucciones;

}
```

Primera parte:

```
/* encendido y apagado por mando a distancia de infrarrojo*/
int valor;       //asignamos la variable
int valor2;      //asignamos la variable 2
int rea=4;       // asignamos el puerto digital 4 a rea
int sensor=A0;   //asignamos el puerto analógico a sensor
int led=2;       //asignamos el puerto digital 2 a led
```

Segunda parte:

```
void setup() {
pinMode (sensor, INPUT); // configuramos sensor de entrada
pinMode (led, OUTPUT);   //configuramos led de salida
pinMode (rea, INPUT);    //configuramos el pin rea como entrada
}
```

Tercera parte:

```
void loop() {
valor=analogRead(sensor);   //alamcena el valor del sensor analogico en valor
valor2=digitalRead(rea);        // alamcena el valor de rea digital en valor2
if (valor<100 && valor2==LOW) { //condicionante  si valor menor de 100 y valor2
bajo
digitalWrite(led, HIGH);        //encendemos led
delay(500);                     //espera medio segundo
  }
if (valor<100 && valor2==HIGH) { //condicionante si valor menor de 100 y
valor2 alto
digitalWrite (led, LOW);        // apagamos led
delay(500);                     //espera medio segundo
}
}
```

Aunque no se utilicen o estén vacias, ambas funciones **void setup()** y **void loop()** deben estar presente en el sketch para que el programa funcione. Es recomendable tener en cuenta lo siguiente para evitar problemas de compilación:

- Se debe llamar a una variable de cualquier tipo después de su declaración. De lo contrario, habrá un error.
- La declaración y el uso de clases o tipos de datos de una biblioteca / archivo deben realizarse después de incluir la biblioteca / archivo.
- La función se puede llamar tanto antes como después de la declaración, porque C++ es un lenguaje compilado, la compilación se lleva a cabo en varias etapas y las funciones se «asignan» por separado, por lo que se pueden llamar en cualquier parte del programa.

## Sintaxis

En la programación de Arduino se necesita utilizar unas series de elementos sintácticos fundamentales para conseguir que el programa funcione correctamente y no dé errores en la compilación.

Los cuerpos de funciones se encierran entre llaves { }. Las llaves sirven para definir el principio y el final de un bloque de instrucciones. Se utilizan para los bloques de programación setup(), loop(), if.., etc.

Una llave de apertura "{" siempre debe ir seguida de una llave de cierre "}", si no es así el programa dará errores al compilarlo.

```
type  funcion() {  //bloque de la función
instrucciones;     // instrucciones que realiza el bloque funcion
}
```

**Nota:** El entorno de programación de Arduino incluye una herramienta de gran utilidad para comprobar el total de llaves. Sólo tienes que hacer click en el punto de inserción de una llave abierta e inmediatamente se marca el correspondiente cierre de ese bloque (llave cerrada).

Cada comando termina con un punto y coma. **El punto y coma** ";" se utiliza para separar instrucciones en el lenguaje de programación de Arduino. También se utiliza para separar elementos en una instrucción de tipo "bucle for".

```
int x = 13;   //declara la variable  'x' como tipo entero de valor  13
```

**Nota:** Olvidarse de poner fin a una línea con un **punto y coma** se traducirá en un error de compilación. El texto de error puede ser obvio, y se referirá a la falta de una coma, o puede que no. Si se produce un error raro y de difícil detección lo primero que debemos hacer es comprobar que los puntos y comas están colocados al final de las instrucciones.

Los **bloques de comentarios**, o multilíneas de comentarios, son áreas de texto ignorados por el programa que se utilizan para las descripciones del código o comentarios que ayudan a comprender el programa. Comienzan con /* y terminan con */ y pueden abarcar varias líneas.

Sólo se puede emplear al principio del programa. Este método se utiliza como cabecera, en la que el autor puede escribir lo que desee, es decir, su autoría, el título del programa, dedicatorias, etc.

```
/*Interruptor activado por detector de infrarrojo.
Por JM Fecha 02/11/23 */
int A=0;
void setup() {
pinMode (13, OUTPUT);
}
```

Debido a que los comentarios son ignorados por el programa y no ocupan espacio en la memoria de Arduino pueden ser utilizados con generosidad y también pueden utilizarse para "comentar" bloques de código con el propósito de anotar informaciones para depuración.

**Nota:** Dentro de una misma línea de un bloque de comentarios no se puede escribir otro bloque de comentarios (**usando /* ….. */** )

**Los paréntesis (…)** es otro de los símbolos que se suelen utilizar bastante en las funciones para incorporar datos.

Ejemplo:

```
pinMode(A0,INPUT);          //ponemos el pin analógico A0 de entrada
variableEntrada= analogRead(A0); //la variable recoge el valor analógico
del pin A0 leido
delay(1000);              // Espera 1 segundo
```

Una llamada a una función o método siempre termina entre paréntesis, incluso si la función no toma parámetros. Ejemplo: **loop().**

El método se aplica al objeto a través de **un punto**. Ejemplo: **Serial.begin();** y el separador decimal es un punto. Ejemplo: **0.25**

Los argumentos de las funciones y métodos, así como los miembros de una matriz, se enumeran con **comas**. Ejemplo: digitalWrite(3, HIGH);  matriz – int myArray[] = {3, 4, 5, 6};

Además, la coma es un operador independiente.

La **línea de comentarios** empieza con **"/ /"** y terminan con la siguiente línea de código. Al igual que los comentarios de bloque, los de línea son ignoradas por el programa y no ocupan espacio en la memoria.

```
pinMode(13,OUTPUT);          //ponemos el pin 13 de salida
```

Una línea de comentario se utiliza a menudo después de una instrucción, para proporcionar más información acerca de lo que hace esta o para recordarla más adelante.

Cuando se está programando es muy importante introducir comentarios sobre qué funciones desempeñan cada una de las líneas que conforman el código. Esto es muy útil cuando los códigos son visionados o revisados por otras personas diferentes del programador, ahorrando tiempo y esfuerzo en la compresión del código.

Las cadenas y las matrices de caracteres se incluyen entre **comillas dobles ("…")**. Se utilizan para visualizar un texto, que se encuentra dentro de las comillas, en la ventana del monitor serie. Por ejemplo:

```
Serial.println("¡ATENCION ZONA ACTIVADA!"); //visualiza por el monitor
serie  el mensaje de texto
delay(500);                                    // se detiene 0,5 segundos
}
else {                                         //de lo contrario
Serial.println("TODO ESTA CORRECTO");        //visualiza por monitor serie
que  el mensaje de texto
delay(500);  //se detiene 0,5 segundos
}
```

El texto entre comillas en el **Serial.println("Texto")** es lo que aparece por pantalla.

**Las comillas simples ('…')** es otro símbolo utilizado en la programación de Arduino. Los caracteres aislados se encierran entre comillas simples 'a'.

En el caso del ejemplo siguiente se utiliza para configurar unas letras para una operación de envío y recepción en el monitor serie, mediante las instrucciones **Serial.available()** y **Serial.read()**.

Ejemplo:

```
void loop() {
if (Serial.available()) {  // comprobamos que hay algo en la barra de texto del
monitor
tecla=Serial.read(); //lee lo que hay en el monitor serie y lo guarda en la
variable tecla
if (tecla=='a') {                // si la tecla pulsada es una a…
Serial.println (tecla);         // se imprime el carácter en el monitor serie
digitalWrite (pin, HIGH);       // enciende el led
}
if (tecla=='z') {               // si el carácter introducido es una z…
Serial.println(tecla);          // se visualiza por el monitor serie la letra z
digitalWrite(pin, LOW);         //apaga el led
}
}
}
```

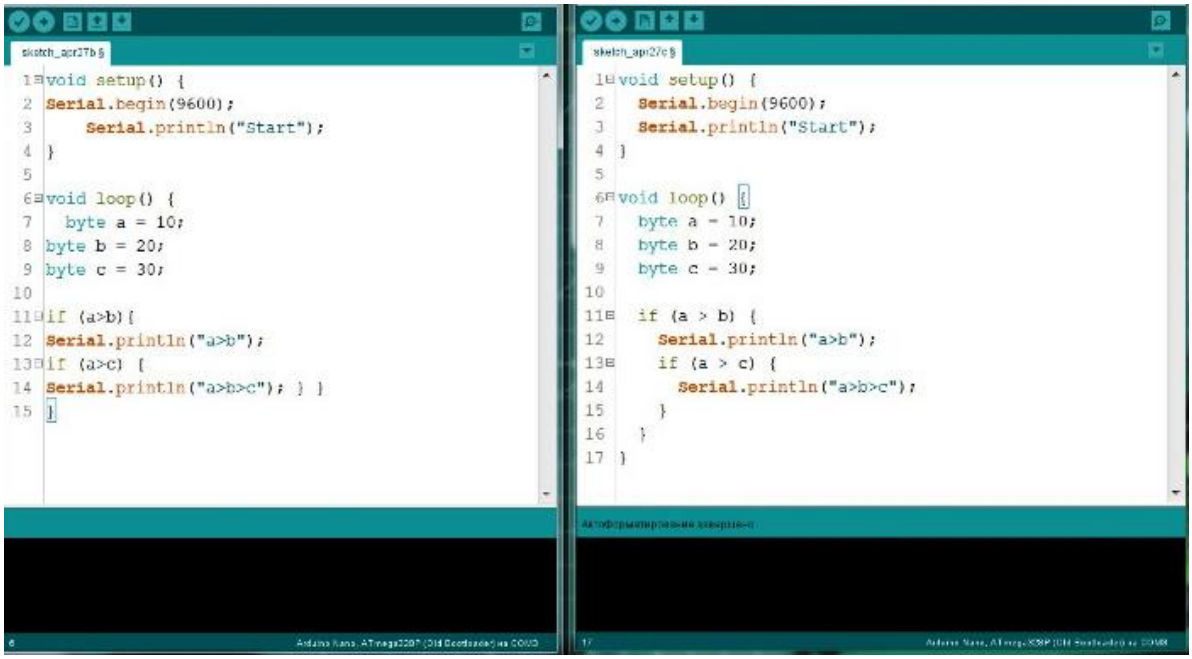
Los nombres de las variables pueden contener letras latinas mayúsculas y minúsculas, números y guiones bajos. Ejemplo: myVal_35. Los nombres de las variables no pueden comenzar con un dígito. Solo con letra o subrayado

Es case sensitive, es decir la mayúscula es diferente a la minúscula. Ejemplo: las variables val y Val No son lo mismo.

**Nota:** Es importante prestar mucha atención cuando se está escribiendo el código de programación, si se olvida colocar un símbolo, o colocarlo donde no es, o el código mal escrito, esto dará error en la compilación.

## Formateo

Existe el formateo (alineación) del código, es decir, respetar los espacios y el espaciado. Por ejemplo, compare estos dos códigos. ¿Cuál se ve más claro y visual?



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

void loop() {
  byte a = 10;
byte b = 20;
byte c = 30;

if (a>b){
Serial.println("a>b");
if (a>c) {
Serial.println("a>b>c"); } }
}
```

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

void loop() {
  byte a = 10;
  byte b = 20;
  byte c = 30;

  if (a > b) {
    Serial.println("a>b");
    if (a > c) {
      Serial.println("a>b>c");
    }
  }
}
```

Todos los IDE serios tienen formateo de código automático, funciona tanto en el proceso de escritura como de guardia. Arduino IDE – no es una excepción, tiene el formateo de código y combinaciones de teclas calientes **Ctrl + T.** Ten en cuenta que:

- Hay un espacio entre las operaciones matemáticas, los signos de comparación, la asignación y todo eso.
- Como en el texto ordinario, se coloca un espacio después y no antes de una coma, dos puntos y un punto y coma.

- La sangría del borde izquierdo de la pantalla es un carácter de tabulación, el código se desplaza hacia la derecha y los comandos se forman a partir de un bloque de código a la misma distancia. En Arduino IDE, una tabulación equivale a dos espacios.
- Cada acción, sentencia, se realiza en una nueva línea (el formato automático no soluciona esto).
- Es habitual escribir llaves al principio y al final de un bloque de código en una línea separada. Además, muchas personas escriben un paréntesis abierto en una línea con un operador, esto ahorra espacio.

## Nombres de variables

- Es costumbre escribir los nombres de las variables comenzando con una letra minúscula, para nombrarlas de manera que quede claro. Ejemplo: value.
- Si un nombre de variable consta de dos o más palabras, están separadas por la primera letra mayúscula de cada nueva palabra, o las palabras están separadas por guiones bajos. Ejemplo: myButtonState, button_flag.
- Es costumbre escribir los nombres de los tipos y clases de datos con mayúscula inicial. Ejemplo: Signal, Servo.
- Es costumbre escribir los nombres de las constantes en mayúsculas, la separación es el subrayado. Ejemplo: MOTOR_SPEED.
- Al escribir bibliotecas y clases, se acostumbra escribir los nombres de las variables internas comenzando con el carácter de subrayado. Ejemplo: _position.
- Hay varias abreviaturas comunes para los nombres de variables, a menudo las verá en el firmware y las bibliotecas de otras personas:
    - button – btn, botón
    - índex – idx – i, índice
    - búffer – buf, búfer
    - value – val, valor
    - variable – var, variable
    - pointer – ptr, puntero
- Es habitual comenzar los nombres de funciones y métodos con un verbo que describa brevemente la acción de la función. Estos son los que verá todo el tiempo:
    - get – obtener el valor (getValue)
    - set – establecer valor (setTime)
    - print, show – mostrar algo

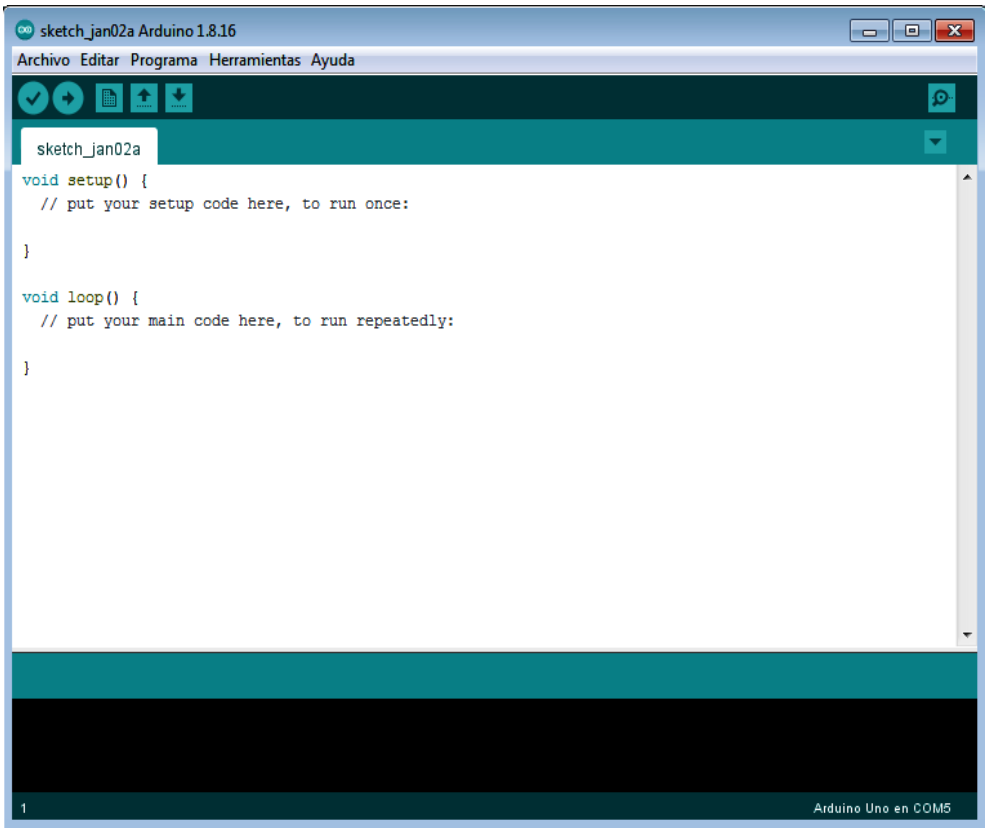
- read– leer
- write – escribir
- change – cambiar
- clear – borrar
- begin, start – empezar
- end, stop – terminar, detener

**Nota:** El peso del firmware cargado en el microcontrolador no se ve afectado, porque el código se convierte en código máquina y no hay nombres, solo direcciones.

## Estructura de código

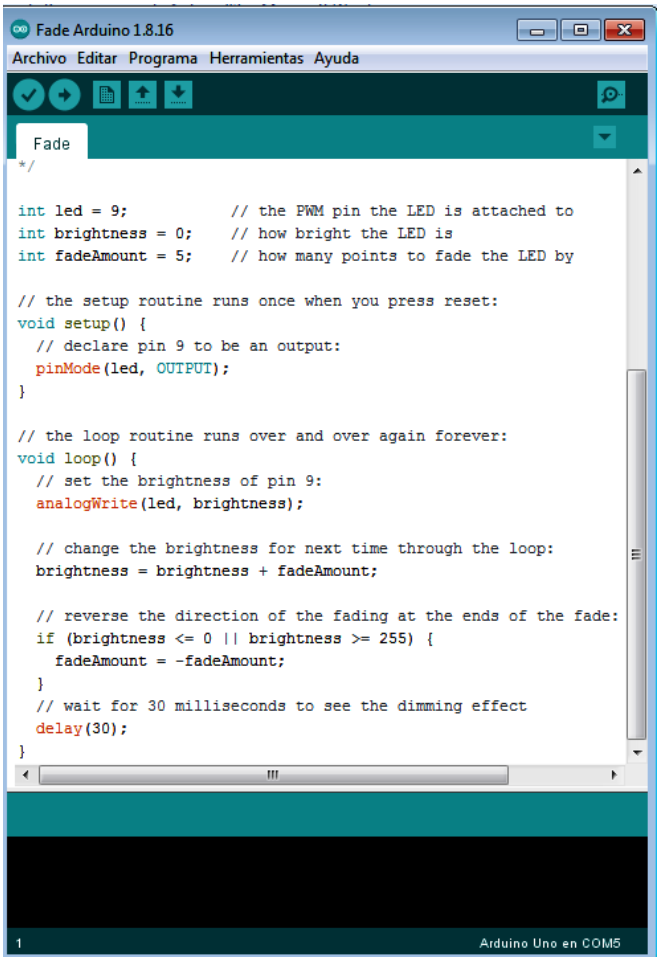
Al iniciar el IDE de Arduino, nos da un espacio en blanco en forma de dos funciones requeridas: **setup()** y **loop().**

El código en bloque **setup()** se ejecuta una vez cada vez que se inicia el microcontrolador. El código en el bloque **loop()** se realiza «en círculo» durante todo el funcionamiento del microcontrolador, a partir del momento de finalización de setup().



A modo práctico se muestra a continuación la estructura de boceto que se puede utilizar para facilitar el trabajo en la programación de Arduino:

1. Descripción del firmware, enlaces útiles, notas, autoría.
2. Constantes de ajuste (definidas y regulares).
3. Constantes de servicio (que solo deben cambiarse con pleno conocimiento del tema).
4. Bibliotecas conectadas y archivos externos, declaración de los tipos y clases de datos correspondientes.
5. Variables globales.
6. setup().
7. loop().
8. Las funciones.



## 3. TIPOS DE DATOS Y VARIABLES

Existen en el microcontrolador de Arduino tres tipos de memorias que utilizamos para nuestra programación:

- **Flash o ROM:** memoria de solo lectura, memoria no volátil del microcontrolador, que almacena el código del programa, firmware, funciones, procedimientos, operaciones, todo esto se queda ahí y no cambia durante la operación del código (puede, por supuesto, ingresar allí, pero no lo haremos, y no lo necesitamos). Durante la descarga del firmware, el cargador de arranque escribe el firmware aquí.
- **SRAM o RAM:** memoria de acceso aleatorio. Almacena variables cuyos valores pueden cambiar durante la operación, tenemos más que libre acceso a esta memoria y la usaremos activamente. Después de reiniciar el microcontrolador, esta memoria se borra por completo y toma valores erraticos.
- **EEPROM:** Memoria de solo lectura: ROM programable borrable eléctricamente, memoria no volátil asignada al usuario para almacenar valores que se pueden cambiar, pero que no se restablecerán después de un reinicio. Para datos de configuración etc, hay una lección aparte al respecto.

En la programación del microcontrolador, la información se almacena, convierte y transmite en forma digital, es decir, en forma de **ceros** y **unos**. En consecuencia, una celda de memoria elemental solo puede recordar 0 o 1 y se llama **bit**.

| Binario | Decimal |
|---|---|
| 0000 | 0 |
| 0001 | 1 |
| 0010 | 2 |
| 0011 | 3 |
| 0100 | 4 |
| 0101 | 5 |
| 0110 | 6 |
| 0111 | 7 |
| 1000 | 8 |
| 1001 | 9 |
| …. | …. |
| 10000 | 16 |

En programación, el conteo comienza desde cero, es decir, con 5 bits podemos codificar un número decimal de 0 a 31, con 8 bits – de 0 a 255, 10 bits – de 0 a 1023. Es muy importante entender y recordarlo:

- 5 bits – 32
- 6 bits – 64
- 7 bits – 128
- 8 bits – 256
- 9 bits – 512
- 10 bits – 1024

La siguiente unidad de medida más grande en el mundo digital es un **byte**, que consta de 8 bits. Debido a que los buses de los primeros microprocesadores tenían un ancho de 8 bits, por lo que probablemente este número se tomó como una unidad de memoria más antigua. Además, 8 es 2 elevado a 3, lo cual es muy simbólico y adecuado. Y también, para codificar todas las letras latinas, signos de puntuación, signos matemáticos y solo símbolos (todo en el teclado), antes eran suficientes 7 bits (128 caracteres), pero luego se volvieron pocos, y se introdujo un bit adicional, el octavo. Es decir, 8 bits también es el tamaño de la tabla de caracteres, que se llama ASCII. Pero estas son excepciones de los procesadores antiguos, en los dispositivos digitales modernos, un byte siempre contiene 8 bits ( en diferentes arquitecturas AVR pueden ser diferentes), lo que le permite codificar 256 números decimales del 0 al 255, respectivamente.

- B = Byte
- b = bit
- K = Kilo = 1.000
- M = Mega = 1.000.000
- 1 KB = 1024 B
- 1 MB = 1024 kB
- 1 GB = 1024 MB
- Etc.

Los datos en la memoria del microcontrolador se almacenan en representación binaria, pero además, existen otros sistemas de cálculo en los que podemos trabajar. Trate de recordar de inmediato y comprender que no necesita traducir números de un sistema numérico a otro, a Arduino no le importa en absoluto en qué formato alimenta el valor de una variable, se interpretarán automáticamente en forma binaria. Se introducen diferentes sistemas de numeración principalmente para la conocimiento del programador.

Ahora, en esencia: arduino admite (y en general no necesita nada más) cuatro sistemas numéricos clásicos: **binario**, **octal**, **decimal** y **hexadecimal**. Un pequeño recordatorio: el sistema hexadecimal tiene 16 valores por dígito, los primeros 10 son como decimales, el resto son las primeras letras del alfabeto latino: 0, 1, 2, 3, 4, 5, 6, 7, 8 , 9, a, b, c, d, e, f.

Con el sistema decimal, todo es simple, escribimos los números como se ven. 10 es diez, 25 es veinticinco y así sucesivamente. El binario tiene el prefijo 0b (cero b) o B, es decir, el número binario 101 se escribirá en el IDE como 0b101 o B101. Octal tiene el prefijo 0 (cero), por ejemplo 012. El hexadecimal tiene el prefijo 0x (cero x), FF19 se escribirá como 0xFF19.

| Base | Prefijo | Ejemplo | Características |
|---|---|---|---|
| 2 (binario) | B o 0b (cero b) | B1101001 | dígitos 0 y 1 |
| 8 (octal) | 0 (cero) | 0175 | dígitos 0 – 7 |
| 10 (decimal) | | 160503 | dígitos 0 – 9 |
| 16 (hexadecimal) | 0x (cero x) | 0xFF21A | números 0-9, letras A-F |

El sistema binario se usa generalmente para la presentación visual de datos y configuraciones de bajo nivel de varios registros hardware. Por ejemplo, la configuración está codificada con un byte, cada bit en ella es responsable de una configuración separada (encendido / apagado), y al transferir un byte de la forma 0b10110100, puede configurar inmediatamente un montón de cosas.

## Variables

Una variable es un pequeño conteneder de memoria que se ubica en la memoria SRAM y se emplea para almacenar datos, ya sean letras, números o una combinación de ambos. Tiene su propio nombre único y almacena números de acuerdo con su tamaño. Podemos referirnos a una variable por su nombre y obtener su valor o cambiarlo.

Una variable es una manera de nombrar y almacenar un valor alfanumérico para su uso posterior por el programa.

Como su nombre indica, las variables pueden alterar su contenido a lo largo del tiempo, es decir, el programador puede cambiar su valor de forma directa durante la escritura del programa o de forma indirecta mientras se ejecuta el programa. Una variable debe ser declarada y, opcionalmente, asignarle un valor.

Es posible tener dos o más variables del mismo nombre en diferentes partes del mismo programa que pueden contener valores diferentes. La garantía de que sólo una función tiene acceso a sus variables dentro del programa simplifica y reduce el potencial de errores de programación.

Las variables pueden ser declaradas en diferentes lugares del programa, en función del lugar donde sean declaradas, las variables van a ser de tipo **global** o **local**, determinando esto el ámbito de aplicación o la capacidad de ciertas partes del programa para hacer uso de las mismas.

Una **variable global** es aquella que puede ser vista y utilizada por cualquier función y estamento de un programa. Las variables globales se declaran fuera de las funciones y está disponible para leer y escribir en cualquier parte del programa, en cualquiera de sus funciones.

```
byte var;    //asignamos el valor de la variable
void setup() {
  // cambia la variable global
  var = 50;
}
void loop() {
  // cambia la variable global
  var = 70;
}
```

Una **variable local** es aquella que se define dentro de una función o como parte de un bucle. Sólo será visible y sólo podrá utilizarse dentro de la función o bucle donde es declarada.

```
void  setup() {            // no es necesario configurar
}
void  loop(){
for  (int  x=0;  x<20;){  //'x'  solo es visible  dentro del bucle for
x++;                      // x +1
}
float  f;                 // 'f'  es visible solo dentro del bucle
}
```

Por lo tanto, podremos tener tantas variables como deseemos, pero para ello se deberán "**declarar**" en el programa para que el procesador las tenga en cuenta.

El siguiente código de ejemplo declara una variable llamada **variableEntrada** y luego le asigna el valor obtenido en la entrada analógica del PIN2:

```
int  variableEntrada = 0; // declara una variable y le asigna el valor 0
variableEntrada = analogRead(2); //la variable recoge el valor analógico del pin2
```

**'variableEntrada'** es la variable en sí. La primera línea declara que será de tipo entero "**int**". La segunda línea fija a la variable el valor correspondiente a la entrada analógica **PIN2**. Esto hace que el valor de PIN2 sea accesible en otras partes del código.

Cuando se programa en Arduino podemos declarar las variables en cualquier lugar del mismo, es decir, podemos hacerlo antes del bloque "**void setup**", dentro del bloque o en el bloque "**void loop**". Cuando declaramos una variable, podemos hacerlo de dos formas: una dando un valor inicial a la variable: **int A=0;** y la otra asignándole un valor después de la declaración cuando se está ejecutando el programa **int A;**

Una vez que una variable ha sido asignada, o reasignada, se puede probar su valor para ver si cumple ciertas condiciones (instrucciones if..), o puede utilizar directamente su valor. Como ejemplo ilustrativo veamos tres operaciones útiles con variables: el siguiente código prueba si la variable "entradaVariable" es inferior a 100, si es cierto se asigna el valor 100 a "entradaVariable" y, a continuación, establece un retardo (**delay**) utilizando como valor "entradaVariable" que ahora será como mínimo de valor 100:

```
if (entradaVariable < 100) { // pregunta si la variable  es menor de  100
entradaVariable = 100;       //si es cierto asigna el valor 100
}
delay(entradaVariable);      //  usa  el valor como retardo
```

**Nota:** Las variables deben tomar nombres descriptivos, para hacer el código más legible. Nombres de variables pueden ser "contactoSensor" o "pulsador", para ayudar al programador y a cualquier otra persona a leer el código y entender lo que representa la variable. Nombres de variables como "var" o "valor", facilitan muy poco que el código sea inteligible. Una variable puede ser cualquier nombre o palabra que no sea una palabra reservada en el entorno de Arduino.

Es muy práctico asignar a las variables nombres que indiquen qué valor o valores van a almacenar en el programa, de esta forma, podemos saber a simple vista qué datos está almacenando la variable en el programa. Todas las variables tienen que declararse antes de que puedan ser utilizadas en el programa para que se tenga en cuenta.

**Nota:** si el nombre de una variable local es el mismo que uno global, entonces la prioridad de llamar por nombre en la función se le da a la variable local:

```
void setup() {
  // pasa 10 como argumento
  myFunc ( 10 ) ;
}
void loop() {
}
void myFunc ( byte var ) {
  // var es "local" aquí
  // sumar 20
  var + = 20;
  // después de ¡la variable se eliminará de la memoria!
}
```

## Tipos de variables

A la hora de usar una variable dentro del programa debemos de pararnos a pensar que información o qué tipo de operaciones vamos a aplicar a estas variables y en función de esto asignarle un tipo u otro. Debemos de asignarles que el tipo de variable que estemos usando tiene rango suficiente para almacenar los datos sin llegar a ser desproporcionado, ya que usar variables grandes va a consumir más recursos de memoria.

Para adecuar la variable con el fin de que contenga los diferentes tipos de datos, se debe declarar la variable y el tipo de dato que almacenará, por ejemplo:

```
int A=5;
//declaramos variable int (integro)  A con valor 5 y lo almacena en
memoria. Son  números enteros que van del rango de 32.767 a -32.768
boolean F=false;
//declaramos variable booleano F con valor Falso. Esta variable almacena
dos posibles valores: True (verdad) o False (falso)
char Texto=´Buenos días´;
//declaramos variable de caracteres Texto que almacenará “Buenos día”
```

En el primer caso, podemos ver como declaramos la variable **A**, del tipo entero (5), y se almacenará el número **5** en este caso.

En el segundo caso, podemos observar que declaramos la variable **F**, del tipo booleano, y almacenamos en ella el valor **falso**.

En el tercer caso, declaramos una variable llamada **Texto**, del tipo carácter y que almacena “**Buenos días**”.

Al declarar una variable se comienza por definir su tipo como **int** (entero), **long** (largo), **float** (coma flotante), etc, asignándoles siempre un nombre, y, opcionalmente, un valor inicial. Esto sólo debe hacerse una vez en un programa, pero el valor se puede cambiar en cualquier momento usando aritmética y reasignaciones diversas.

El siguiente ejemplo declara la variable **entradaVariable** como una variable de tipo entero "**int**", y asignándole un valor inicial igual a cero. Esto se llama una **asignación**.

```
int entradaVariable = 0;  //declaramos variable entradaVariable a 0
```

Una variable puede ser declarada en una serie de lugares del programa y en función del lugar en donde se lleve a cabo la definición esto determinará en que partes del programa se podrá hacer uso de ella.

En la programación de Arduino podemos utilizar diferentes tipos de variables dependiendo de los datos que se van a almacenar en la programación, estos son:

| Nombre | alias | Peso | Rango | Característica |
|---|---|---|---|---|
| **boolean** | bool | 1 byte | 0 o 1.<br>**verdadero** o **falso** | Variable booleana. bool en Arduino también toma 1 byte. |
| **char** | int8_t | 1 byte | 0 – 255 | Almacena el número de carácter de la tabla de caracteres ASCII. Tipo entero |
| **byte** | uint8_t | 1 byte | -128 – 127 | Tipo entero. |
| **int** | int16_t, short | 2 bytes | -32 768 ⋯ 32 767 | Tipo entero. |
| **unsigned int** | uint16_t, word | 2 bytes | 0 ⋯ 65 535 | Tipo entero |
| **long** | int32_t | 4 bytes | -2147 483 648 ⋯ 2147483647 | Tipo entero |
| **unsigned long** | uint32_t | 4 bytes | 0 ⋯ 4 294967 295 | Tipo entero |
| **float** | – | 4 bytes | -3.4028235E + 38 ⋯ 3.4028235E + 38 | Almacena números de coma flotante (decimales). Precisión: 6-7 dígitos |
| **double** | – | 4 bytes | | Para AVR es lo mismo que **float** |
| – | int64_t | 8 | – (2 ^ 64) / 2 ⋯ (2 ^ 64) | * Números muy grandes. La serie |

| | | | | |
|---|---|---|---|---|
| | | bytes | / 2-1 | estándar no puede generarlos |
| – | uint64_t | 8 bytes | 2 ^ 64-1 | * Números muy grandes. La serie estándar no puede generarlos |

A continuación veremos algunos ejemplos de estos tipos de variables:

### byte

**byte** almacena un valor numérico de 8 bits sin decimales. Tienen un rango entre 0 y 255.

```
byte unaVariable = 180;   // declara 'unaVariable' como tipo byte
```

### int

**int** es un tipo de datos primarios que almacena valores numéricos enteros de 16 bits sin decimales comprendidos entre el rango 32.767 a -32.768.

```
int unaVariable = 1500; // declara 'unaVariable' como una variable de
tipo entero
```

**Nota:** Las variables de tipo entero "**int**" pueden sobrepasar su valor máximo o mínimo como consecuencia de una operación. Por ejemplo, si x = 32767 y una posterior declaración agrega 1 a x, x = x + 1 entonces el valor de x pasará a ser -32.768. (algo así como que el valor da la vuelta).

### long

El formato de variable numérica de tipo extendido "**long**" se refiere a números enteros (tipo 32 bits) sin decimales que se encuentran dentro del rango =2147483648 a 2147483647.

```
long unaVariable = 90000;  //  declara 'unaVariable' como tipo long
```

### float

El formato de dato del tipo "**punto flotante**" "**float**" se aplica a los números con decimales. Los números de punto flotante tienen una mayor resolución que los de 32 bits con un rango comprendido 3.4028235E +38 a +38=3.4028235E.

```
float unaVariable = 3.14;  //declara  'unaVariable' como tipo flotante
```

**Nota:** Los números de punto flotante no son exactos, y pueden producir resultados extraños en las comparaciones. Los cálculos matemáticos de punto flotante son también mucho más lentos que los del tipo de números enteros, por lo que debe evitarse su uso si es posible.

## Unsigned long

**Unsigned long** es una variable que como su nombre indica, **unsigned**, prepara a la variable para que contenga un número **sin signo** y por otro lado el tipo **long** indica que la variable será de una longitud de 32 bits o lo que es lo mismo de 4 bytes.

Una variable tipo **long** puede almacenar números que van de -21474836648 a 2147483647.

```
/* Ejemplo variable unsigned long*/
unsigned long tiempo; //variable unsigned long asignada a tiempo
void setup() {
}
void loop() {
tiempo=millis();  //función se activa y se guarda en la variable tiempo
}
```

## arrays

Un **array** es un conjunto de valores a los que se accede con un número índice. Cualquier valor puede ser recogido haciendo uso del nombre de la matriz y el número del índice. El primer valor de la matriz es el que está indicado con el índice 0, es decir el primer valor del conjunto es el de la posición 0. Un array tiene que ser declarado y opcionalmente asignados valores a cada posición antes de ser utilizado.

```
int  miArray[] = {valor0,  valor1,  valor2...}
```

Del mismo modo es posible declarar una matriz indicando el tipo de datos y el tamaño y posteriormente, asignar valores a una posición especifica:

```
int  miArray[5];     //  declara un array de enteros de  6  posiciones
miArray[3] = 10;     //  asigna l valor 10 a la posición 4
```

Para leer de un **array** basta con escribir el nombre y la posición a leer:

```
x = miArray[3];  //x ahora es igual a  10 que está en la posición 3 del
array
```

Las matrices se utilizan a menudo para estamentos de tipo bucle, en los que la variable de incremento del contador del bucle se utiliza como índice o puntero del array. El siguiente ejemplo usa una matriz para el parpadeo de un LED.

En el siguiente ejemplo utilizamos un bucle tipo **for**, el contador comienza en cero 0 y escribe el valor que figura en la posición de índice 0 en la serie que hemos escrito dentro del **array parpadeo**[], en este caso 180, que se envía a la salida analógica tipo **PWM** configurada en el pin10, se hace una pausa de 200 ms y a continuación se pasa al siguiente valor que asigna el índice "i".

```
int  ledPin = 10;  //  Salida LED  en el PIN 10
byte  parpadeo[] = {180, 30, 255, 200, 10,  90, 150, 60};  //  array de 8
valores diferentes

void  setup() {
pinMode(ledPin, OUTPUT);  //configura la salida pin10
}
void  loop(){                //  bucle del programa
```

## 4. CONSTANTES

Lo que es una constante se desprende claramente de su nombre: algo, cuyo valor solo podemos leer y no podemos cambiar. Hay dos formas de establecer (declarar) una constante: Al igual que una variable, antes del tipo de datos con la palabra **const**. Si el valor de la variable no cambia durante la ejecución del programa, se recomienda declararlo como constante, esto permitirá al compilador optimizar mejor el código y en la mayoría de los casos será un poco más fácil y rápido.

```
const byte myConst = 10; // declara una constante de tipo byte
```

Usando una directiva de preprocesador #define, que hace lo siguiente: en la etapa de compilación del código, el preprocesador reemplaza todas las secuencias de caracteres especificadas en el documento actual (recuerde que las pestañas de Arduino IDE son un documento) con sus valores correspondientes. Constante definida con #define no ocupa espacio en la RAM, pero se almacena como código de programa en la memoria Flash, esta es la mayor ventaja de este método. Sintaxis: #define 'el valor del nombre' La coma no se usa. De esta manera, generalmente se indican los pines de conexión, los ajustes, varios valores, etc. Ejemplo:

```
#define BTN_PIN 10
#define DEFAULT_VALUE 3423
```

Algunas palabras más sobre constantes y variables: si una variable ordinaria no cambia en ningún lugar durante la ejecución del programa, el compilador puede convertirla en una constante por sí sola y no ocupará espacio en la RAM, es decir, se colocará en Flash.

Las constantes se utilizan para hacer los programas más fáciles de leer y se clasifican en grupos.

A diferencia de las variables, que pueden cambiar a cada momento según se determine en el programa, un dato constante no cambiará nunca de valor.

Arduino incorpora en su programación unas constantes que se utilizan para determinar ciertos valores, en sensores o componentes electrónicos como los diodos led, para determinar si una expresión es cierta o falsa, y para establecer si el pin al que se conecta un sensor es de entrada o de salida.

## TRUE / FALSE (cierto/falso)

Estas son constantes booleanas que definen los niveles **HIGH** (alto) y **LOW** (bajo) cuando estos se refieren al estado de las salidas digitales. **FALSE** se asocia con 0 (cero), mientras que **TRUE** se asocia con 1, pero **TRUE** también puede ser cualquier otra cosa excepto cero. Por lo tanto, en sentido booleano, =1, 2 y = 200 son todos también se define como TRUE. (esto es importante tenerlo en cuanta).

```
if  (b == TRUE) {

ejecutar las instrucciones; }
```

## HIGH / LOW (alto/bajo)

Estas constantes definen los niveles de salida altos o bajos y se utilizan para la lectura o la escritura digital para los pines del microcontrolador en el bloque de **void loop(). HIGH** se define como en la lógica de nivel 1 ó 5 voltios, mientras que **LOW** es lógica nivel 0 o 0 voltios.

```
digitalWrite(13, HIGH);   // activa la salida 13 con un nivel alto (5v.)
digitalWrite(12, LOW);    //desactiva el pin 12 con un nivel bajo (0V)
```

## INPUT/OUTPUT (entrada/salida)

Estas constantes son utilizadas para definir, al comienzo del programa en el **void setup()**, el modo de funcionamiento de los pines mediante la instrucción **pinMode** de tal manera que el pin puede ser una entrada **INPUT** o una salida **OUTPUT**.

```
pinMode(13, OUTPUT);     // designamos que el PIN 13 sea de salida
pinMode(10, INPUT);      // designamos que el pin 10 sea de entrada
```

## 5. FUNCIONES

Una función es parte de un programa que tiene su propio nombre y consta de un conjunto de comandos que son ejecutados cuando se llama a la función realizando una determinada tarea en Arduino.

Un programa grande se puede construir a partir de varias funciones, cada una de las cuales realiza su propia tarea. El uso de funciones simplifica enormemente la escritura y lectura de código y, en la mayoría de los casos, lo hace óptimo en términos de la cantidad de memoria ocupada.

Hasta ahora se han estado analizando funciones predefinidas para la programación de la placa Arduino éstas funciones son **void setup()** y **void loop()** de las que ya se ha comentado anteriormente.

La función debe describirse (declararse) y, a continuación, puede llamarse. La función debe describirse fuera de otras funciones. En general, la función tiene la siguiente estructura:

```
tipo de datos nombre_función (conjunto de parámetros) {
  < cuerpo_función >
}
```

Donde el tipo de datos es el tipo de datos que devuelve la función, el nombre de la función es el nombre con el que se llama a la función, el conjunto de parámetros es un conjunto opcional de variables que admite para trabajar y el cuerpo de la función es el conjunto real de operaciones. La función se llama por el nombre de la función y pasando el conjunto de parámetros, si los hay: si no admite ningún parámetro, debe especificar paréntesis vacíos de todos modos.

La función puede necesitar parámetros, puede no aceptarlos, puede devolver algún valor, puede no devolver nada. Echemos un vistazo a estas opciones.

La opción más fácil de entender, empecemos por ella, es void, que se traduce del inglés como «vacío». Al crear una función de tipo void, le decimos al compilador que no se devolverán valores (más precisamente, la función devolverá “nada”). Escribamos una función que encuentre la suma de dos números y la asigne al tercer número. Dado que la función no tiene parámetros y no devuelve nada, las variables deberán declararse previamente y hacerse globales, de lo contrario la función no tendrá acceso a ellas y recibiremos un error.

```
byte a, b;
int c;
void setup() {
  a = 10;
  b = 20;
  sumFunction();
  // después de llamar a la función
  // c es 30
}
void loop() {
}
void sumFunction() {
  c = a + b;
}
```

Las funciones de usuario pueden ser escritas para realizar tareas repetitivas y para reducir el tamaño de un programa. Las funciones se declaran asociadas a un tipo de valor "**type**". Este valor será el que devolverá la función, por ejemplo **'int'** se utilizará cuando la función devuelva un dato numérico de tipo entero. Si la función no devuelve ningún valor entonces se colocará delante la palabra "**void**", que significa "**función vacía**". Después de declarar el tipo de dato que devuelve la función se debe escribir el nombre de la función y entre paréntesis se escribirán, si es necesario, los parámetros que se deben pasar a la función para que se ejecute.

```
type  nombreFunción(parámetros)
{
instrucciones;
}
```

La función siguiente devuelve un número entero, **delayVal()** se utiliza para poner un valor de retraso en un programa que lee una variable analógica de un potenciómetro conectado a una entrada de Arduino. Al principio se declara como una variable local, ´**v**´ recoge el valor leído del potenciómetro que estará comprendido entre 0 y 1023, luego se divide el valor por 4 para ajustarlo a un margen comprendido entre 0 y 255, finalmente se devuelve el valor ´v´ y se retornaría al programa principal. Esta función cuando se ejecuta devuelve el valor de tipo entero ´**v**´

```
int delayVal() {
int v;                //  crea una variable temporal  'v'
v= analogRead(pot);    // lee el valor del potenciómetro
v  /=  4;              //  convierte  0-1023 a  0-255
return  v;             //  devuelve el valor final
}
```

En muchas ocasiones, después de plasmar las ideas de un proyecto y transcribir el código en el IDE de Arduino, y tras realizar toda una serie de cambios, pruebas y modificaciones dentro del propio *sketch*, podría ser que el código se acabe asemejando más a un maremágnum de instrucciones y números que a un programa bien estructurado, ocasionando que al cabo de un tiempo pueda resultarnos complicado volver a comprender el código que se había escrito.

Para ello, las funciones de usuario establece el código en bloques o partes perfectamente diferenciadas y esto reporta las siguientes ventajas a la hora de programar:

- Mejor compresión del código
- El código se convierte en escalable. Esto quiere decir que si se necesita añadir algunas líneas más, podremos hacerlo sin apenas complicaciones al estar debidamente estructurado.
- El código se ve más claro y limpio, permitiendo deshacernos de posibles variables o procesos que no necesita nuestro programa.

Para realizar esto podemos crear nuestras propias funciones, que serán llamadas por el programa principal en el mismo *sketch,* pero que nos dará una visión más limpia del código, más clarificadora y apta para futuras modificaciones. Veamos cómo implementarlo.

Para invocar a la función simplemente escribiremos, allí donde deseemos que se ejecute dentro del **void loop()** el nombre de nuestra función **mi_funcion(),** y fuera del **void loop()** escribiremos el bloque con nuestro código de nuestra función **void mi_funcion() { }**

Veamos cómo se coloca en la estructura de un programa:

```
void setup(){
}
void loop(){
mi_funcion();           // llama a la función mi_funcion
}
void mi_funcion(){     // bloque de comandos de "mi_funcion"
}
for(int  i=0; i<8; i++){  // crea un bucle tipo for utilizando la
variable i de 0 a 7
analogWrite(ledPin, parpadeo[i]);   //  escribe en la salida PIN 10 el
valor al  que apunta i  dentro del array parpadeo[]
delay(200);  // espera  200ms
}
}
```

## 6. ARITMÉTICA Y LÓGICA

En varias ocasiones necesitaríamos incluir operadores aritméticos en nuestros programas para gobernar Arduino. Un ejemplo muy típico es el crear contadores mediante una variable. Esta variable se va incrementando o decrementando según una condición o la ejecución de una determinada instrucción, haciendo las veces de variable contador.

En Arduino contamos con los operadores aritméticos más comunes, es decir, adición (suma), sustracción (resta) producto (multiplicación) y cociente (división). Otras operaciones matemáticas, como la potenciación o la raíz cuadrada o funciones como el seno, el coseno y el logaritmo también están presentes en la programación de Arduino.

Podemos implementar estas operaciones en cualquier lugar del código, dentro de los bloques "**void setup**" y "**void loop**".

En este manual nos centraremos en los operadores básicos: suma, resta, multiplicación y división.

Estos operadores aritméticos que se incluyen en el entorno de programación de Arduino, suma, resta, multiplicación y división, devuelven la suma, diferencia, producto, o cociente, respectivamente, de dos operandos:

```
y = y + 3;
x = x - 7;
i = j * 6;
r = r / 5;
```

La operaciones se efectúa teniendo en cuenta el tipo de datos que hemos definido para los operandos **(int**, dbl, **float**, etc..), por lo que, por ejemplo, si definimos 9 y 4 como enteros "int", 9 / 4 devuelve de resultado 2 en lugar de 2,25 ya que el 9 y 4 son valores de tipo entero "int" (enteros) y no se reconocen los decimales con este tipo de datos.

Esto también significa que la operación puede sufrir un desbordamiento si el resultado es más grande que lo que puede ser almacenada en el tipo de datos. Recordemos el alcance de los tipos de datos numéricos que ya hemos explicado anteriormente.

Si los operandos son de diferentes tipos, para el cálculo se utilizará el tipo más grande de los operandos en juego. Por ejemplo, si uno de los números (operandos) es del tipo **float** y otra de tipo **integer**, para el cálculo se utilizará el método de **float** es decir el método de coma flotante.

Elija el tamaño de las variables de tal manera que sea lo suficientemente grande como para que los resultados sean lo precisos que usted desea. Para las operaciones que requieran decimales utilice variables tipo **float**, pero sea consciente de que las operaciones con este tipo de variables son más lentas a la hora de realizarse el cómputo.

Mediante los siguientes ejemplos se muestran estas operaciones:

## Suma

```
/* Operación suma. Este ejemplo simplemente realiza la suma 3+2 una y
otra vez*/
int A;  //asignamos la variable A
void setup() {
}
void loop() {
A=3+2;   //la variable A vale 5
}
```

Estas operaciones se pueden implementar tanto en el bloque “void setup” como el bloque “void loop”.

## Resta

```
/* Operación resta. Este ejemplo simplemente realiza la resta 3-2 una y
otra vez*/

int A;          //asignamos la variable A
int B=2;        //asignamos a la variable B con el valor 2
void setup() {
}
void loop() {
A=3-B;          //La variable A vale 1
}
```

Tal como se ve en el ejemplo, podemos combinar variables y números en las operaciones aritméticas en los programas.

## Multiplicación

```
/* Operación multiplicación. Este ejemplo simplemente realiza la
multiplicación 3*2 una y otra vez*/
int A;  //asignamos la variable A
void setup() {
}
void loop() {
A=3*2;               //la variable A vale 6
}
```

### División

```
/* Operación división. Este ejemplo simplemente realiza la división 4/2
una sola vez, ya que no está dentro del bloque loop*/
int A;  //asignamos la variable A
int B=2;  // asignamos a la variable B el valor 2
void setup() {
A=4/B;    //A es igual a 2
}
void loop() {
}
```

## Asignaciones compuestas

Las asignaciones compuestas combinan una operación aritmética con una variable asignada. Estas son comúnmente utilizadas en los bucles. Estas asignaciones compuestas pueden ser:

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

**Nota:** Por ejemplo, x * = 3 hace que x se convierta en el triple del antiguo valor x y por lo tanto x es reasignada al nuevo valor.

## Operadores de comparación

Las comparaciones de una variable o constante con otra se utilizan con frecuencia en las estructuras condicionales del tipo **if..** para testear si una condición es verdadera. En todo programa para Arduino llega el momento en que debemos emplear operaciones lógicas, esto es: igual a, mayor que, menor que, no es igual a, mayor o igual que y menor o igual que. En los ejemplos que siguen en las próximas páginas se verá su utilización práctica usando los siguientes tipo de condicionales:

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

Ejemplos de aplicación.

```
/* variable A se incrementa en una unidad */

int A=5;        // asignamos a la variable A el valor 5
void setup(){
A++;            // la variable A suma 5+1
}
void loop() {
}
```

```
/* variable A se decrementa en una unidad */
int A=5;         //asignamos a la variable A el valor 5
void setup(){
A--;             //la variable A resta 5-1
}
void loop(){
}
```

## Operadores lógicos

Los operadores lógicos son usualmente una forma de comparar dos expresiones y devolver un VERDADERO o FALSO dependiendo del operador. Existen tres operadores lógicos, **AND (&&), OR (||)** y **NOT (!)**, que a menudo se utilizan en instrucciones de tipo **if**..:

### Lógica AND:

Es la **"y"** que se utiliza para evaluar más de una condición. Estos operadores lógicos se emplean cuando hacemos preguntas dl tipo "Si A es mayor que B y B es mayor que C…" Se representa mediante **"&&**".

```
/* Operando AND*/

int A=5;   //asignamos a la variable A el valor 5
int B=3;    //asignamos a la variable B el valor 3
int C=7;    //asignamos a la variable C el valor 7
void setup(){
}
void loop(){
if (A>B &&  A>C){ //si A es mayor que B y A mayor que C
A=A+1;            //A vale 6
}
}
```

En este ejemplo, leeremos: Si A es mayor que B y A es mayor que C, A vale 5+1.

### Lógica OR:

Es la "o" que se utiliza para escoger una u otra condición o sentencia. Lo empleamos cuando decimos: "Si A es mayor que B o es mayor que C…" Se representa mediante "||".

```
/* Operando OR*/

int A=5;    //asignamos a la variable A el valor 5
int B=3;    //asignamos a la variable B el valor 3
int C=7;    //asignamos a la variable C el valor 7
void setup() { }
void loop() {
if (A>B ||  A>C) {      //si A es mayor de B ó A mayor que C
A=A+B;                  // A vale 8
}
}
```

En este ejemplo, leeremos: Si A es mayor que B o A mayor que C, A vale 5+3

### Lógica NOT:

Es la negación. Se representa mediante "**!**". En este ejemplo, leeremos Si (if) no es A, C vale 3+2.

```
/* Operando NOT*/

int A=5;   //asignamos a la variable A el valor 5
int B=3;   //asignamos a la variable B el valor 3
int C=7;   //asignamos a la variable C el valor 7
void setup(){
}
void loop(){
if ( !A){      // si no es A
C=B+2;         // c vale 3+2
}
}
```

## Mínimo y Máximo

Son instrucciones utilizadas en la programación de Arduino para escoger el valor mínimo o máximo entre dos números y el resultante guardarlo en una variable.

### min (x,y)

La instrucción **min(x,y)** va a comparar dos valores devolviendo el menor de ellos, calculando el mínimo de dos números para cualquier tipo de datos devolviendo el número más pequeño.

```
dato=min(sensor,100);  //dato toma el valor mínimo entre "sensor" y 100
```

En este ejemplo “dato” va a tomar el valor del sensor siempre que este valga menos de 100. Esta instrucción puede utilizarse para limitar superiormente el valor de “dato”.

## max (x,y)

La instrucción **max(x,y)** va a comparar dos valores devolviendo el mayor de ellos, calculando el máximo de dos números, para cualquier tipo de datos devolviendo el número mayor de los dos.

```
dato=max(sensor,100);  //dato toma el valor máximo entre “sensor” y 100.
```

De esta forma nos aseguramos de que valor será como mínimo 100.

Esta función funciona igual que **min(x,y)**, la diferencia es que va a entregar el mayor de los dos números, esto puede ser utilizado para inferiormente el valor que tomará la variable “dato”.

## 7. INSTRUCCIONES DE CONTROL

Este tipo de instrucciones permiten al programador controlar las acciones y decisiones que debe tomar Arduino frente a un problema, ya sea un robot autónomo móvil o ya sea para realizar un proyecto de domótica en el que se desee subir o bajar una persiana automáticamente.

Estas instrucciones nos permiten controlar el flujo de datos que van siendo capturados o generados por Arduino y sus sensores, actuadores o componentes electrónicos.

Estos operandos son: IF, ELSE, FOR, WHILE, DO-WHILE, SWITCH, BREAK, CONTINUE.

### Condicionales

Los condicionales son elementos que chequean un estado o condición y si esta condición se cumple se pasa a ejecutar las sentencias englobadas dentro de la condición.

#### if (Si condicional)

**if** es la instrucción que se utiliza para probar si una determinada condición se ha alcanzado, como por ejemplo averiguar si un valor analógico está por encima de un cierto número, y ejecutar una serie de declaraciones (operaciones) que se escriben dentro de llaves, si es verdad. Si es falso (la condición no se cumple) el programa salta y no ejecuta las operaciones que están dentro de las llaves.

El formato para **if** es el siguiente:

```
if (unaVariable ?? valor) {  //si unaVariable es (comparación) valor

ejecutaInstrucciones;         //ejecuta las siguientes instrucciones
}
```

En el ejemplo anterior se compara una variable con un valor, el cual puede ser una variable o constante. Si la comparación, o la condición entre paréntesis se cumple (es cierta), las declaraciones dentro de los corchetes se ejecutan. Si no es así, el programa salta sobre ellas y sigue.

```
if (A==3) {   //si A es igual a 3, haz lo siguiente….
A=A+5;        //A almacenará el resultado de 3+5
C=5+4;        //C almacenará el resultado de 5+4
B=A+C;        //B almacenará el resultado de sumar A+C
}
C=4;         // si la condición no se cumple (A=3), C almacena un 4
```

Cuando la condición no se cumple**, A=3,** Arduino salta todo el bloque del if y pasa directamente a ejecutar la línea **C=4.**

**Nota:** Tenga en cuenta el uso especial del símbolo '=', poner dentro de if (A = 3), podría parecer que es válido pero sin embargo no lo es ya que esa expresión asigna el valor 3 a la variable A, por eso dentro de la estructura if se utilizaría A==3 que en este caso lo que hace el programa es comprobar si el valor de A es 3. Ambas cosas son distintas por lo tanto dentro de las estructuras if, cuando se pregunte por un valor se debe poner el signo doble de igual "**==**"

## if….else (si…entonces)

**if… else** viene a ser una estructura que se ejecuta en respuesta a la idea "si esto no se cumple haz esto otro". Por ejemplo, si se desea probar una entrada digital, y hacer una cosa si la entrada fue alto o hacer otra cosa si la entrada es baja, por ejemplo:

```
if (inputPin == HIGH){  // si el valor de la entrada inputPin es alto
digitalWrite(13, LOW);     //desactiva el pin 13
}
else
{
digitalWrite(2, HIGH);  //ejecuta si no se cumple la condición de activar
el pin 2
}
```

**else** puede ir precedido de otra condición de manera que se pueden establecer varias estructuras condicionales de tipo unas dentro de las otras (anidamiento) de forma que sean mutuamente excluyentes pudiéndose ejecutar a la vez. Es incluso posible tener un número ilimitado de estos condicionales. Recuerde sin embargo qué sólo un conjunto de declaraciones se llevará a cabo dependiendo de la condición probada:

```
if (inputPin < 500)
{
instruccionesA;              // ejecuta las operaciones A
}
else if (inputPin >= 1000)
{
instruccionesB;             // ejecuta las operacione B
}
else
{
instruccionesC;            // ejecuta las operaciones C
}
```

**Nota:** Una instrucción de tipo **if** prueba simplemente si la condición dentro del paréntesis es verdadera o falsa. Esta declaración puede ser cualquier declaración válida. En el anterior ejemplo, si cambiamos y ponemos (inputPin == HIGH). En este caso, el estamento **if** sólo chequearía si la entrada especificado está en nivel alto (HIGH), o +5 v.

## Bucles

Los bucles son elementos que hacen que el programa entre en un ciclo de repetición mientras se cumpla las condiciones del bucle.

Un bucle es aproximadamente un marco, el código dentro del cual se ejecuta de arriba a abajo y se repite desde el principio hasta que llega al final. Este comportamiento continúa mientras se cumpla alguna condición. Hay dos bucles principales con los que trabajaremos, estos son **for** y **while**.

### for (para…)

La declaración **for** se usa para repetir un bloque de sentencias encerradas entre llaves un número determinado de veces, se suele utilizar como temporización. Cada vez que se ejecutan las instrucciones del bucle se vuelve a testear la condición. La declaración **for** tiene tres partes separadas dentro del paréntesis que determina la variable y elementos estos son: **inicialización; condición;** y **expresión**) los dos primeros van con punto y coma (;), veamos su sintaxis:

```
 for  (inicialización;  condición;  expresión)
{
ejecutaInstrucciones;
}
```

La **inicialización** de una variable local se produce una sola vez y la **condición** se testea cada vez que se termina la ejecución de las instrucciones dentro del bucle. Si la condición sigue cumpliéndose, las instrucciones del bucle se vuelven a ejecutar. Cuando la condición no se cumple, el bucle termina.

El siguiente ejemplo inicia el entero A en el 0, y la condición es probar que el valor es inferior a 20 y si es cierta A se incrementa en 1 y se vuelven a ejecutar las instrucciones que hay dentro de las llaves hasta que el valor de A sea mayor de 20.

```
for (int A=0; A<20; A++)   // declara  A,  prueba que es menor que 20,
incrementa  A en  1
{
digitalWrite(13,  HIGH);    //  enciende el led
delay(250);                 // espera 250 milisegundos
digitalWrite(13,  LOW);     // apaga el led
delay(250);                 // espera 250 milisegundos
}
```

La primera parte declara ahí mismo la variable A como entero, y le asignamos el valor de inicio 0. La segunda parte introduce la condición para que el bucle se repita. En este caso, hasta que A sea menor que 20.La tercera parte incrementa en una unidad a la variable A hasta que llegue a 20, donde se cumplirá la condición expuesta en la segunda parte y el bucle finalizará.

En un bucle **for**, puedes hacer varios contadores, varias condiciones y varios incrementos, separándolos mediante el operador de coma, ver un ejemplo:

```
// declara i y j
// suma i + 1 y j + 2
for ( byte i = 0, j = 0; i < 10; i ++, j + = 2 ) {
 // i cambio de 0 a 9
// y j cambia de 0 a 18
}
```

**Nota:** El bucle en el lenguaje C es mucho más flexible que otros bucles encontrados en algunos otros lenguajes de programación, incluyendo BASIC. Cualquiera de los tres elementos de cabecera puede omitirse, aunque el punto y coma es obligatorio. También las declaraciones de inicialización, condición y expresión puede ser cualquier estamento válido en lenguaje C sin relación con las variables declaradas. Estos tipos de estados son raros pero permiten  disponer soluciones a algunos problemas de programación raras.

## while (mientras…)

Un bucle del tipo **while** es un bucle de ejecución indefinido hasta que se cumpla la condición que está dentro del paréntesis en la cabecera del bucle. La variable de prueba tendrá que cambiar para salir del bucle. La situación podrá cambiar a expensas de una expresión dentro el código del bucle o también por el cambio de un valor en una entrada de un sensor.

```
while  (unaVariable  ??  valor)
{
ejecutarSentencias;
}
```

En el siguiente ejemplo la condición de **while** es que **mientras** la variable A sea menor que 200, el diodo led del pin 13 está encendido, hasta que A no sea inferior a 200.

```
while (A < 200)          // comprueba si A es menor que 200
{
pinWrite (13, HIGH);    //  mientras A sea menor de 200 enciende el led 13
A++;                    //  incrementa la variable en 1
}
pinWrite(13, LOW);      // apaga  el led 13 cuando A sea mayor de 200
```

Veamos el siguiente ejemplo con **while**, que consiste en parpadear un diodo led **mientras** que A sea menor de 50, cuando sea mayor de 50 sale del bucle y salta a la siguiente instrucción apagando el led pin2 y encendiendo el led2 pin4.

```
int A;                    // declaramos variable A
int led=2;                // declaramos led en pin 2
int led2=4;               // declaramos led 2 en pin 4
void setup() {
pinMode (led, OUTPUT);    //ponemos pin 2 de salida
pinMode (led2, OUTPUT);   //ponemos pin 4 de salida
}
void loop() {
while ( A<50){            //mientras que A sea menor de 50 ejecuta…
digitalWrite(led,HIGH);   // enciende el led
delay(1000);              // espera 1 segundo
digitalWrite(led, LOW);   //apaga el led
delay(1000);              //espera 1 segundo
A++;                      // A suma 1
}
digitalWrite(led,LOW);    //apaga el led definitivamente
digitalWrite(led2, HIGH); //enciden el led2
}
```

En este otro ejemplo se crean dos bucles con while, el primer bucle, **(while(digitalRead(pulsador)==LOW)** sirve para que la función digitalRead lea repetidamente el valor del pin digital de entrada en el que está conectado un pulsador. Mientras que este pin tenga valor **LOW** significa que el pulsador no se ha presionado. El bucle deja de repetirse cuando se presiona el pulsador, puesto que entra el valor **HIGH** del siguiente bucle, **(while(digitalRead(pulsador)==HIGH**) permitiendo que la ejecución continúe en la siguiente instrucción.

Esta configuración de dos bucles de while sirve para que la ejecución no continue mientras entre valor **HIGH** por el pin digital D10 del pulsador. Esto evita que se cuenten los rebotos y las pulsaciones largas y contará como una única pulsación.

```
const int pulsador=10;
void setup(){
pinMode (pulsador,INPUT);
void loop() {
   while (digitalRead(pulsador)==LOW);   //mientras pulsador no se pulse
   while (digitalRead(pulsador)==HIGH);  //mientras pulsador se pulse
}
```

## do…..while (haz esto…. mientras)

El bucle **do… while** funciona de la misma manera que el bucle **while**, con la salvedad de que la condición se prueba al final del bucle, por lo que el bucle siempre se ejecutará al menos una vez.

Esta instrucción ejecuta una serie de sentencias por un periodo indefinido, mientras no se cumple la condición que está dentro del paréntesis de **while**. Aquí se ejecutan las instrucciones primero, y más tarde se evalúa la condición mediante el **while**.

Ejemplo:

```
do { // haz esto…
A=B+1;  //A=B+1
w++;    // ve sumando 1 a la variable w
}  while  (w=10);   //hasta que w valga 10
```

El siguiente ejemplo asigna el valor leído **leeSensor()** a la variable 'x', espera 50 milisegundos, y luego continua mientras que el valor de la 'x' sea inferior a 100:

```
do {                  //haz esto
x = leeSensor();      //guarda el valor del sensor en x
delay(50);            //espera 50 milisegundos
}
while (x < 100);  //mientras que x sea menor de 100
```

En el siguiente ejemplo la programación trata de activar un sensor de infrarrojo puesto de entrada en el pin 28 A0, que cuando se pulsa un mando a distancia por infrarrojo el valor del sensor hace activar un bucle **do…while** durante un determinado tiempo encendiendo un diodo led y al finalizar el bucle se apaga el led y se enciende el led2.

```
int valor;                  // declaramos variable valor
int x=0;                    //declaramos variable x a 0
int leeSensor=A0;           //declaramos variable leeSensor en el pin A0
int led=2;                  //declaramos variable led del pin 2
int led2=4;                 //declaramos  variable led2 del pin 4
void setup() {
pinMode (leeSensor, INPUT);   // ponemos pin 28 de entrada
pinMode (led, OUTPUT);        //ponemos pin 2 de salida
pinMode (led2, OUTPUT);       //ponemos pin 4 de salida
}
void loop() {
valor=analogRead (leeSensor);   // almacena el valor del pin 28
if (valor<=50) {                // si valor es menor o igual a 50
  do {                          // haz lo siguiente
    x++;                        // añade 1 a x
    digitalWrite(led, HIGH);    // encendemos led
    delay(500);                 // espera medio segundo
}
 while (x<50);   // ejecuta el bloque anterior mientras x sea menor de 10
digitalWrite (led, LOW);        // apaga el led
digitalWrite (led2, HIGH);      // enciende el led2
}
}
```

## switch/case y break

Estas dos sentencias (**SWITCH/CASE** y **BREAK**) son un complemento la una de la otra, al igual que ocurre con las instrucciones IF y ELSE.

Cuando Arduino se encuentra con un bloque formado por estas dos instrucciones, actúa de la siguiente manera:

1. Se compara el valor de la variable que posee **switch** (entre paréntesis) con el valor de una variable del programa que figura en una instrucción **case**.
2. Si el valor de la variable es igual a algunos de los valores que contiene la instrucción **case**, se ejecutará esa parte del código.

Veamos un ejemplo:

```
switch{
case 5: A=A/2;
break;
caso 10: B=A+5;
break;
case 15: C=A+B;
break;     }
```

La instrucción **default** se utiliza cuando no se cumple con ninguna de las sentencias de **case**.

```
switch{

case 5: A=A/2;
break;
caso 10: B=A+5;
break;
case 15: C=A+B;
default: A=A+B;  // si no se cumple con ninguna condición, por defecto
ejecuta A=A+B
break;
}
```

Si no se cumple con ninguna condición, por defecto el programa pasa a default, ejecutándose las sentencias que contiene.

Al principio se ha comentado que **switch** y **break** eran unas sentencias que se complementaban y es que la sentencia **break** se emplea para devolver la secuencia del programa a la siguiente instrucción que se va a ejecutar después de haber entrada en la estructura **switch**.

Si prescindimos de **break**, el programa se quedaría en ese punto, esperando la sentencia **break** para seguir con el programa.

**Nota:** Todos estos controles no llevan **punta y coma**, la operación que tienen que hacer van entre **paréntesis (…)** y en un bloque limitado entre **llaves {…}.**

## Elementos de control de flujo

El lenguaje de programación de Arduino, como muchos lenguajes de programación, ha heredado características de los primeros lenguajes de programación BASIC, estas características incluyen sentencias para el control del flujo como **GOTO**, **RETURN** y **BREAK**. Estas sentencias no son muy apreciadas a la hora de programar, ya que en muchas ocasiones rompen el flujo del programa y dificultan la composición del mismo. Además en lenguajes de programación de alto nivel como el que usa Arduino se pueden usar funciones y otros elementos alternativos para realizar las mismas tareas, aunque conviene conocer la existencia de estas funciones ya que pueden ser de utilidad en algunas ocasiones.

## goto

Esta sentencia realiza un salto a cualquier parte del programa que esté marcada con la etiqueta correspondiente, la posición desde la que realicemos el salto quedará almacenada en la pila del programa para que podamos regresar al lugar desde donde realicemos el salto.

```
if (x==110){    // si x es igual 110
goto marca1;    // salta a la etiqueta marca1
}
marca1;   //etiqueta a la que saltará (puede estar en cualquier parte)
```

Los saltos que podemos realizar con Arduino están limitados a la pila de programa, no pueden ser infinitos, por lo que no es recomendable anidar bucles y saltos, ya que podemos romper el flujo de programa.

## returrn

La sentencia **return** se usa para volver de un salto de "**goto**". En el momento que es leída por el programa, se cargará la última dirección almacenada en la pila de programa, esto hará que se regrese a la posición desde la que se realizó el último salto.

## break

La sentencia **break** es una sentencia que se debe evitar a toda costa, tan solo debemos usarla cuando sea totalmente necesario. Esta sentencia rompe la iteración del bucle donde se encuentre, haciendo salir al programa del bucle sin tener en cuenta que se cumplan las condiciones para salir del mismo.

```
for (int x=0; x<255; x++) {  //bucle de x de 0 a 255
while (sensor<100) {        // bucle mientras sensor sea menor de 100
x=0;                        //ponemos la vaiable x a 0
break;      //rompe el bucle while saliendo al bucle for
}
}
```

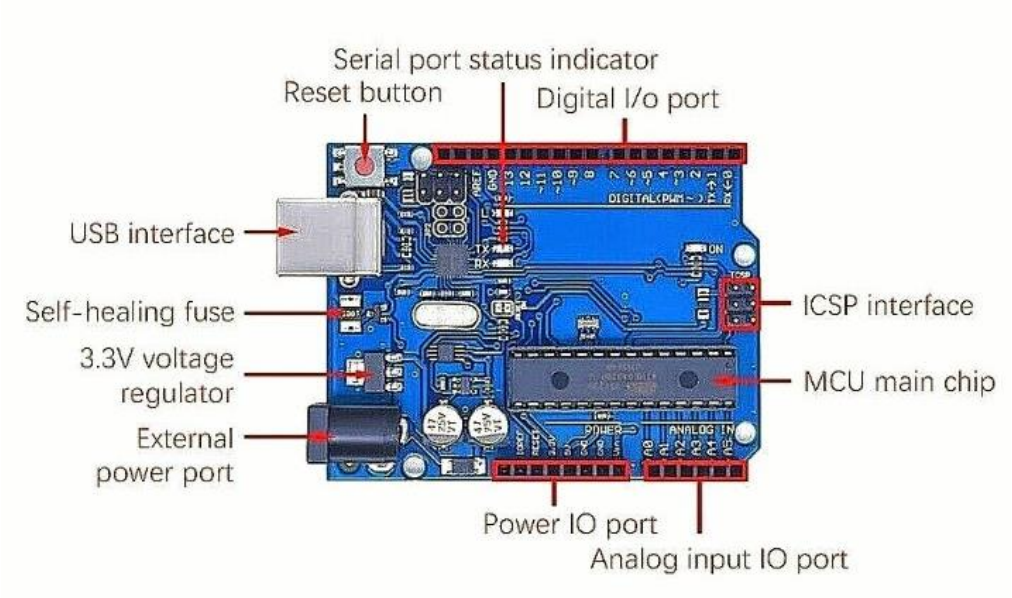
**break** le permite dejar el ciclo antes de lo programado, puede usarlo por condición o de cualquier manera conveniente. Por ejemplo, salir del ciclo antes de lo programado cuando se alcance un cierto valor:

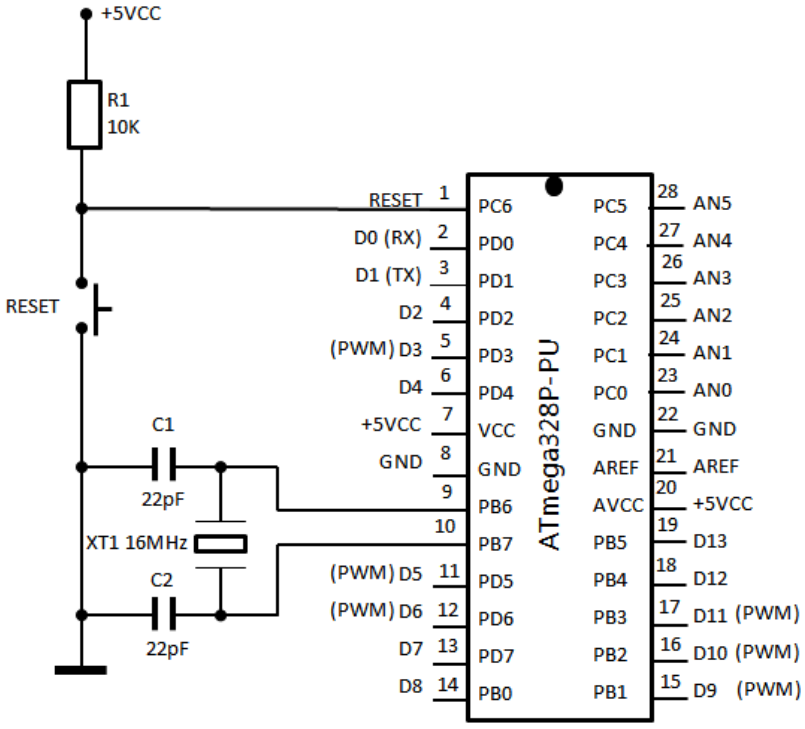
```
for ( int i = 0; i < 100; i ++ ) {
    if ( i == 50 ) break ;  // abandona el bucle al llegar a 50
}
```

## 8. ASIGNACIÓN DE ENTRADAS Y SALIDAS

Arduino es una plataforma que nos permite interactuar con el medio exterior gracias a la multitud de sensores, módulos y complementos que posee.

Para poder decirle a Arduino que deseamos adquirir cierta información mediante un sensor, debemos configurar los pines de que dispone para tal efecto. Para ello, debemos conocer el patillaje del microcontrolador para identificar sus pines e implementarlo en la programación.




**Microcontrolador ATmega328P-PU**

Antes de determinar el pin deberá adquirir información **analógica** o **digital**, deberemos declarar si deseamos que sea de entrada o de salida, es decir, si nos interesa que introduzca datos en Arduino o los extraiga. Cuando utilicemos entradas o salidas analógicas deberemos declarar cualquiera de los 6 nombres introduciendo AN0, AN1, AN2, AN3, AN4 o AN5, no el número de pin o patillaje del microcontrolador.

La siguiente instrucción se deberán declarar en el bloque **void setup().**

## pinMode (pin,modo)

Esta instrucción es utilizada en la parte de configuración de la función **void setup()** y sirve para configurar el modo de trabajo de un PIN pudiendo ser **INPUT** (entrada) u **OUTPUT** (salida).

```
pinMode(pin, OUTPUT);      //configura 'pin' como salida
```

Los terminales de Arduino, por defecto, están configurados como entradas, por lo tanto no es necesario definirlos en el caso de que vayan a trabajar como entradas. Los pines configurados como entrada quedan, bajo el punto de vista eléctrico, como entradas en estado de alta impedancia.

Estos pines tienen a nivel interno una resistencia de 20 KΩ a las que se puede acceder mediante software. Estas resistencias se acceden de la siguiente manera:

```
pinMode(pin,  INPUT);         // configura el 'pin' como entrada
digitalWrite(pin,  HIGH);     //  activa las resistencias internas
```

Las resistencias internas normalmente se utilizan para conectar las entradas a interruptores. En el ejemplo anterior no se trata de convertir un pin en salida, es simplemente un método para activar las resistencias interiores.

Los **pins** configurado como **OUTPUT** (salida) se dice que están en un estado de baja impedancia estado y pueden proporcionar 40 mA (miliamperios) de corriente a otros dispositivos y circuitos. Esta corriente es suficiente para alimentar un diodo LED (no olvidando poner una resistencia en serie), pero no es lo suficiente grande como para alimentar cargas de mayor consumo como relés, solenoides, o motores.

Un cortocircuito en las patillas Arduino provocará una corriente elevada que puede dañar o destruir el chip **Atmega**. A menudo es una buena idea conectar en la **OUTUPT** (salida) una resistencia externa de 470 o de 1000 Ω.

## Entradas y Salidas Digitales

Arduino está dotado de una serie de 14 pines de entradas y salidas digitales, estos son: pin2 (D0), pin 3 (D1), pin 4 (D2), pin 5 (D3), pin 6 (D4), pin 11 (D5), pin 12 (D6), pin 13 (D7), pin 14 (D8), pin 15 (D9), pin 16 (D10), pin 17 (D11), pin 18 (D12) y pin 19 (D13). Esto significa que la información que sea introducida o extraída será de carácter digital. Cuando la información es de carácter digital nos estamos refiriendo a que los datos son 1 o 0, alto o bajo, 5 voltios o 0 voltios.

Por lo tanto, estos pines serán idóneos para conectar sensores o dispositivos que trabajen con valor digital, es decir, si el sensor da información, enviará 5 voltios (1), por el contrario si el sensor no detecta actividad, enviará 0 voltios (0).

No todos los sensores trabajan o pueden trabajar de este modo, ya que hay algunos que proporcionan valores intermedios del 0 a los 5 voltios. Estos sensores se conectarán a las entradas y salidas analógicas.

### digitalRead (pin)

El valor que lee esta instrucción de un sensor (pin) se almacena en una variable que define el usuario. Es por eso que normalmente veamos la función **digitalRead()** precedida por una variable y un igual.

```
sensor = digitalRead(Pin); // guarda en la variable sensor el valor
digital leído en el Pin
```

En el ejemplo, lee el valor de pin (definido como digital) y lo almacena, dando un resultado **HIGH** (alto) o **LOW** (bajo), en **sensor**. El pin se puede especificar ya sea como una variable o una constante.

En el siguiente ejemplo encendemos un led cuando se detecta la pulsación de un botón.

```
/*Pulsador botón para encendido de un led*/
const int botonPin=2;         // declara D2 el pin pulsador botón
const int ledPin=4;             // declara D4 pin LED
int Estadoboton=0;            // declara variable a 0  para Estado boton
void setup() {
pinMode(ledPin, OUTPUT);     // inicializa el pin LED como salida
pinMode(botonPin, INPUT);    // inicializa el pin boton como entrada:
}
void loop() {
Estadoboton = digitalRead(botonPin); //lee el estado del valor pin botón
```

```
if (Estadoboton == HIGH) { //presiona el botón si estado botón es igual
alto
digitalWrite(ledPin, HIGH);    //  enciende el led
  delay(5000);                 // espera 5 segundos encendido
  }
else {                         // de lo contrario apaga el led
digitalWrite(ledPin, LOW);     //apaga el led
}
}
```

## digitalWrite (pin, value)

Cuando el pin sea digital y esté previamente configurado como **OUTPUT** (salida) el valor **HIGH** o **LOW** poniendo un 1 o un 0 a la salida, se empleará la siguiente función preestablecida en Arduino: **digitalWrite (pin, valor);** El pin se puede especificar ya sea como una variable o como una constante (0=13).

```
digitalWrite(pin, HIGH); // deposita en el 'pin' un valor HIGH (alto o 1)
```

El siguiente ejemplo lee el estado de un pulsador conectado a una entrada digital y lo visualiza en un LED de salida digital y al mismo tiempo una salida analógica hace sonar unos tonos:

```
/*Boton con sonido y led*/

int piezo=A5;  //declaramos la variable piezo para el puerto analógico A5
const int botonPin=2;      // declara D2 el pin botón
const int ledPin=4;        // declara D4 pin LED
int Estadoboton=0;         // declara variable para Estado boton
void setup() {
pinMode (piezo, OUTPUT);   // inicializa piezo como salida
pinMode(ledPin, OUTPUT);    // inicializa el pin LED como salida
pinMode(botonPin, INPUT);   // inicializa el pin boton como entrada:
}
void loop() {
Estadoboton = digitalRead(botonPin);   //lee el estado del valor pin
botón

if (Estadoboton == HIGH) { //cuando presiona el botón si estado botón es
igual alto
digitalWrite(ledPin, HIGH);   // enciende el led
for (int i=100; i<=3000; i=i+100) { //hacemos un temporizador
tone (piezo,220,300);          //activa tono de sonido
delay(320);                    // espera
  }   }
else {                         // de lo contrario apaga el led
digitalWrite(ledPin, LOW);     //apaga el led
 noTone(piezo);                // apaga el tono } }
```

## Entradas y Salidas Analógicas

Arduino está dotado de una serie de pines de entrada y salida analógicas: pin 23 (AN0), pin 24 (AN1), pin 25 (AN2), pin 26 (AN3), pin 27 (AN4) y pin28 (AN5). Por lo que a diferencia de la información digital 1 o 0 (5v, 0V), aquí nos podemos encontrar con que Arduino debe leer o escribir datos donde sus valores pueden oscilar o variar dentro del rango de 0 y 5 voltios.

Por lo tanto, estos pines serán idóneos para conectar sensores o dispositivos que al ser analógicos puedan dar cualquier valor comprendidos entre 0 y 5 voltios.

Como los valores comprendidos entre 0 y 5 voltios pueden ser muchos, Arduino pone el límite en 1.024 valores. Dicho de otro modo, Arduino asignará un (cero) para representar los 0 voltios, asignará el número 512 para representar 2,5 voltios y el número 1.023 para los 5 voltios.

Un posible ejemplo de sensor que es susceptible de ser conectado en un pin analógico es lo que se conoce como fotorresistencia o LDR. Una LDR varía su resistencia según la intensidad de la luz, estando los valores que proporciona comprendidos entre 0 y 1024.

Otro ejemplo sería el sensor **LM35**. Se trata de un sensor de temperatura. Los valores que proporciona no serán 0 o 5 voltios (1 o 0, alto o bajo), sino que empleará un rango de valores según la temperatura, 1º, 8,5º, 25º, etc.

Como ocurre en el caso anterior de los pines digitales, las funciones que controlan a los pines analógicos son dos: **analogWrite()** y **analogRead()**.

### analogRead(pin)

El valor que lee esta función de un sensor normalmente se almacena en una variable que define el usuario. Es por esto que normalmente veremos la función **analogRead()** precedida por una variable y un igual.

Por lo tanto el valor que lee esta instrucción debe ser de un determinado pin analógico definido como entrada analógica con una resolución de 10 bits. Esta instrucción sólo funciona en los pines (AN0…AN5). El rango de valor que podemos leer oscila de 0 a 1023.

```
valor = analogRead(pin); //asigna a valor  lo que lee en la entrada ´pin'
```

Ejemplo:

```
/* Parte de código de un programa para medir la temperatura con un
sensor*/
int temp=A0;       // asociamos el pin analógico A0 con la variable temp
void setup() {
pinMode (temp, INPUT);   // configuramos el pin A0 (temp) como entrada
}
void loop(); {
valor= analogRead(temp); // la variable valor almacena el dato de temp
}
```

## analogWrite (pin, value)

Cuando el pin sea analógico y esté configurado como **OUTPUT** emplearemos la instrucción **analogWrite().**

Esta instrucción sirve para escribir un pseudo valor analógico utilizando el procedimiento de **modulación por ancho de pulso** (**PWM**) a uno de los pin de Arduino marcados como "**pin PWM**". El más reciente Arduino, que implementa el chip ATmega328P, permite habilitar como salidas analógicas tipo **PWM** los pines 3, 5, 6, 9, 10 y 11.

Los modelos de Arduino más antiguos que implementan el chip ATmega8, solo tiene habilitadas para esta función los pines 9, 10 y 11. El valor que se puede enviar a estos pines de salida analógica puede darse en forma de variable o constante, pero siempre con un margen de 0…255.

```
analogWrite(pin, valor);  //escribe 'valor' en el 'pin' definido como
analógico
```

Si enviamos el valor 0 genera una salida de 0 voltios en el pin especificado; un valor de 255 genera una salida de 5 voltios de salida en el pin especificado. Para valores de entre 0 y 255, el pin saca tensiones entre 0 y 5 voltios = el valor HIGH de salida equivale a 5v (5 voltios). Teniendo en cuenta el concepto de señal **PWM**, por ejemplo, un valor de 64 equivaldrá a mantener 0 voltios de tres cuartas partes del tiempo y 5 voltios a una cuarta parte del tiempo; un valor de 128 equivaldrá a mantener la salida en 0 la mitad del tiempo y 5 voltios la otra mitad del tiempo, y un valor de 192 equivaldrá a mantener en la salida 0 voltios una cuarta parte del tiempo y de 5 voltios de tres cuartas partes del tiempo restante.

Debido a que esta es una función de hardware, en el pin de salida analógica (**PWN**) se generará una onda constante después de ejecutada la instrucción **analogWrite** hasta que se llegue a ejecutar otra instrucción **analogWrite** (o una llamada a **digitalRead** o **digitalWrite** en el mismo pin).

**Nota:** Las salidas analógicas a diferencia de las digitales, no necesitan ser declaradas como **INPUT** u **OUTPUT**.

El siguiente ejemplo lee un valor analógico de un pin de entrada analógica, convierte el valor dividiéndolo por 4, y envía el nuevo valor convertido a una salida del tipo **PWM** o salida analógica:

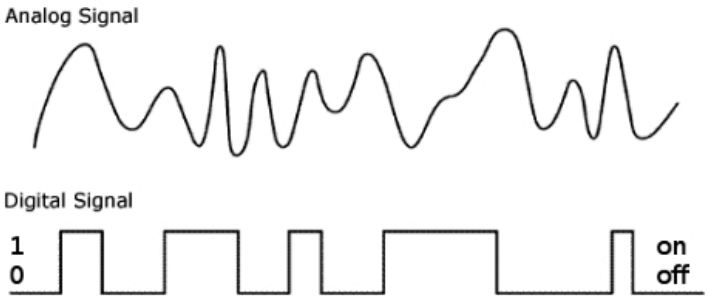
```
int  led = 10;           //  define  el pin 10 como ´led´
int  analog  = AN0;      // define el pin 0 como ´analog´
int  valor;              // define la variable ´valor´
void  setup() { }       //  no es necesario configurar entradas y salidas
void  loop()  {
valor=analogRead(analog);   // lee el pin 0 y lo asocia a la variable
valor
valor /=  4;              //divide valor entre 4 y lo reasigna a valor
analogWrite(led,  valor);  // escribe en el pin10  valor
}
```

Un ejemplo típico para comprender mejor esta función es el de variar la luminosidad de un led conectado a un pin analógico.

```
/* variacion de luminosidad de un led*/
int led = 3;  // declaramos el pin D3  PWM para  encender y apagar el LED
int brillo= 0;   // declaramos brillo a  0
int valor = 5; // declaramos valor a  5 para variar los pasos de
luminosidad del LED
void setup() {
pinMode(led, OUTPUT);  // declaramos el pin D3 led de salida
}
void loop() {
analogWrite(led, brillo); // pone analógicamente el punto de brillo en el
led pin D3
brillo = brillo + valor;  // añade  a brillo la suma de brillo y  valor
progresivamente
 if (brillo <= 0 || brillo >= 255) { //si brillo es menor o igual a 0  o
si es mayor o igual 255
 valor = -valor; //valor es igual a menos valor
}
 delay(50); //espera 50 milisegundos para ver  los efectos
}
```

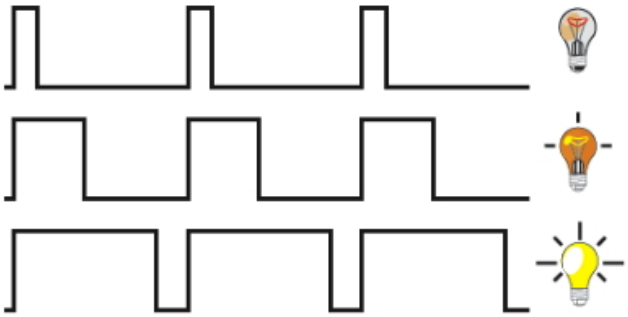
## 9. SEÑALES PWM EN ARDUINO

Muy a menudo en robótica, es necesario controlar sin problemas algún proceso, ya sea el brillo de un LED, la potencia de un calentador o la velocidad de rotación de un motor. Es bastante obvio que el control está directamente relacionado con el cambio de voltaje en el receptor: tanto el LED brillará de manera diferente como el motor girará a una velocidad diferente. Pero el problema es que solo un **DAC**, un convertidor de digital a analógico, puede controlar el voltaje, y nuestro microcontrolador no tiene un DAC incorporado, solo tenemos una señal digital, es decir, ya sea encendido o apagado:




Señales analógico digitales en Arduino

¿Es posible lograr un control suave de la señal digital? ...se puede. Imagine un ventilador funcionando a plena potencia, el voltaje es constante. Imagínese ahora que el voltaje se aplica por un segundo, y por un segundo – no, y así continúa «en un círculo». El ventilador comenzará a girar el doble de lento, pero lo más probable es que notemos los momentos de encendido y apagado, especialmente si el ventilador es pequeño. Un ventilador grande es más inerte y es posible que ni siquiera note los cambios de velocidad en dos segundos. Ahora puede encender el voltaje durante 0,5 segundos y apagarlo durante los 1,5 segundos restantes. El ventilador girará al 25% de la velocidad máxima. Acabamos de presentar la llamada señal PWM, modulación de ancho de pulso.

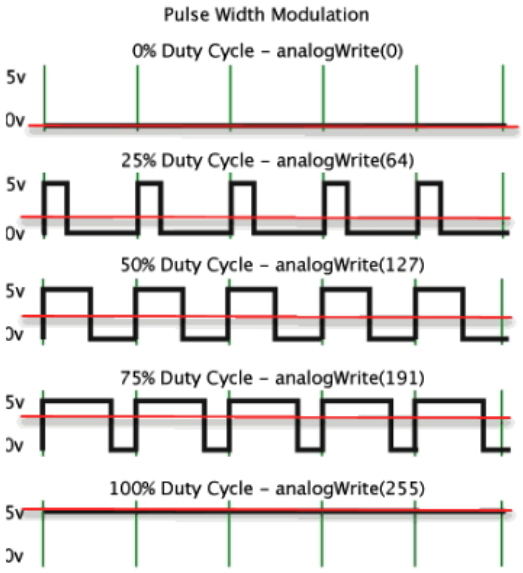


Estructura de las señales PWM en Arduino

También funcionará con una bombilla incandescente, porque tiene gran inercia, pero con un LED veremos cómo se enciende y se apaga, porque prácticamente no tiene retardo de encendido / apagado. ¿Qué hacer? Es muy simple, sube la frecuencia. En el experimento mental, tuvimos un período de 2 segundos, que es 0.5 Hz. Ahora imagine una señal de este tipo con una frecuencia de, digamos, 1000 Hz. o 25.000 Hz (25 kHz). Ahora el papel lo juega la inercia del ojo, no notará destellos a tal frecuencia, para él solo será una disminución en el brillo. ¡El problema esta resuelto!

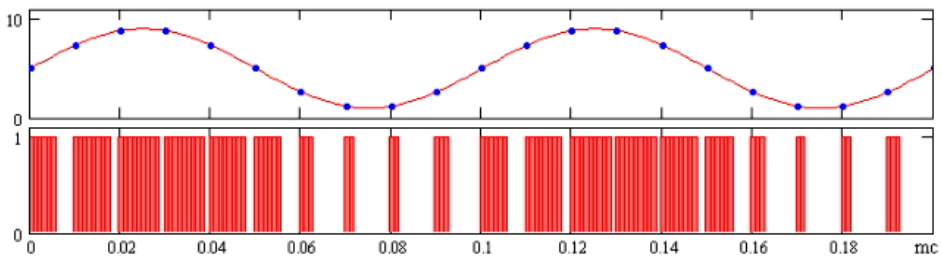
Al cambiar el llamado «llenado» de la señal PWM, es posible cambiar el voltaje «total» (integrado) durante un período determinado. Cuanto mayor sea el llenado de PWM, mayor será el voltaje, pero no más alto que el voltaje máximo «PWM».

A la relación entre el tiempo en que la señal esta en On y el tiempo que esta en Off se le llama **Duty Cycle**. También llamado relación **marca-espacio**.



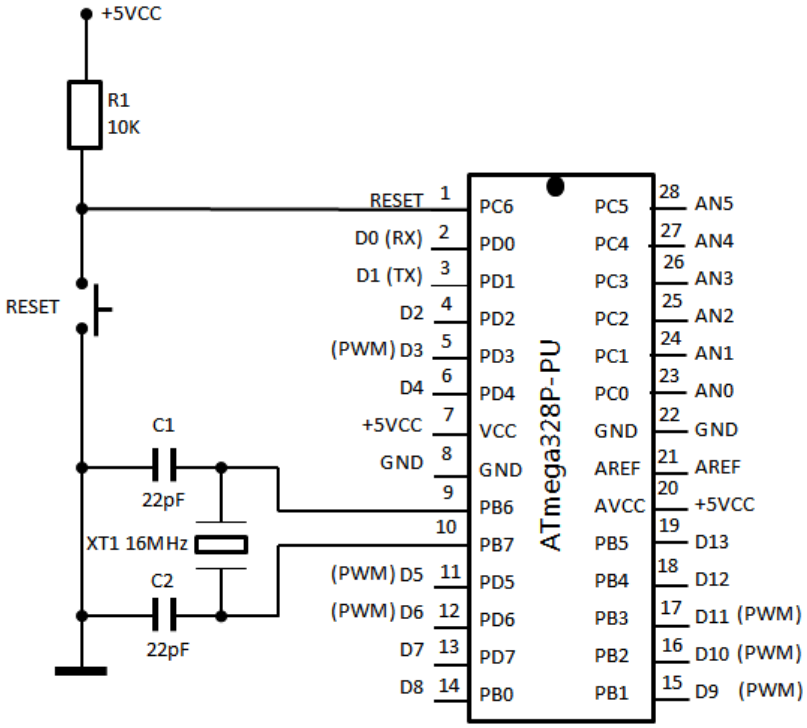

Duty cycle en señales PWM

Con una señal PWM, incluso puede modular señales analógicas complejas, como una onda sinusoidal. La siguiente imagen muestra el PWM (abajo) y el mismo PWM después de los filtros:




Modulación de onda sinusoidal

Así funcionan los inversores DC-AC. Volviendo a las propiedades de la señal PWM, solo hay dos: la frecuencia (frequency) y la relación (duty).




El microcontrolador, el más reciente de Arduino, que implementa el chip ATmega328P, permite habilitar como salidas analógicas tipo **PWM** los pines 3, 5, 6, 9, 10 y 11, y tiene los llamados contadores que cuentan los «ciclos» del generador de reloj (cuarzo). Estos contadores generan una señal **PWM**, es decir el núcleo computacional del propio microcontrolador no participa en esto. Además de los cálculos, incluso la salida de la señal de la pata del µP descansa sobre los hombros del contador. Esto es muy importante de entender, porque la señal PWM no ralentiza la ejecución del código, ya que literalmente “es otra pieza de hardware” la que está involucrada en su generación.

En las placas UNO / Nano / Pro Mini, tenemos tres contadores de temporizador, cada temporizador tiene dos salidas a los pines MC, es decir, tenemos 2 * 3 = 6 pines capaces de generar una señal PWM. Para la generación PWM, tenemos una función lista para usar **analogWrite ( pin, duty ):**

- **Pin, es un pin del temporizador**. Para Nano / Uno, estos son los pines D3, D5, D6, D9, D10, D11. En algunas placas están marcadas * con un asterisco, pero en general, para determinar los pines PWM en cualquier otro modelo de Arduino, basta con mirar el pinout.

- **Duty– Relación de la señal PWM**. Por defecto, todas las «salidas» PWM son de 8 bits, es decir, el duty puede tomar un valor con una «resolución» de 8 bits, que es 0-255.

Combinemos este conocimiento con la lección anterior e intentemos cambiar el brillo del LED conectado a través de una resistencia al pin D3. Potenciómetro conectado al pin A0.

```
void setup() {
  pinMode(3, OUTPUT);  // D3 como salida
}
void loop() {
 // PWM para pin 3 , 1023/4 = 255
  analogWrite(3, analogRead(0) / 4);
  delay(10);
}
```

El ejemplo considerado cambia el brillo del LED dependiendo de la posición del eje del potenciómetro.

Algunas palabras sobre la señal **PWM** "estándar": la obtenemos con la configuración que nos brinda la biblioteca Arduino.h, y esta configuración se subestima en gran medida en comparación con las capacidades de Arduino. Hablaremos de «mejorar» PWM más adelante, pero ahora echemos un vistazo a las características de PWM «fuera de la caja»:

| Temporizador | Pines | Frecuencia | Resolución |
|---|---|---|---|
| Temporizador 0 | D5 y D6 | 976 Hz | 8 bits (0-255) |
| Temporizador 1 | D9 y D10 | 488 Hz | 8 bits (0-255) |
| Temporizador 2 | D3 y D11 | 488 Hz | 8 bits (0-255) |

Estos son números mejorables, especialmente en términos de frecuencia. Todos los temporizadores son de talla única, para que el usuario no piense y no estudie documentación innecesaria. Volveremos a cambiar la frecuencia y el ancho del PWM en una lección separada.

## 10. GESTIÓN DEL TIEMPO

En la programación de Arduino existen algunas funciones relacionadas con el tiempo que nos permiten aplicar tiempos en la aplicación de instrucciones.

### delay(ms)

La función **delay()** (tiempo en milisegundos) se utiliza para retardar la siguiente instrucción que se va a ejecutar en un programa Arduino. Esta detiene la ejecución del programa la cantidad de tiempo en **ms** que se indica en la propia instrucción. De tal manera que 1000 equivale a 1seg.

```
delay(1000);   //  espera 1 segundo
```

La función **delay** hará esperar 1 segundo al microcontrolador antes de pasar a la siguiente instrucción. El programa se detiene durante el tiempo que indique **delay()** para continuar posteriormente con la programación.

Veamos el siguiente ejemplo utilizando el **delay()** para ver el efecto del encendido y apagado de dos leds con un tiempo de 100 milisegundos.

```
/* encendido y apagado de dos leds*/

int A;                //declaramos variable A
int led=2;           //declaramos pin digital D2
int led2=4;          //declaramos pin digital D4
void setup() {
pinMode (led, OUTPUT);     //ponemos pin 2 de salida
pinMode (led2, OUTPUT);    //ponemos pin 4 de salida
}
void loop() {
while ( A<50) {  //establecemos un tiempo para que se enciendan y apaguen
los leds
digitalWrite(led,HIGH);
delay(100);  //establecemos un tiempo para ver el efecto de encendido
digitalWrite(led, LOW);
delay(100);     //establecemos un tiempo para ver el efecto de apagado
digitalWrite(led2, HIGH);
delay(100);     //establecemos un tiempo para ver el efecto de encendido
digitalWrite(led2, LOW);
delay(100);    //establecemos un tiempo para ver el efecto de apagado
A++;            // A suma 1
}
}
```

## millis()

La función **milis()** actúa de contador en el momento que es activada, por lo que contará en milisegundos desde el momento que es activada hasta aproximadamente 50 días.

```
Valor = millis();   //valor recoge el número de milisegundos
```

Normalmente será un valor grande (dependiendo del tiempo que esté en marcha la aplicación después de cargada o después de la última vez que se pulsó el botón "reset" de la tarjeta). Este número se desbordará (si no se resetea de nuevo a cero), después de aproximadamente 9 horas.

Para que la función **millis()** devuelva el tiempo en milisegundos y los almacene en una variable, ésta debe ser del tipo **unsigned long**.

```
/* Ejemplo de la función millis con la variable unsigned long*/
unsigned long tiempo;  //variable unsigned long asignada a tiempo
void setup() {
}
void loop() {
tiempo = millis();  // la función se activa y se guarda en la variable
tiempo
}
```

## micros()

La función **micros()** también actúa de contador en el momento que es activada, por lo que contará en microsegundos desde el momento que es activada hasta unos 70 minutos.

Para que la función **micros()** devuelva el tiempo contabilizado y los almacene en una variable, ésta también debe ser del tipo **unsigned long**.

```
/* Ejemplo de la función micros*/
unsigned long tiempo;  //variable unsigned long asignada a tiempo
void setup() {
}
void loop() {
tiempo=micros();  // la función se activa y se guarda en la variable
tiempo
}
```

## delaymicroseconds()

Esta función responde de igual forma que la función **delay(),** solo que entre paréntesis introduciremos el tiempo en microsegundos y no en milisegundos.

```
 delaymicroseconds(1000);   //  espera 1 microsegundos
```

## 11. FUNCIONES DE SONIDOS

En la programación de Arduino existen dos funciones que nos permite sacar sonidos a través de las salidas analógicas A0, A1, A2, A3, A4 o A5. Estas señales consisten en pulsos que varían en el tiempo y pueden ser escuchadas por un pequeño altavoz o amplificadas. Estas dos funciones son **tone()** y **noTone()**.

### tone()

La función **tone()** nos permite generar más tonos y de una forma más fácil. Su sintaxis es la siguiente:

- **tone** (pin, frecuencia, duración); donde:
  - **pin:** aquí deberemos introducir el pin donde conectaremos el positivo del altavoz de salida con una resistencia en serie de unos 100Ω.
  - **frecuencia:** frecuencia de la señal.
  - **duración:** duración del tono. Si no se específica nada, el tono sigue sonando hasta que el programa no encuentre la función **noTone()**.

### noTone()

La función **noTone(pin)** hace "callar" el tono que se ejecuta en ese momento. Para volver a poder reproducir el tono o tonos se debe invocar la función **tone()** una vez más.

```
noTone(pin);  // pin es donde se encuentra el altavoz y lo desactiva
```

Veamos a continuación varios ejemplos, utilizando **tone()** y **noTone()** con diferentes frecuencias y duración y la desconexión dentro de un bucle temporizado y activado mediante un pulsador.

Ejemplo 1

```
/*Pulsador botón que activa un led y suena tonos*/
int piezo=A5;   // declaramos la variable piezo para puerto analógico A5
const int botonPin=2;       // declara D2 el pin botón
const int ledPin=4;          // declara D4 pin LED
int Estadoboton=0;         // declara variable para Estado boton

void setup() {
pinMode (piezo, OUTPUT);    // inicializa piezo como salida
pinMode(ledPin, OUTPUT);    // inicializa el pin LED como salida
pinMode(botonPin, INPUT);  // inicializa el pin boton como entrada:
}
void loop() {
Estadoboton = digitalRead(botonPin); //lee el estado del valor pin botón
if (Estadoboton == HIGH) {//cuando pulsa  botón si estado botón es igual
alto
digitalWrite(ledPin, HIGH);        //  enciende el led
 for (int i=100; i<=4000; i=i+50){ //hacemos un temporizador
  tone (piezo,320,100);            //activa tono de sonido
  tone (piezo,420,50);             //activa tono de sonido diferente
  delay(500);                                  // espera medio segundo
  }
  }
else {            // de lo contrario apaga el led y calla el sonido
digitalWrite(ledPin, LOW);     //apaga el led
 noTone(piezo);                // desconecta el tono
}
}
```

Ejemplo 2

```
/*emitir rango de tonos con Arduino cuando se pulsa un boton*/

int piezo=A0;       //establecemos variable piezo al puerto analógico A0
int boton=11;           //establecemos el pin digital 11 a boton
int estadopulsador;     //establecemos la variable estadopulsador
voiid setup() {
pinMode (piezo, OUTPUT);   //configuramos piezo de salida
pinMode (boton, INPUT);    // configuramos botón de entrada
}
void loop() {
estadopulsador=digitalRead(boton);  // lee boton y guárdalo en
estadopulsador
if (estadopulsador==HIGH) {    //si estado pulsador esta a nivel alto
delay(320);                    // espera 320 milisegundos
for (int i=100; i<=3000; i=i+100) { // temporiza con la variable i el
bucle siguiente
tone (piezo, 320,150);   // suena por piezo el sonido configurado en tono
delay(320);                 // espera 320 milisegundos
noTone(piezo);              //detiene el sonido }}
```

Ejemplo 3

```
/*emitir rango de tonos con Arduino cuando se pulsa un boton*/

int piezo=A0;       //establecemos variable piezo al puerto analógico A0
int boton=11;       //establecemos el pin digital 11 a boton
int estadopulsador;  //establecemos la variable estadopulsador
voiid setup() {
pinMode (piezo, OUTPUT);   //configuramos piezo de salida
pinMode (boton, INPUT);        // configuramos botón de entrada
}
void loop() {
estadopulsador=digitalRead(boton); //lee boton y guárdalo en
estadopulsador
if (estadopulsador==HIGH) {    //si estado pulsador esta a nivel alto
delay(100);                    // espera 100 milisegundos
for (int i=100; i<=4000; i=i+50) {   // temporiza con la variable i el
bucle siguiente
tone (piezo, 320,100);    // suena un tono con una frecuencia y duracion
tone (piezo, 420,50);    //suena otro tono con diferente frecuencia y
duracion
delay(100);              //espera 100 milisegundos
noTone(piezo);           //detiene el sonido
}
}
```

Ejemplo 4

```
/*emitir rango de tonos con Arduino cuando se pulsa un boton*/
int piezo=A0;      //establecemos variable piezo al puerto analógico A0
int boton=11;      //establecemos el pin digital 11 a boton
int estadopulsador;   //establecemos la variable estadopulsador
voiid setup() {
pinMode (piezo, OUTPUT);    //configuramos piezo de salida
pinMode (boton, INPUT);     // configuramos botón de entrada
}
void loop() {
estadopulsador=digitalRead(boton);  // lee boton y guárdalo en
estadopulsador
if (estadopulsador==HIGH) {   //si estado pulsador esta a nivel alto
delay(100);                   // espera 100 milisegundos
for (int i=200; i<=1800; i=i+100) { // temporiza con la variable i el
bucle siguiente
tone (piezo, i,100);   // variación de la frecuencia de tono que cambia i
delay(100);            // espera 100 milisegundos
noTone(piezo);         //desactiva el sonido
}
}
```

Ejemplo 5

```
/*emitir rango de tonos con Arduino cuando se pulsa un boton*/
int piezo=A0;   //establecemos variable piezo al puerto analógico A0
int boton=11;     //establecemos el pin digital 11 a boton
int estadopulsador;    //establecemos la variable estadopulsador
voiid setup() {
pinMode (piezo, OUTPUT);    //configuramos piezo de salida
pinMode (boton, INPUT);    // configuramos boton de entrada
}
void loop() {
estadopulsador=digitalRead(boton); // lee boton y guárdalo en
estadopulsador
if (estadopulsador==HIGH) { //si estado pulsador esta a nivel alto
delay(100);                 // espera 100 milisegundos
for (int i=200; i<=2500; i=i+100) { // temporiza con la variable i el
bucle siguiente
tone (piezo, 300,200);  // suena por piezo el sonido configurado en tono
delay(200);             // espera 200 milisegundos
noTone(piezo);          //desactiva el sonido
tone(piezo, 500,150);   //suena otro tono que diferente frecuencia y
duracion
delay(100);             //espera 100 milisegundos
}
}
```

## 12. INTERRUPCIONES DE HARDWARE

Una **interrupción** la podemos definir como una llamada al microcontrolador, el cual, deja lo que está ejecutando y atiente dicha llamada. Esta llamada o interrupción, normalmente, "lleva" al microcontrolador a otra parte del código que debe ejecutarse con mayor prioridad.

Una vez ejecutado esa parte del código, el microcontrolador vuelve al punto anterior, es decir, a la línea de la instrucción donde lo había dejado antes de recibir la interrupción.

**Arduino UNO** posee dos pines, el pin **2** y **3**, para crear interrupciones. En este caso, serán interrupciones externas, ya que las vamos a generar nosotros desde fuera del microcontrolador.

A este tipo de interrupciones también las podemos llamar **interrupciones de hardware**.

Para desarrollar la interrupción con Arduino, tenemos una función que genera la interrupción deseada. Esta función es **attachInterrupt()** y tiene la siguiente sintaxis:

```
attachInterrupt (nº int, nombre función, modo); //interrupccion hardware
```

Veamos cada uno de los parámetros de la función:

- **Nº int:** número de la interrupción. Si utilizamos la interrupción 0, el pin a utilizar será el 2. Si utilizamos la interrupción 1, el pin será el 3.
- **Nombre función:** nombre que le damos a la función interrupción que deseamos llamar.
- **Modo:** Define cuando la interrupción deberá ser activada. Observamos 4 formas de activación:
    1. **CHANGE:** Se activa cuando el valor en el pin pasa de HIGH a LOW o de **LOW a HIGH.** Es el modo normalmente empleado.
    2. **LOW:** Se activa cuando el valor en el pin es LOW.
    3. **RISING:** Se activa cuando el valor del pin pasa de LOW a HIGH.
    4. **FALLING:** se activa cuando el valor del pin pasa de HIGH a LOW.

Antes de emplear un ejemplo con las interrupciones, veamos un código donde se introduce un bucle de retardo para simular la perdida de información cuando Arduino está ocupado.

Aquí el botón no responderá a nuestra petición de encendido con solo pulsar, y se deberá insistir para que nos obedezca.

Código de encendido de un led sin interrupción:

```
/*Retardo del encendido de un led mediante botón*/

int botón=7;    //variable que declaramos el pin digital D7
int led=13;     // variable que declaramos el pin digital D13
int vb=0;       //variable que almacena el valor del boton
void setup() {
pinMode (botón, INPUT);
pinMode (led, OUTPUT);
Serial.begin (9600);
}
void loop() {
for(int t=0; t<500; t++) {   //bucle de retardo
Serial.println (t);
}
vb=digitalRead (botón);
if (vb==HIGH) {
digitalWrite(led, HIGH);
delay (220);
}
else {
digiralWrite (led, LOW);
delay(220);
}
}
```

Veamos el código donde se emplea una interrupción. Como se podrá comprobar, utilizando interrupciones, el botón es atendido por Arduino con la mayor prioridad posible, encendiendo el diodo led al pulsar dicho botón y obviando del bucle de retardo.

```
/*Encendido de un led mediante una interrupcion*/
int botón=2;  //variable que declaramos el pin digital D2
int led=13;    // variable que declaramos el pin digital D13
int vb=0;       //variable que almacena el valor del boton
void setup() {
pinMode (botón, INPUT);
pinMode (led, OUTPUT);
Serial.begin (9600);
attachInterrupt (0, parpadeo, HIGH);  // interrupción en el pin 2 de
entrada a nivel alto
}
void loop() {
for(int t=0; t<500; t++) {   //bucle de retardo
Serial.println (t);
}
}
```

Ampliemos un poco más los conceptos de interrupciones de hardware. Éstas son diferentes, es decir, no las interrupciones en sí mismas, sino sus razones: una interrupción puede ser activada por un convertidor de analógico a digital, un temporizador-contador o, literalmente, un pin de microcontrolador. Estas interrupciones se denominan interrupciones externas de hardware.

La interrupción de hardware externa es una interrupción causada por un cambio en el voltaje en un pin del microcontrolador. El punto principal es que el microcontrolador (núcleo de computación) no sondea el pin y no pierde tiempo en él, otra pieza de hardware está conectada en el pin. Tan pronto como cambia el voltaje en el pin (lo que significa que se aplica una señal digital, se aplica +5 / se elimina + 5), el microcontrolador recibe una señal, deja todo, procesa la interrupción y vuelve a funcionar. ¿Por qué es necesario? La mayoría de las veces, las interrupciones se utilizan para detectar eventos cortos: pulsos o incluso para contar su número sin cargar el código principal. Una interrupción de hardware puede detectar una pulsación breve de un botón o un disparador de sensor durante cálculos largos y complejos o retrasos en el código, es decir, en términos generales, el pin se consulta en paralelo con el código principal. Además, las interrupciones pueden despertar al microcontrolador de los modos de ahorro de energía, cuando en general casi todos los periféricos están apagados. Veamos cómo trabajar con interrupciones de hardware en el IDE de Arduino.

Comencemos con el hecho de que no todos los pines pueden «interrumpir». Ahora debemos entender que las interrupciones de hardware solo las pueden generar ciertos pines:

| Mc / número de interrupción | INT 0 | INT 1 | INT 2 | INT 3 | INT 4 | INT 5 |
|---|---|---|---|---|---|---|
| ATmega 328/168 (Nano, UNO, Mini) | D2 | D3 | – | – | – | – |
| ATmega 32U4 (Leonardo, Micro) | D3 | D2 | D0 | D1 | D7 | – |
| ATmega 2560 (Mega) | D2 | D3 | D21 | D20 | D19 | D18 |

Pines de interrupción de Arduino.

Como se vió en la tabla, las interrupciones tienen su propio número, que difiere del número de pin. Por cierto, hay una función relacionada, **digitalPinToInterrupt ( pin )** que toma un número pin y devuelve el número de interrupción. Habiendo alimentado esta función con el número 3 en el Arduino nano, obtenemos 1.

La interrupción también se puede desactivar mediante la función **detachInterrupt ( pin )**, donde pin es nuevamente el número de interrupción de arduino.

También puede deshabilitar globalmente las interrupciones con la función **noInterrupts ()** y resolverlos de nuevo con **interrupts()**. ¡Cuidado con ellos! noInterrupts () también detendrá las interrupciones de los temporizadores, y todas las funciones de tiempo y la generación de PWM se «detendrán».

Veamos un ejemplo en el que las pulsaciones de botones se cuentan en la interrupción y, en el bucle principal, se emiten con un retraso de 1 segundo. Trabajando con el botón en el modo habitual, es imposible combinar una salida tan aproximada con un delay:

```
volatile int counter = 0;  //contador variable
void setup() {
  Serial.begin(9600);   // abre un puerto para la comunicación
 // conectó el botón a D2 y GND
  pinMode(2, INPUT_PULLUP);
  // D2 es interrupción 0
  // controlador - función buttonTick
  // CAYENDO - cuando se presiona el botón, habrá una señal de 0, y la
capturamos
  attachInterrupt(0, buttonTick, FALLING);
}
void buttonTick() {
  counter++;  // + clic
}
void loop() {
  Serial.println(counter);  // salida
  delay(1000);              // espera
}
```

## 13. ALEATORIEDAD

En la programación de Arduino no iba a faltar la explicación de dos instrucciones utilizadas en la aleatoriedad de un programa de Arduino. Estas son: **randomSeed()** y **random().**

### randomSeed (seed)

Para utilizar la instrucción **random()**, primero debemos emplear la instrucción **randomSeed().**

Esta instrucción nos permite "inicializar", a partir de una variable o de otra función, una semilla para generar números aleatorios.

La sintaxis será:

- **RandomSeed** (valor). Establece un valor, o semilla, como punto de partida para la función **random()**.

Por ejemplo, **randomSeed(millis)** genera números aleatorios a partir del valor de la función **millis**. Recordemos que esta función devuelve en milisegundos el tiempo desde que Arduino está ejecutando el programa actual. Después de 50 días de funcionamiento, se produce un desbordamiento y vuelve a cero.

```
randomSeed(valor);  //  hace que valor sea la semilla del random
```

Debido a que Arduino es incapaz de crear un verdadero número aleatorio, **randomSeed** le permite colocar una variable, constante, u otra función de control dentro de la función **random**, lo que permite generar números aleatorios "al azar". Hay una variedad de semillas, o funciones, que pueden ser utilizados en esta función, incluido **millis()** o incluso **analogRead()** que permite leer ruido eléctrico a través de un pin analógico.

### random(max) random (min,max)

La función **random** (aleatorio) genera números aleatorios en un rango de 0 a un máximo, o un rango preestablecido por el programador con las variables **min** y **max**, que devuelve un número aleatorio entero de un intervalo de valores especificado entre los valores **mínimo** y **máximo**.

```
valor = random(100, 200);  //asigna a la variable 'valor'  un numero
aleatorio  comprendido entre 100 y 200
```

**Nota:** Esta función debe utilizarse después de usar el **randomSeed().**

El siguiente ejemplo genera un valor aleatorio entre 0 y 255 y lo envía a una salida analógica PWM :

```
int  randNumber;      //  variable que almacena el valor aleatorio
int  led =10;         //  define el pin digital D10
void  setup()  {}     //  no  es necesario configurar nada
void  loop() {
randomSeed(millis()); //  genera una semilla para aleatorio a partir de
la función  millis()
randNumber = random(255);  //  genera número aleatorio entre  0-255
analogWrite(led, randNumber);  //  envía a la salida led de tipo PWM el
valor
delay(500);               // espera medio segundo
}
```

Otro ejemplo:

```
/*Aplicando aleatoriedad a un led*/

int max= 255;  // declaramos una variable tipo entero max y asignamos
valor hasta 255
void setup() {
pinMode (5, OUTPUT); //declaramos pin 5 de salida
}
void loop() {
randomSeed (millis()); //creamos una semilla  para aplicar aleatoriedad
mediante millis()
analogWrite (5, random(max)); //Activamos pin 5 con valores aleatorios
dado por random()
delay(50);     //los valores tomados se mantienen durante 50 milisegundos
}
```

## 14. USANDO LAS BIBLIOTECAS

Las bibliotecas son una herramienta muy poderosa cuando se trabaja con Arduino, especialmente para principiantes. Una biblioteca es un archivo (un conjunto de archivos) que contiene exactamente el mismo código C ++ en el que escribimos un sketch (a veces también hay inserciones de ensamblador). Podemos conectar la biblioteca a nuestro código y utilizar las capacidades que proporciona, y hay muchas opciones: “herramientas” listas para usar y para trabajar con sensores y módulos externos, para trabajar con los periféricos internos del microcontrolador (temporizadores, ADC, memoria), bibliotecas de diversas herramientas, matemáticas y mucho más.

Lo bueno de trabajar con una biblioteca es que no necesitamos saber cómo funciona el código que contiene, utilizamos herramientas listas para usar proporcionadas por el desarrollador de la biblioteca. Muy a menudo hay descripciones / documentación y ejemplos de uso de bibliotecas.

La lista está compilada con bibliotecas útiles para UNO, NANO, MEGA, es decir, no hay bibliotecas potentes para placas similares DUE y ZERO. Fuentes:

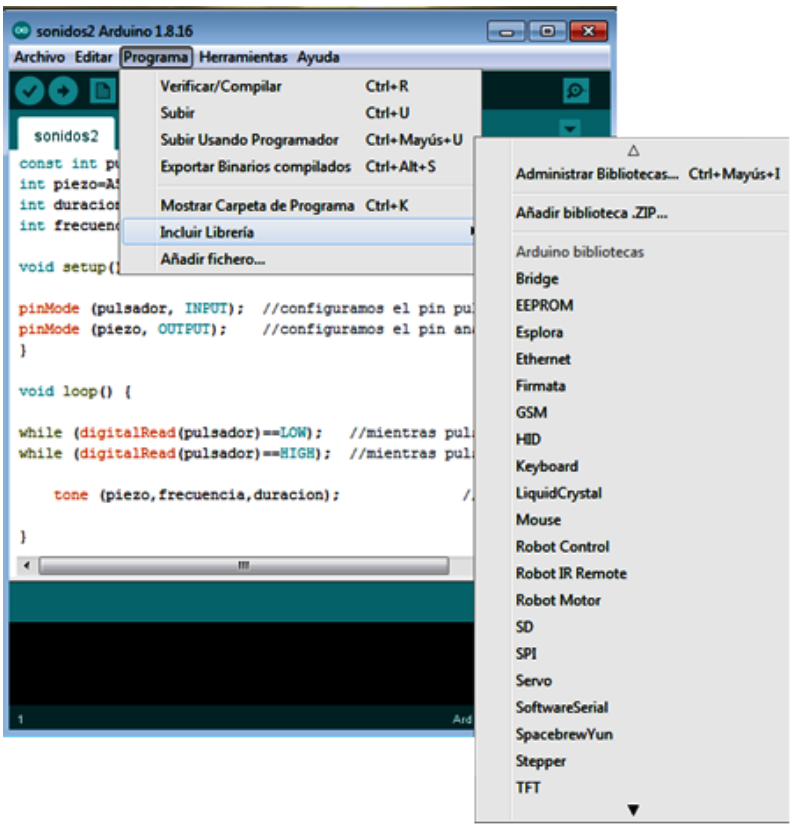
https://www.arduino.cc/en/reference/libraries

https://github.com/Lembed/Awesome-arduino/blob/master/README.md

https://www.arduinolibraries.info/architectures/avr

### Asistente de bibliotecas

Como se ha comentado anteriormente una biblioteca es una colección de archivos de texto con código. La biblioteca se puede instalar de dos formas: desde el repositorio oficial o manualmente. Arduino tiene varias bibliotecas que se pueden obtener directamente del programa Arduino IDE utilizando el administrador de bibliotecas incorporado, que le permite instalar, eliminar y actualizar bibliotecas. Esto es para iniciarse, porque esta lista no incluye todas las bibliotecas existentes y el administrador no proporciona una descripción normal. Para instalar una biblioteca del repositorio oficial de Arduino, vaya a **Programa / Incluir librería** … Se abrirá el administrador de la biblioteca, en el que puede encontrar e instalar una biblioteca de la lista con un solo clic.




**Incluir librería en el IDE de Arduino**

## Dentro de la Biblioteca

La biblioteca, dependiendo de la cantidad de código y el estado de ánimo del programador, se puede diseñar de forma muy compacta y detallada, con un montón de archivos y carpetas adicionales. Consideremos la composición clásica de la biblioteca. Para mayor comodidad, recomiendo habilitar la visibilidad de las extensiones de archivo . Todos los siguientes archivos de muestra son archivos de texto ordinarios, puede abrirlos con un bloc de notas normal. También recomiendo usar el «bloc de notas de programador» – Notepad ++ ( enlace al sitio oficial ), que resalta la sintaxis y, en general, es una herramienta muy adecuada para el desarrollador.

El **<nombre de biblioteca>.h:** es un archivo de encabezado, el archivo de biblioteca más importante. Es tan importante que la biblioteca solo puede estar formada por él. Por lo general, se encuentra en la raíz de la biblioteca o en la carpeta src, source (fuente). Este archivo generalmente enumera todas las clases / métodos / funciones / tipos de datos, contiene información sobre la biblioteca, a menudo hay una descripción extendida para cada método o función. Muy a menudo, el archivo de encabezado principal es la mini documentación de la biblioteca.

Una biblioteca puede tener una estructura de varios archivos con una gran cantidad de archivos de encabezado, pero el archivo de encabezado principal siempre es uno, tiene el mismo nombre que la carpeta con la biblioteca.

Un archivo con la extensión **.cpp** es un archivo de implementación que contiene el código ejecutable principal del programa. Por lo general, va en parejo a su archivo **.h** de encabezado , es decir, **<nombre de la biblioteca> .cpp** .

**keywords.txt:** un archivo que enumera los nombres de funciones, métodos y otros nombres de trabajo de la biblioteca resaltados en el IDE de Arduino (resaltados en un color diferente) en el código.

Archivo **library.properties:** archivo que contiene información sobre la biblioteca para los desarrolladores y el administrador de la biblioteca (nombre, versión, autor, categoría, etc.). Carpeta **Src:** esta carpeta puede contener los archivos de la biblioteca principal ( .h , .cpp , .c ).

**Nota:** La carpeta de ejemplos es una carpeta con ejemplos de uso de la biblioteca. Además de los archivos y carpetas enumerados, la carpeta con la biblioteca puede contener otros archivos y carpetas de servicio, a veces incluso puede encontrar documentación completa en forma de archivos de texto o páginas html.

## ¿Cómo trabajar con la biblioteca?

Digamos que compró algún tipo de módulo o sensor, buscó información en Google sobre él, encontró un artículo con un ejemplo. Los ejemplos suelen ser sencillos, muestran cómo se conecta y funciona. Descargamos la biblioteca del artículo, lo probamos, todo funciona. ¿Que sigue? A continuación, debe abrir la carpeta con la biblioteca y ver los ejemplos oficiales, averiguar cómo funcionan y qué pueden hacer. Los ejemplos se encuentran en la carpeta de ejemplos en la carpeta de la biblioteca.

Los ejemplos generalmente no revelan todas las capacidades de la biblioteca, por lo que abrimos y leemos el archivo de encabezado, que es library_name.h. Contiene literalmente una lista de herramientas de biblioteca, a menudo con descripciones para cada una. Con esta información, puede extraer del módulo todas las funciones que el desarrollador de la biblioteca le ha prescrito. Echemos un vistazo a la biblioteca de servo.h, creo que la mayoría ha trabajado con ella.

Veamos los ejemplos que están en la carpeta de la biblioteca: **knob.ino**

```
/* Controlar la posición de un servo usando un potenciómetro (resistencia
variable) por Michal Rinott <http://people.interaction-ivrea.it/m.rinott>
modificado el 8 de noviembre de 2013 por Scott Fitzgerald
 http://www.arduino.cc/en/Tutorial/Knob */

#include <Servo.h>
Servo myservo;  // crea un objeto servo para controlar un servo
int potpin = 0; // pin analógico utilizado para conectar el potenciómetro
int val;        // variable para leer el valor del pin analógico
void setup() {
  myservo.attach(9); // conecta el servo en el pin 9 al objeto servo
}
void loop() {
  val=analogRead(potpin);  // lee el valor del potenciómetro (valor entre
0 y 1023)
  val=map(val,0,1023,0,180); // escalarlo para usarlo con el servo (valor
entre 0 y 180)
  myservo.write(val); // establece la posición del servo de acuerdo con
el valor escalado
  delay(15);          // espera a que llegue el servo
}
```

```
/ * sweep
 por BARRAGAN <http://barraganstudio.com>
 Este código de ejemplo es de dominio público.
 modificado el 8 de noviembre de 2013
 por Scott Fitzgerald
 http://www.arduino.cc/en/Tutorial/Sweep
* /
#include <Servo.h>
Servo myservo; // crea un objeto servo para controlar un servo
// Se pueden crear doce objetos servo en la mayoría de las placas
int pos = 0;   // variable para almacenar la posición del servo
void setup() {
  myservo.attach(9); // conecta el servo en el pin 9 al objeto servo
}
void loop() {
  for (pos=0; pos<=180; pos+=1 ) { // va de 0 grados a 180 grados
    // en pasos de 1 grado
    myservo.write(pos);  // decirle al servo que vaya a la posición en la
variable 'pos'
    delay(15);      // espera 15ms a que el servo alcance la posición
  }
  for (pos=180; pos>=0; pos-=1) { // va de 180 grados a 0 grados
     myservo.write(pos);    // decirle al servo que vaya a la posición en
la variable 'pos'
    delay(15);       // espera 15ms a que el servo alcance la posición
}
}
```

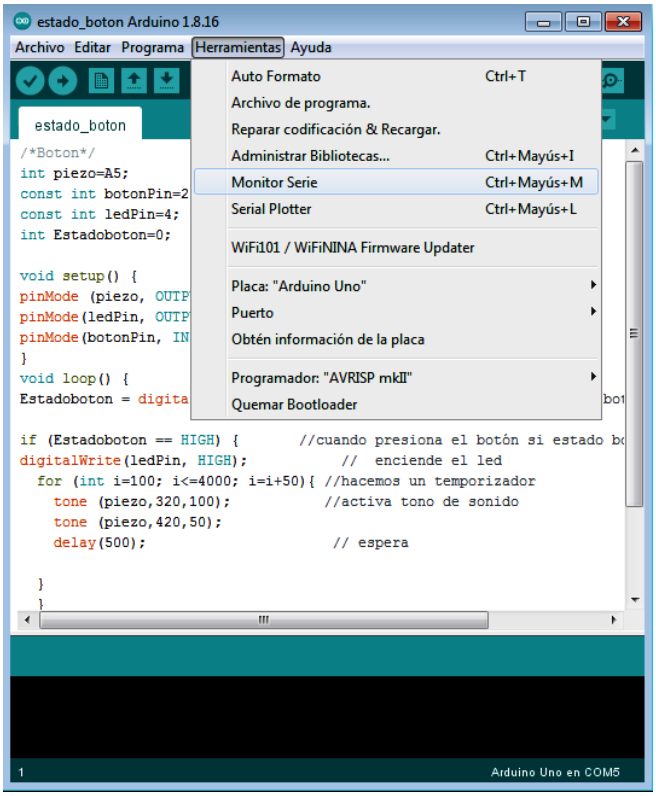
Abramos ahora el archivo de encabezado **Servo.h**, que se encuentra en la carpeta.

```
#define Servo_VERSION 2 // versión de software de esta biblioteca
#define MIN_PULSE_WIDTH 544 // el pulso más corto enviado a un servo
#define MAX_PULSE_WIDTH 2400 // el pulso más largo enviado a un servo
#define DEFAULT_PULSE_WIDTH 1500 // ancho de pulso predeterminado cuando
el servo está conectado
#define REFRESH_INTERVAL 20000 // tiempo mínimo para actualizar los
servos en microsegundos
#define SERVOS_PER_TIMER 12 // el número máximo de servos controlados por
un temporizador
#define MAX_SERVOS (_Nbr_16timers * SERVOS_PER_TIMER)
#define INVALID_SERVO 255 // bandera que indica un índice de servo no
válido
#if! !defined(ARDUINO_ARCH_STM32F4)
typedef struct {
  uint8_t nbr:6;         // un número de pin de 0 a 63
  uint8_t isActive:1;    // verdadero si este canal está habilitado, pin
no pulsado si es falso
} ServoPin_t;
typedef struct {
  ServoPin_t Pin;
  volatile unsigned int ticks;
} servo_t;
class Servo
{
public:
  Servo ();
  uint8_t attach(pin int);  // adjunta el pin dado al siguiente canal
libre, establece pinMode, devuelve el número de canal o 0 si falla
  uint8_t attach (int pin,int min,int max); // como arriba pero también
establece valores mínimos y máximos para escrituras.
  detach vacío();
  void write(int value);    // si el valor es <200, se trata como un
ángulo, de lo contrario como ancho de pulso en microsegundos
  void writeMicroseconds(int value); // Escribe el ancho del pulso en
microsegundos
  int read();    // devuelve el ancho de pulso actual como un ángulo
entre 0 y 180 grados
  int readMicroseconds();   // devuelve el ancho de pulso actual en
microsegundos para este servo (fue read_us () en la primera versión)
  bool attached();          // devuelve verdadero si este servo está
adjunto, de lo contrario falso
private :
  uint8_t servoIndex;     // indexa los datos del canal para este servo
  int8_t min;             // mínimo es este valor multiplicado por 4
añadido a MIN_PULSE_WIDTH
  int8_t max;             // máximo es este valor multiplicado por 4
sumado a MAX_PULSE_WIDTH
}
```

## 15. VISUALIZACION DE VARIABLES POR EL MONITOR SERIE

Un aspecto muy importante a la hora de probar el buen funcionamiento de un programa es saber si las variables que se han introducido cumplen con las expectativas establecidas o bien están almacenando los datos que les hemos propuesto.

Para dar solución a esta cuestión, la **IDE** de Arduino permite ejecutar la ventana **monitor serie**, donde se muestran los datos que se están tratando durante la ejecución de un programa en Arduino.

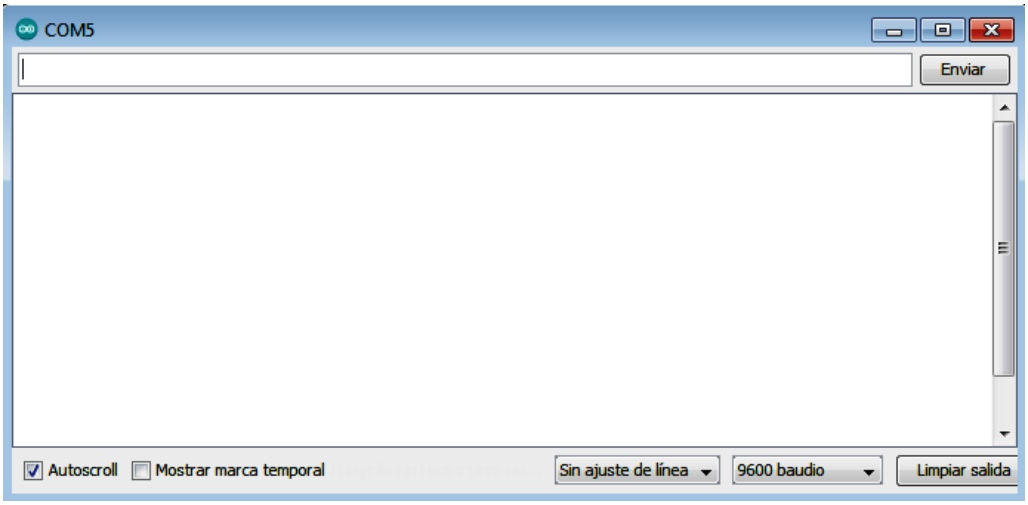



Esto es posible gracias a la conexión **USB** entre el **PC** y la placa de Arduino. Para activar el monitor serie debemos tener conectada la placa de Arduino para establecer la comunicación.

Las placas Arduino tienen un convertidor USB-TTL que permite que el microcontrolador se comunique con el PC a través de la interfaz serie, Serial, en el modo de texto “consola”. Se crea un puerto COM virtual en la computadora, al cual puede conectarse usando los programas de terminal de puerto y recibir y enviar datos de texto. El firmware se carga a través del mismo puerto. El soporte en serie está integrado en el microcontrolador en el nivel de «hardware», y el convertidor USB-TTL está conectado a estos pines del microcontrolador. En la placa Arduino, esto es, por cierto, los pines D0 y D1.

Es por eso que los pines D0 y D1 en las placas Arduino Nano / Arduino Uno no pueden ser ocupados por sensores o conectados a la alimentación / tierra en el momento de la carga del programa: el microcontrolador no podrá recibir datos y recibirá un error de arranque.

El propio Arduino **IDE** también tiene una «consola» incorporada: un monitor de puerto, un botón con un icono de lupa en la esquina superior derecha del programa. Al hacer clic en este botón, abriremos el propio monitor del puerto, en el que habrá configuraciones:




Todo está bastante claro **enviar**, el **desplazamiento automático**, y la **marca temporal,** entonces consideraremos con más detalle las siguientes opciones: **Nueva línea**, la **velocidad** y **limpiar salida**:

- **Nueva línea**: hay varias opciones para elegir. Es mejor no poner fin a la línea, ya que esto evitará errores incomprensibles en las primeras etapas del conocimiento de Arduino:
  - Sin final de línea: no hay caracteres adicionales al final de los caracteres ingresados después de presionar el botón enviar.
  - NL – carácter de avance de línea al final de los datos enviados.
  - CR – carácter de retorno de carro al final de los datos enviados.
  - NL + CR – ambos
- **Velocidad:** aquí se nos ofrece una lista completa de velocidades para elegir. La comunicación vía Serial puede realizarse a diferentes velocidades, medidas en baudios ( baudios ), y si las velocidades de recepción y envío no coinciden, los datos se recibirán incorrectamente. La velocidad predeterminada es **9600 baudio** , así que la dejaremos así.
- **Limpiar Salida**: aquí todo está claro, borra la salida.

Comencemos a conocer una de las cosas más útiles de un desarrollador de Arduino: Serial.

Serial es un objeto de la clase Stream, que le permite simplemente recibir / enviar datos a través de un puerto serie, y hereda un montón de características y trucos interesantes de la clase Stream.

| Serial.begin ( **velocidad** ) |
| --- |
| Inicia la comunicación a través de **serie** a<br>la **velocidad** ( **velocidad** en baudios, bits por segundo). Se puede configurar cualquier velocidad, pero hay varias «estándar». Lista de velocidades en baudios para el monitor de puerto Arduino IDE:<br>300<br>1200<br>2400<br>4800<br>**9600 es el** más utilizado, se puede llamar estándar TTL<br>19200<br>38400<br>57600<br>**115200** también **es** común<br>230400<br>250.000<br>500.000<br>1,000,000<br>2,000,000 |

| Serial.end () |
| --- |
| Detiene la comunicación serie. Para UNO / NANO (ATmega328), esto permite liberar los pines 0 y 1 para fines normales (lectura / escritura). |

| Serial.available() |
| --- |
| Devuelve el número de bytes almacenados en el búfer (el **tamaño del búfer es de 64 bytes** ) y están disponible para lectura. |

| Serial.availableForWrite () |
| --- |
| Devuelve el número de bytes que se pueden escribir en el búfer del puerto serie sin bloquear la función de escritura. |

| Serial.write ( **val** ), Serial.write ( **buf** , **len** ) |
| --- |
| Envía un valor numérico o una cadena al puerto **val**, o envía **len** bytes desde **buf** . ¡Importante! Envía datos en bytes (ver tabla **ASCII** ), es decir, al enviar 88 recibirás la letra X: Serial.write (88); imprimirá la letra X |

| Serial.print ( **val** ), Serial.print ( **val** , **format** ) |
| --- |
| Envía el valor **val** al puerto: un número o una cadena. A diferencia de la anterior, genera exactamente caracteres, es decir enviando 88 obtienes 88: Serial.print (88); generará 88. Además, el método print / println tiene varias configuraciones para diferentes datos, lo que lo convierte en una herramienta de depuración muy acertada |

| Serial.println (), Serial.println ( **val** ), Serial.println ( **val** , **format** ) |
| --- |
| Completo análogo de print (), pero traduce automáticamente la cadena después de la salida. También permite ser llamado sin argumentos (con corchetes vacíos) solo para saltos de línea. |

| Serial.flush () |
| --- |
| Esperando el final de la transferencia de datos. |

| Serial.peek () |
| --- |
| Devuelve el byte actual al inicio del búfer sin eliminarlo del búfer. Al llamar a Serial.read (), se leerá el mismo byte, pero ya se eliminará del búfer. |

| Serial.read () |
| --- |
| Lee y devuelve un byte como código de carácter de la tabla ASCII. Si necesita devolver un dígito, haga Serial.read () – '0'; |

| Serial.setTimeout ( **t**ime ) |
| --- |
| Establece el **tiempo de** espera (milisegundos) para esperar las siguientes funciones. *El valor predeterminado es 1000 ms (1 segundo)* |

| Serial.find ( **destino** ), Serial.find ( **destino** , **longitud** ) |
| --- |
| Lee datos del búfer y busca el conjunto de caracteres de **destino** (tipo char), opcionalmente puede especificar la **longitud**. Devuelve verdadero si encuentra los caracteres especificados. |

| Serial.findUntil(**target**, **terminal**) |
| --- |
| Lee datos del búfer y busca el juego de caracteres de **destino** (tipo char) o el terminal de cadena **terminal**. Espera el final de la transmisión por **tiempo** de **espera**, o completa la recepción después de leer el **terminal**. |

| Serial.readBytes ( **búfer** , **length**) |
| --- |
| Lee los datos del puerto y los carga en el **búfer** (char array [] o byte []), también se indica el número de bytes a leer – **length** (para no desbordar el búfer). |

| Serial.readBytesUntil ( **carácter** , **búfer** , **longitud** ) |
| --- |
| Lee datos del puerto y los carga en el **búfer** (char array [] o byte []), también se indica el número de bytes a leer – **longitud** (para no desbordar el búfer) y el **carácter** terminal. El final de la recepción en el **búfer** ocurre cuando **se** alcanza la cantidad de **longitud** especificada, cuando **se** recibe el **carácter** terminal (no entra en el búfer) o por **tiempo** de **espera**. |

| Serial.readString () |
| --- |
| Lee el búfer del puerto y forma una cadena a partir de los datos, que devuelve. |

| Serial.readStringUntil ( **terminator**) |
| --- |
| Lee el búfer del puerto y forma una cadena a partir de los datos, que devuelve. **Sale** en el **tiempo** de **espera** o al recibir un carácter de **terminación** (carácter char) |

| Serial.parseInt (), Serial.parseInt ( **skipChar** ) |
| --- |
| Lee un valor entero del búfer del puerto y lo devuelve (tipo long). También puede especificar por separado el **skipChar** que se omitirá. |

| Serial.parseFloat () |
| --- |
| Lee un valor de coma flotante del búfer del puerto y lo devuelve. |

Veamos algunos ejemplos de instrucciones para visualizar el valor de las variables que se deben introducir en el programa de forma que Arduino sepa que debe mostrar estos valores por la pantalla del monitor serie.

### Serial.begin (rate)

En ésta instrucción abre el puerto serie y fija la velocidad en baudios para la transmisión de datos en serie. El valor típico de velocidad para comunicarse con el ordenador es de **9600 baudios**, aunque otras velocidades pueden ser soportadas. Esta instrucción debe colocarse dentro del **void setup()**.

```
void  setup()
{
Serial.begin(9600);  //abre el Puerto serie  configurando la velocidad en
9600 baudio
}
```

Esta instrucción le hará saber al PC que Arduino activará la comunicación con él mediante el puerto serie a 9.600 bps y comenzará a transmitir y recibir datos.

Una vez hecho esto, será el cuerpo del programa, es decir, en el **void loop()**, donde se deberá especificar qué variable deseamos que se imprima por pantalla.

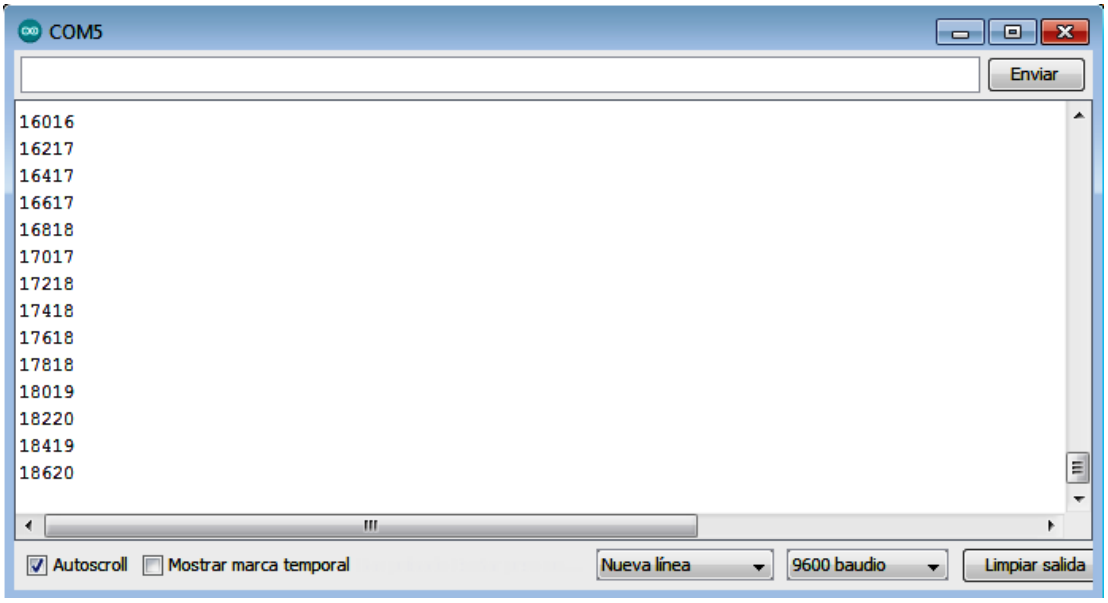
```
void loop() {
Serial.println (contador); // visualiza por el monitor serie los datos de contador
}
```

La instrucción específica que se escriba por el puerto serie el valor o los valores de la variable **contador**. La parte donde se escribe **println** visualizará el valor o valores de la variable y realizará un cambio de línea; de esta forma, los valores de la variable **contador** no aparecerán horizontalmente, sino que podremos observar cómo van apareciendo por el monitor serie verticalmente hacia abajo. Por lo que se puede apreciar que los datos van apareciendo muy rápidos, para ello, introducimos un **delay()** en la programación para dar un tiempo en cada lectura.

**Nota:** Cuando se utiliza la comunicación serie los pins digital **0 (RX)** y **1 (TX)** no puede utilizarse al mismo tiempo.

Veamos el siguiente código donde introducimos estas instrucciones para visualizar el resultado de la función millis().

```
/* Ejemplo función millis y monitor serie*/
unsigned long tiempo;     //declaramos la variable unsigned long
void setup() {
Serial.begin (9600);      //activa el monitor serie
}
void loop() {
Tiempo=millis(); // la función se activa y se guarda en la variable tiempo
Serial.println(tiempo); //visualiza por el monito serie los datos de tiempo
delay(250);    // damos un tiempo para que los datos no vayan deprisa
}
```




Hay que tener en cuenta que después de la instrucción **Serial.println(tiempo);** se incorpora un **delay (250);** haciendo que los datos no pasen tan rápido, evitando que no tengamos tiempo de verlos con claridad. Lógicamente, el valor de la instrucción **delay** se puede modificar para que los datos pasen más rápido o más lento, según la necesidad del programador.

## Serial.println(data)

Ésta otra instrucción visualiza los datos en el puerto serie, seguido por un retorno de carro automático y salto de línea. Este comando toma la misma forma que **Serial.print ()**, pero es más fácil para la lectura de los datos en el Monitor Serie de Arduino.

```
Serial.println(analogValue);  // envía el valor analogValue al puerto
```

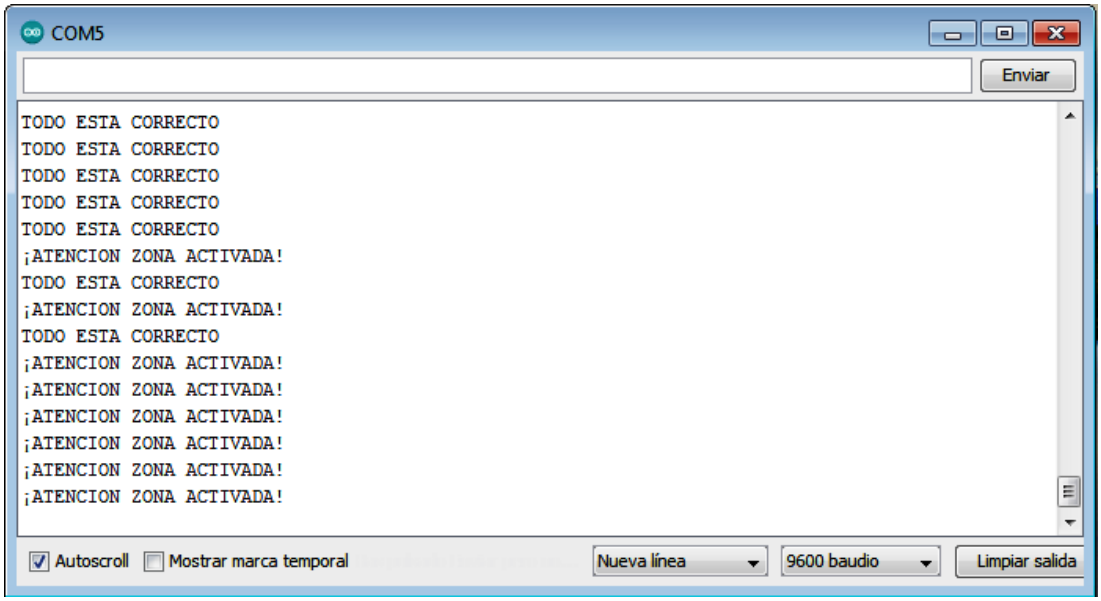
El siguiente ejemplo toma de una lectura analógica pin0 y envía estos datos al ordenador cada 1 segundo.

```
void  setup()
{
Serial.begin(9600);  // configura el puerto serie a 9600 baudios
}
void  loop()
{
Serial.println(analogRead(0)); //  envía valor analógico
delay(1000);                   //  espera 1 segundo para ver
}
```

También es posible escribir  texto para que aparezca por el monitor serie, lo que permite adornar los datos que muestra Arduino.

```
/* Ejemplo de visualización de texto mediante el monitor serie*/
int pin=13;      //declaramos la variable pin digital 13
int alarma;      // declaramos variable alarma
void setup() {
pinMode (pin, INPUT);   //configuramos el pin 13 de entrada
Serial.begin(9600);    //configuramos el monitor serie a 9600 baudios
}
void loop() {
alarma=digitalRead(pin);  // leemos el pin 13 de entrada y lo almacenamos en alarma
if (alarma==HIGH) {      // si alarma se encuentra a nivel alto
Serial.println("¡ATENCION ZONA ACTIVADA!"); //visualiza por el monitor alarma activada
delay(500); // se detiene 0,5 segundos
}
else {  //de lo contrario
Serial.println("TODO ESTA CORRECTO"); //visualiza por monitor serie que esta bien
delay(500);  //se detiene 0,5 segundos
} }
```

Si la entrada al pin 13 se encuentra a nivel bajo aparece en el monitor serie TODO ESTA CORRECTO, si en cambio se encuentra a nivel alto se muestra ¡ATENCION ZONA ACTIVADA!. En el monitor serie aparecería de la siguiente forma:



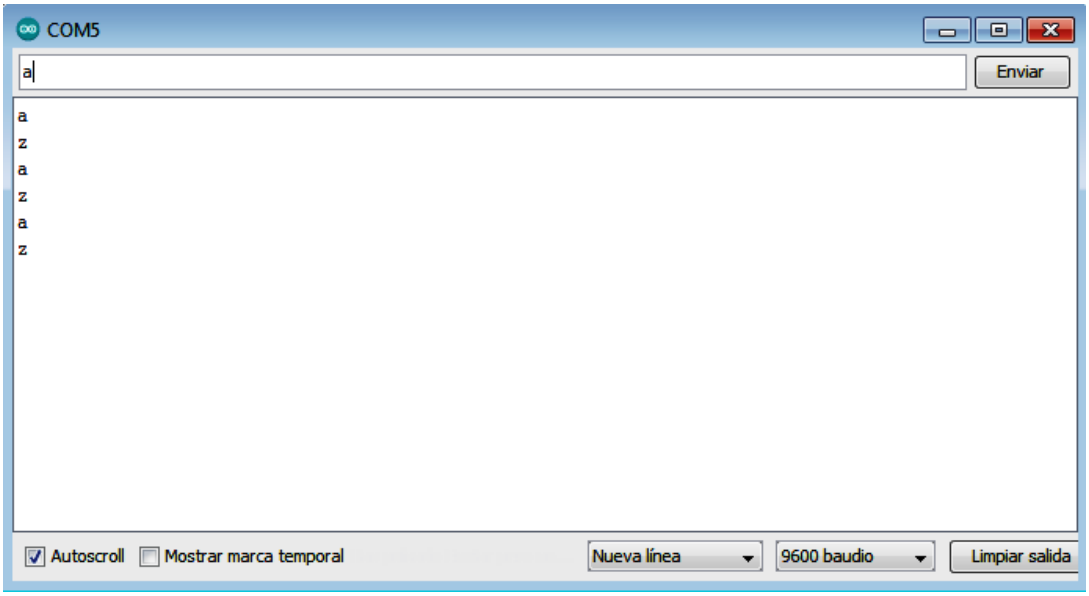

**Nota:** Durante el tiempo de funcionamiento Arduino está constantemente transmitiendo los datos y se señaliza mediante un diodo led TX en la tarjeta de Arduino.

También permite comunicarnos con la placa Arduino, ya sea leyendo el texto que nos muestra por pantalla o por el texto que le podemos introducir nosotros desde el monitor serie.

En el siguiente ejemplo se configura dos letras, la “a” y la “z”, cuando introducimos en la barra de envío del monitor serie la letra “a” y le damos **enter** se enciende el led y la letra “a” aparece y permanece en la zona de impresión, cuando introducimos la letra “z” se apaga el diodo led y la letra “z” aparece y se queda en la zona de impresión.

```
/*Código para gobernar un led mediante el teclado del PC. */

int pin=13;                  // asociamos el pin con el pin digital 13
char tecla;                 //la variable tecla será del tipo carácter
void setup() {
Serial.begin (9600);        //establecemos la comunicación a 9600 baudios
pinMode (pin, OUTPUT);      // configuramos el pin de salida
}
void loop() {
if (Serial.available()) {   // comprobamos que hay algo en la barra de
texto del monitor
tecla=Serial.read();     //lee lo que hay en el monitor serie y lo guarda
en la variable tecla
if (tecla=='a') {          // si la tecla pulsada es una a
Serial.println (tecla);     // se imprime el carácter en el monitor serie
digitalWrite (pin, HIGH);  // enciende el led
}
if (tecla=='z') {        // si el carácter introducido es una z…
Serial.println(tecla);  // se visualiza por el monitor serie la letra z
digitalWrite(pin, LOW);      //apaga el led
}
}
}
```




Arduino cuenta entre sus instrucciones de programación con la función “**map()”**, que permite “mapear” valores y adecuarlos a nuestra conveniencia.

**Función map()** permite adecuar los valores que proporcionan algunos sensores a otro rango de valores más apropiados para nuestros intereses.

Por ejemplo, si diseñamos un circuito con una fotorresistencia y deseamos observar los valores, mediante el **monitor serie**, que obtenemos en función de la luz que capta, comprobaremos que estos valores están situados entre un rango de 0 a 1023, al ser un sensor analógico.

Por el contrario, deseamos que estos valores estén en un rango comprendido entre 0 y 255, que son los valores que obtenemos para valores digitales de 8 bits.

La sintaxis de esta función es la siguiente:

**map** (valor, origen_menor, origen_mayor, destino_menor, destino_mayor)

Ejemplo:

```
/* mapeando el valor de una LDR */

int ldr=A0;              //pin de conexión de la LDR
int valor=0;             //almacena el valor de la LDR
void setup() {
Serial.begin(9600);      //iniciamos la comunicación serie
pinMode (ledr, INPUT);   //configuramos el pin de la ldr como entrada
}
void loop() {
valor=analogRead (ldr);     // lee el valor de la ldr y almacena en valor
map (valor, 0, 1023, 0, 255);   //valor se encuentra min y max   de
analógicos y digitales
if (valor==255) {          //si el valor es igual a 255
Serial.println ("LUZ MAXIMA");  //visualiza el texto en el monitor serie
}
}
```

## Serial.available()

Esta función permite "activar" la comunicación serie para recibir o enviar datos durante la ejecución de un programa en Arduino. De esta forma, cada vez que se lee esta función, se "mira" si el puerto serie contiene información para ser tratada.

La sintaxis es la siguiente:

```
Serial.available();
```

Normalmente, esta función va introducida en un **if** condicional, para evaluar si tenemos o no datos en el puerto serie.

```
if (Serial.available()){ } //comprobamos si hay algún dato en el puerto serie

if (Serial.available()>0) { } //se pregunta si el contenido es mayor que 0
```

Esta función va ligada en la mayoría de los casos a la función **Serial.read()** que se ve a continuación.

## int Serial.available()

En esta se obtiene un número entero con el número de bytes (caracteres) disponibles para leer o capturar desde el puerto serie. Equivaldría a la función **Serial.available()**.

Devuelve un entero con el número de bytes disponibles para leer desde el buffer serie, o 0 si no hay ninguno. Si hay algún dato disponible, **Serial.available()** será mayor que 0. El buffer serie puede almacenar como máximo 64 bytes.

Ejemplo

```
int incomingByte = 0; // almacena el dato serie
void setup() {
Serial.begin(9600); // abre el puerto serie, y le asigna la velocidad de 9600 bps
}
void loop() {
if (Serial.available() > 0) {    // envía datos sólo si los recibe
incomingByte = Serial.read();    // lee el byte de entrada
Serial.print("I received: ");    //lo vuelca a pantalla
Serial.println(incomingByte, DEC);
}
}
```

## Serial.read()

Esta función permite leer todo aquello que proviene del puerto serie. Almacenando estos datos en una variable, podemos tratarlos de la forma que más nos interese.

La sintaxis es la siguiente:

```
Serial.read();
Tecla=Serial.read();// Lee lo que hay en el puerto serie y lo guarda en la variable tecla
```

### int Serial.read()

Lee o captura un byte (un carácter) desde el puerto serie. Equivaldría a la función Serial.read().

Devuelve: El siguiente byte (carácter) desde el puerto serie, o =1 si no hay ninguno.

Ejemplo

```
int incomingByte = 0;    // almacenar el dato serie
void setup() {
Serial.begin(9600);  // abre el puerto serie,y le asigna la velocidad de
9600 bps
}
void loop() {
if (Serial.available() > 0) {     // envía datos sólo si los recibe:
incomingByte = Serial.read();     // lee el byte de entrada:
Serial.print("I received: ");
Serial.println(incomingByte, DEC);    //lo vuelca a pantalla
 }
 }
```

### Serial.println(data,data type)

Vuelca o envía un número o una cadena de caracteres al puerto serie, seguido de un carácter de retorno de carro "CR" (ASCII 13, or '\r') y un carácter de salto de línea "LF"(ASCII 10, or '\n'). Toma la misma forma que el comando **Serial.print()** .

### Serial.print (data, data type)

Vuelca o envía un número o una cadena de caracteres, al puerto serie. Dicho comando puede tomar diferentes formas, dependiendo de los parámetros que utilicemos para definir el formato de volcado de los números. Parámetros:

- **data:** el número o la cadena de caracteres a volcar o enviar.
- **data type:** determina el formato de salida de los valores numéricos (decimal, octal, binario, etc...) DEC, OCT, BIN, HEX, BYTE , si no se pe nada vuelva ASCII.

Veamos las siguientes opciones de **Serial.print()**:

### Serial.print(b)

Vuelca o envía el valor de b como un número decimal en caracteres ASCII. Equivaldría a printInteger().

```
int b = 79;
Serial.print(b);         // imprime la cadena  decimal 79
```

### Serial.print(b, DEC)

Vuelca o envía el valor de b como un número decimal en caracteres ASCII. Equivaldría a printInteger().

```
int b = 79;
Serial.print(b, DEC);       // imprime la cadena  decimal 79
```

### Serial.print(b, HEX)

Vuelca o envía el valor de b como un número hexadecimal en caracteres ASCII. Equivaldría a printHex();

```
int b = 79;
Serial.print(b, HEX); // imprime la cadena hexadecimal  4F
```

### Serial.print(b, OCT)

Vuelca o envía el valor de b como un número Octal en caracteres ASCII. Equivaldría a printOctal();

```
int b = 79;
Serial.print(b, OCT);  // imprime la cadena octal 117
```

### Serial.print(b, BIN)

Vuelca o envía el valor de b como un número binario en caracteres ASCII. Equivaldría a printBinary();

```
int b = 79;
Serial.print(b, BIN);       // imprime la cadena binaria 1001111
```

### Serial.print(b, BYTE)

Vuelca o envía el valor de b como un byte. Equivaldría a printByte();

```
int b = 79;
Serial.print(b, BYTE);  // Devuelve el carácter "O", el cual representa
el carácter ASCII del valor 79. (Ver tabla ASCII).
```

### Serial.print(str)

Vuelca o envía la cadena de caracteres como una cadena ASCII. Equivaldría a printString().

```
Serial.print("Hello World!");     // vuelca "Hello World!".
```

## 16. PROGRAMACIONES Y CIRCUITOS ELÉCTRICOS

Este último capítulo consta de 12 programaciones de diversos casos prácticos, realizados con la tarjeta de Arduino UNO R3, con su correspondiente código y circuito eléctrico.

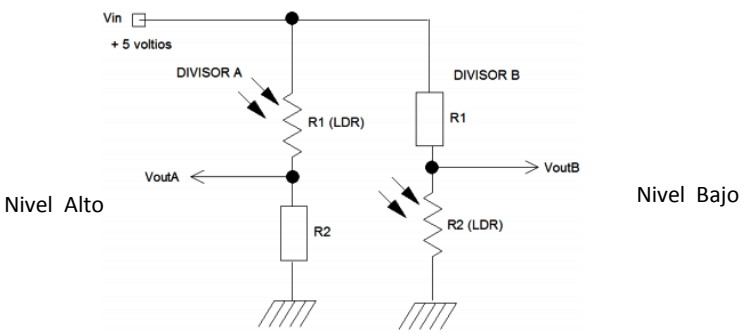
El componente principal y cerebro de la tarjeta de Arduino es el microcontrolador ATmega, en este caso, el ATmega 328P, que es quien lo programamos y lo podemos sacar de la tarjeta Arduino y colocarlo en nuestro circuito eléctrico diseñado.

Este microcontrolador trabaja con niveles de tensión de 3,3, voltios y 5 voltios y tan solo puede entregar una corriente máxima de 40mA a las salidas de sus pines, por lo que, tan solo se va a poder manejar señales que estén dentro de éste rango. Por lo que es recomendable utilizar en los circuitos eléctrónicos una fuente de 5 voltios en continua y de 12 voltios en continua que ofrezca una corriente de unos 500 mA.

Si la señal con la que estamos trabajando está dentro del rango que puede soportar el microcontrolador ATmega, no vamos a tener ningún problema a cualquier tipo de señal, ya sea digital o analógica.

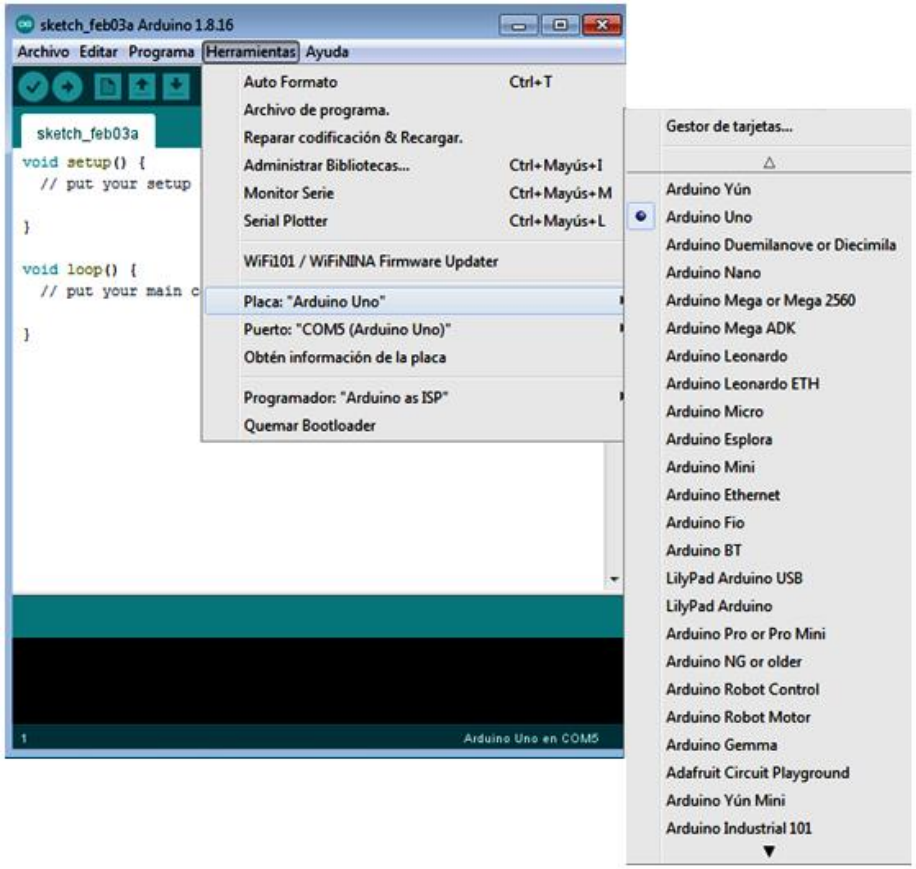
Pero no todos los casos son favorables y lo podemos desconocer, para ello se recomienda de aislar mediante circuitos optoacopladores las señales y tensión de los circuitos exteriores para una protección del microcontrolador.

Otro de los aspectos importantes que se debe conocer es la utilización de resistencias del tipo PULL-UP y PULL-DOWN para polarizar la entrada de los puertos digitales: pulsadores o interruptores, sensores, etc. Para conmutar la entrada de los pines entre un nivel ALTO (señal) y nivel BAJO (sin señal). Para conmutar los niveles ALTOS se utilizan los PULL-DOWN, una resistencia R2 se conecta a GND con el pin de entrada y un y un conmutador o sensor en serie conectado a +5Vcc. Y para conmutar los niveles BAJOS, se usan los PULL-UP, resistencia R1 conectada a +VCC y al pin de entrada y el sensor o conmutador conectado a GND.



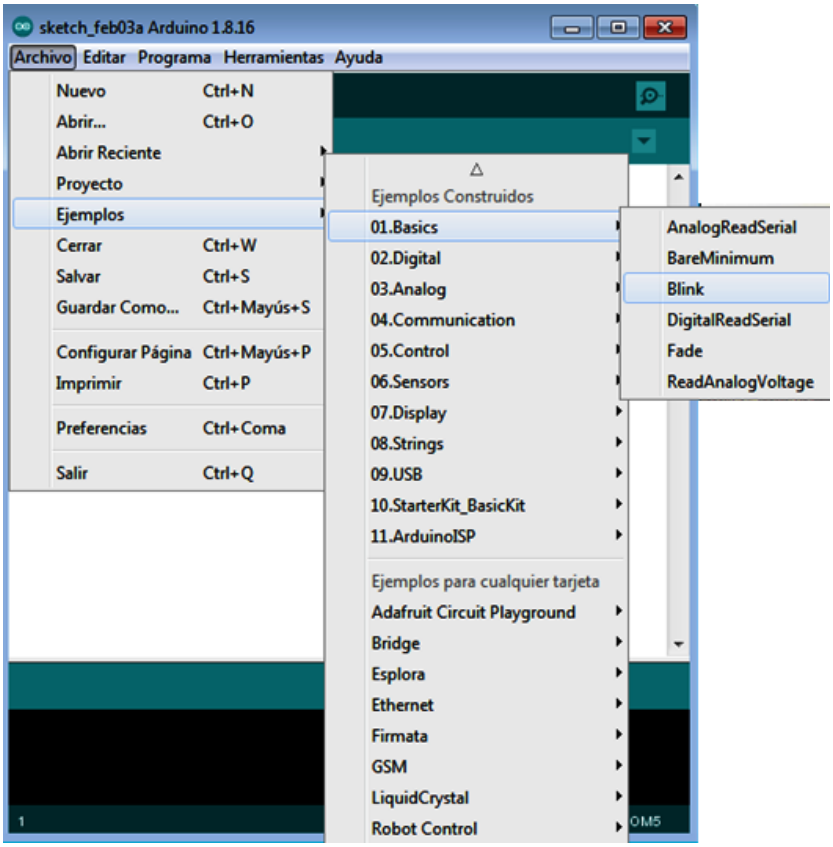
## Primeros programas

Anteriormente descubrimos el entorno de comandos de Arduino, ahora toca prácticar un poco con ellos. Pero antes tenemos que tener asegurado el buen funcionamiento de la placa de Arduino, generalmente que esté actualizada y sus controladores instalados.




**Configuración de Arduino Uno en el puerto serie "COM5" (Arduino Uno)**

El IDE de Arduino posee un amplio repertorio de ejemplos de sketch y para su comprobación cargaremos el sketch denominado "blink" del IDE de Arduino en "**Archivos/Ejemplos/01.Basics/Blink**" que es un programa sencillo que nos permite comprobrar a través de un led verde de la placa Arduino que se enciende intermitentemente y nos indica que todo está correcto: la compilación, el puerto, la tramisión, la recepción y el diodo Led verde.




**Cargando el programa Blink de Ejemplos/01.Basics/Blink**

A continuación se indican una series de información que nos permita solucionar algunos problemas para poder trabajar en el entorno de Arduino.

## Algunos errores en la instalación

No siempre se consigue a la primera que todo vaya bien en la instalación de Arduino y tengamos que echar mano a técnicas para solucionarlo. Tened en cuenta que hay dos partes bien diferenciadas en el entorno de Arduino: una es el hardware de la tarjeta Arduino y otra es la aplicación Software, IDE de Arduino, donde se ensambla y compila el programa que hemos hecho. Pero pueden ocurrir algunos errores:

1. La placa está conectada al PC a través de USB, los LED deben parpadear. Si esto no sucediera:
    - Cable USB defectuoso.
    - El puerto USB del PC está defectuoso.
    - Puerto USB Arduino defectuoso.
    - Pruebe con otro PC para eliminar algunos de los problemas anteriores.

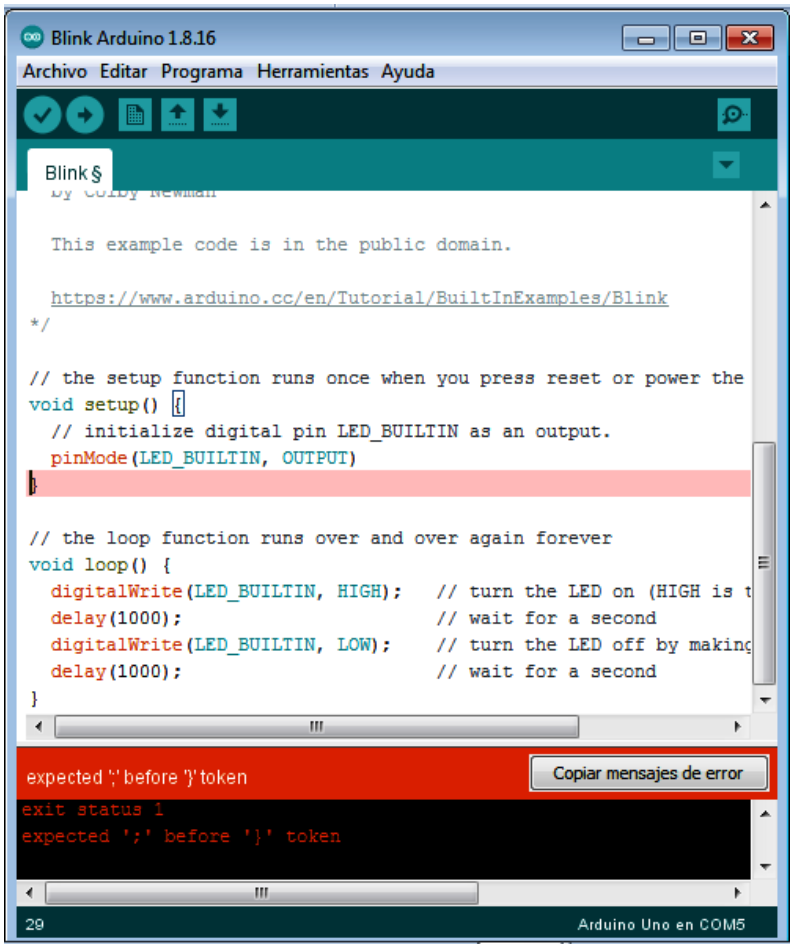
- Un diodo de entrada en la línea USB se quemó en el Arduino debido a un cortocircuito provocado por el usuario al ensamblar el circuito.
- La placa Arduino se quemó por completo debido a que el usuario conectó incorrectamente la alimentación externa o un cortocircuito.
- Pruebe con otra placa (preferiblemente una nueva) para excluir algunos de los problemas anteriores.

2. El PC emitirá una señal característica para conectar nuevos equipos, y en la primera conexión aparecerá la ventana “Instalar nuevo hardware”. Si esto no sucediera:
    - Ver lista de errores anterior.
    - El cable USB debe ser un cable de datos, no un simple «cargador».
    - Es aconsejable conectar el cable directamente al PC y no a través de un concentrador USB.
    - Los controladores Arduino no están instalados (durante la instalación del IDE o desde la carpeta del programa), vuelva a la instalación.
3. En la lista de puertos ( **Arduino IDE/Herramientas/Puerto**) aparecerá un nuevo puerto, generalmente **COM3, COM4** o **COM5**. Si esto no sucediera:
    - Ver lista anterior de errores.
    - El controlador **CH341** no está instalado **correctamente**.
    - Si la lista de puertos está inactiva en absoluto, el controlador Arduino no está instalado correctamente, vuelva a la instalación.

Si aparece el mensaje **Descarga completada**, entonces todo está en orden y puede cargar otros sketch. En cualquier caso, en su camino, encontrará otras dos variantes de eventos que ocurren después de hacer clic en el botón «Descargar»: un error de compilación y/o un error de descarga. Veámoslos con más detalle.

## Errores de compilación.

Ocurre en la etapa de ensamblaje y compilación del sketch. Los errores de compilación se deben a problemas **en el código del firmware**, es decir, el problema es puramente de software. A la izquierda del botón «descargar» hay un botón de marca de verificación y comprobación del sketch si está bien hecho. Durante la verificación, se compila el firmware y se detectan errores, si los hubiera.




**Errores de compilación: Exit status 1 expected “;” before “}” token**

- En este caso, en la ventana negra en la parte inferior del IDE de Arduino, puede leer el **texto completo del error** y comprender dónde empezar a buscar.
- En los sketch listos para usar descargados de Internet, a menudo se produce un error con la descripción **nombre_archivo** .h no existe tal archivo o directorio. Esto significa que el sketch usa la biblioteca **<nombre de archivo>**, y debe colocarlo en **programa / incluir libreria**.
- Al utilizar algunas bibliotecas, métodos o funciones especiales, una placa seleccionada incorrectamente en » *Herramientas / Placa* » puede convertirse en un error. **Ejemplo**: Firmware con *Mouse.h* o *keyboard.h*, la biblioteca se compila sólo para **Leonardo** y **Micro**.
- Si escribe el firmware, se resaltará cualquier error de sintaxis en el código, y en la parte inferior de la ventana negra podrá leer una descripción más detallada de qué es el fallo. Por lo general, se indica la línea en la que se cometió el error, y esta línea también se resalta en rojo.

- A veces, el motivo del error es una versión demasiado antigua o demasiado nueva del IDE de Arduino. Lea los comentarios del desarrollador de los sketch.
- El error de espacio libre insuficiente se produce por razones obvias. Los modelos de procesadores varian en su capacidad de memoria, si el proyecto usa una placa Nano con un procesador 328p y, hemos utilizado un procesador de 168, la capacidad de memoria está más limitada y se notará la diferencia. Optimización: memoria estática: la memoria ocupada por el código (bucles, funciones). La memoria dinámica está ocupada por variables.

Errores frecuentes en el código que conducen a errores de compilación:

- expected ',' or ';' – falta una coma o un punto y coma en la línea anterior.
- stray '\320' in program – Caracteres no admitidos en el código.
- expected unqualified-id before numeric constant: el nombre de la variable no puede comenzar con un dígito.
- … was not declared in this scope: la variable o función se está utilizando pero no se ha declarado. El compilador no puede encontrarla.
- redefinición de… – re-declaración de una función o variable.
- storage size of… isn't known: la matriz se especifica sin especificar el tamaño.

## Errores de carga

Ocurren en la etapa en que el firmware se ensambla, compila, no hay errores críticos en él y se carga en la placa a través de un cable USB. Puede producirse un error tanto debido a un mal funcionamiento del hardware como a la configuración del programa y del controlador.

- El cable USB que se conecta al Arduino debe ser un cable de datos, no un cable de solo carga. Los reproductores y los teléfonos inteligentes se conectan al PC mediante el cable que necesitamos.
- El motivo del error de descarga puede ser que no está instalado / mal instalados los controladores CH340, si tiene un NANO chino.
- También habrá un error avrdude: ser_open (): no se puede abrir el dispositivo si el puerto COM al que está conectado el Arduino no está seleccionado. Si no hay otros puertos además de COM1, lea los dos puntos anteriores, o pruebe con otro puerto USB, o incluso con otra computadora.

- La mayoría de los problemas de arranque causados por la «congelación» de arduino o del cargador de arranque pueden tratarse desconectando completamente el arduino de la fuente de alimentación. Luego insertamos el USB y lo grabamos nuevamente.
- El error de descarga puede deberse a una placa incorrecta seleccionada en **Herramientas / Placa,** así como a un procesador incorrecto en » *Herramientas / Procesador».* Además, en las últimas versiones de IDE, debe seleccionar **ATmega328P (antiguo cargador de arranque)** para placas NANO chinas.
- Si tiene un monitor de puerto COM abierto en otra ventana IDE de Arduino o la placa se comunica a través del puerto COM con otro programa (Ambibox, HWmonitor, SerialPortPlotter, Putty etc.), recibirá un error de descarga porque el puerto está ocupado. Desconéctese del puerto o cierre otras ventanas y programas.
- Si usa los pines RX o TX en su boceto, **desconecte todo de ellos**! Usando estos pines, el Arduino se comunica con la computadora, incluso para descargar el firmware.
- Si la descripción del error contiene que el cargador de arranque no responde y no está sincronizado, y se han verificado todos los elementos anteriores de esta lista, el cargador de arranque está muerto con un 95% de probabilidad. El segundo resultado desagradable es que el gestor de arranque ha fallado y debe volver a actualizarse.

## Advertencias

Además de los errores debido a los cuales el proyecto no se cargará en la tarjeta de Arduino y no funcionará, también hay advertencias que se muestran en texto naranja en el área negra del registro de errores. Pueden aparecer advertencias incluso cuando aparece » **Descarga completa** » encima del registro de errores. Esto significa que no hay errores incompatibles con la vida en el firmware, fue compilado y cargado en la placa. Entonces, ¿Qué significan las advertencias? La mayoría de las veces puede ver lo siguiente:

- **#pragma message**– los mensajes con la directiva Pragma suelen ser mostrados por bibliotecas, informando sobre su versión o algunas configuraciones.
- **No hay suficiente memoria, el programa puede ser inestable**: un poco por encima de esta advertencia, generalmente hay información sobre la memoria utilizada. **La memoria del dispositivo** se puede terminar hasta en un 99%, no pasará nada malo. Es una memoria flash y no cambia durante el funcionamiento. Pero es aconsejable usar la **memoria dinámica** no más del 85-90%, de lo contrario, puede haber fallas realmente incomprensibles en el trabajo, ya que la memoria está constantemente actualizándose durante el trabajo. Depende del boceto y principalmente del número de variables locales. Puede escribir un código que

funcione de manera estable con el 99% de la memoria SRAM ocupada. De nuevo, esto es solo una advertencia, no un error.

## Recuerda

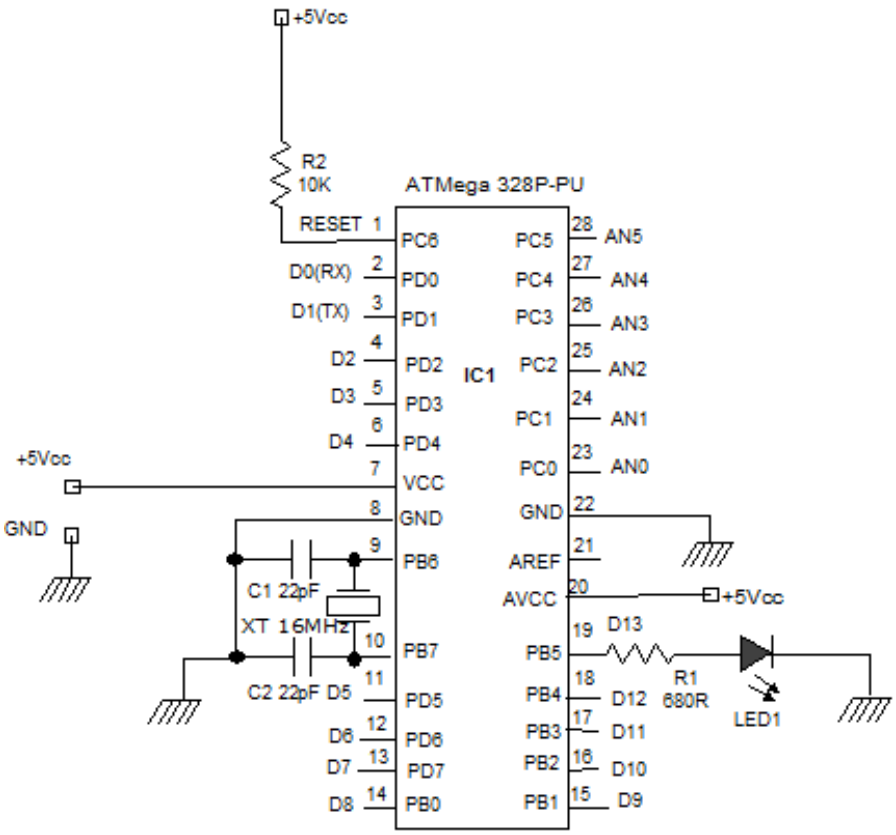
Concluyendo se indica a continuacución algunos puntos interesantes a tener en cuenta:

- Se puede regrabar tantas veces como se desee el sketch, varias decenas de miles de veces, todo depende del recurso de la memoria flash, y es bastante grande.
- La memoria se borra automáticamente cada vez que se vuelve a grabar, el firmware antiguo se elimina automáticamente.
- No se puede escribir dos sketch para que funcionen juntos, el firmware borra absolutamente todos los datos antiguos. Necesita hacer uno de los dos firmwares, para que no haya conflictos.
- Teóricamente es posible extraer el firmware del Arduino y volverlo a grabar, pero solo en forma de código de máquina ilegible, en el que se convierte el firmware C++ durante la compilación, es decir, esto NO ayudará EN NADA si no tienes un título en programación de ensamblador de bajo nivel.
- ¿Por qué es necesario? Por ejemplo, tenemos un dispositivo cerrado y queremos «clonarlo». En este caso, sí, existe la opción de volcar el firmware y subirlo a otra placa con el mismo microcontrolador.
- Si hay un deseo de leer el código, por desgracia, el firmware se lee en forma de código de máquina binario, que una persona común no puede volver a convertir en un código legible similar a C.
- Puede extraer el firmware, hablando de manera más científica: puede volcar el firmware utilizando el programador **ISP**.
- Puede eliminar el volcado de firmware solo si el desarrollador no ha limitado esta posibilidad, por ejemplo, escribiendo los **bits de bloqueo** que prohíben leer la memoria Flash o deshabilitando el bus **SPI** por completo. Si el desarrollador es usted y desea proteger su dispositivo de la copia tanto como sea posible, escriba los bits de bloqueo y la desactivación del bus SPI.

## Programa 1. Salida Digital

En este ejemplo el LED está conectado en el pin13, y se enciende y apaga "parpadea" cada segundo. La resistencia que se debe colocar en serie con el led en este caso puede omitirse ya que el pin13 de Arduino ya incluye en la tarjeta esta resistencia. Es una opción básica para comprobar tanto la programación realizada, la transmisión, recepción del programa en la tarjeta Arduino, como su funcionamiento, haciendo simplemente el encendido y apagado de un diodo led, nos indica que todo ha ido perfectamente.

```
/*brink de un led*/
int  ledPin=13;                // LED en el pin digital  13
void  setup() {                //  configura el pin de salida
pinMode(ledPin, OUTPUT);       // configura el pin 13 como salida
}
void  loop() {                 //  inicia el bucle del programa
digitalWrite(ledPin, HIGH);    //  activa el LED
delay(1000);                   //  espera 1 segundo
digitalWrite(ledPin, LOW);     //  desactiva el LED
delay(1000);                   //  espera 1 segundo
}
```

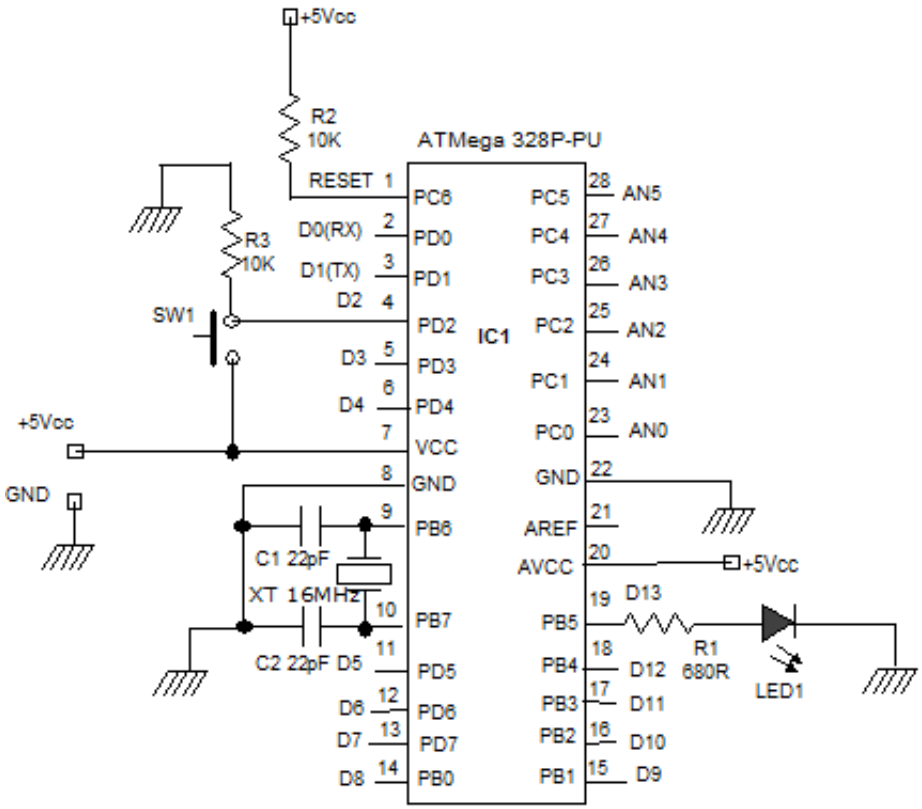



**Circuito eléctrico**

## Programa 2. Entradas digital

Ésta es la forma sencilla de entrada con sólo dos posibles estados: encendido o apagado. En este ejemplo se lee un simple switch o pulsador conectado a PIN2. Cuando el pulsador está oprimido cierra el circuito y el pin de entrada se lee ALTO y encenderá un LED colocado en el PIN13, de lo contrario está apagado.

```
/*encendido de un led al pulsar un botón*/
int  ledPin=13;          //  pin 13 asignado para el LED de salida
int  Pin=2;              //  pin 2 asignado para el pulsador
void  setup() {          //  Configura entradas y salidas
pinMode(ledPin, OUTPUT); //  declara  LED  como salida
pinMode(inPin, INPUT);   //  declara  pulsador como entrada
}
void  loop(){
if  (digitalRead(Pin)==HIGH) {  //testea si la entrada esta activa HIGH
digitalWrite(ledPin, HIGH);  //  enciende el LED
delay(1000);                 //  espera 1 segundo
}
digitalWrite(ledPin, LOW);   //  apaga el LED
}
```




**Circuito eléctrico**

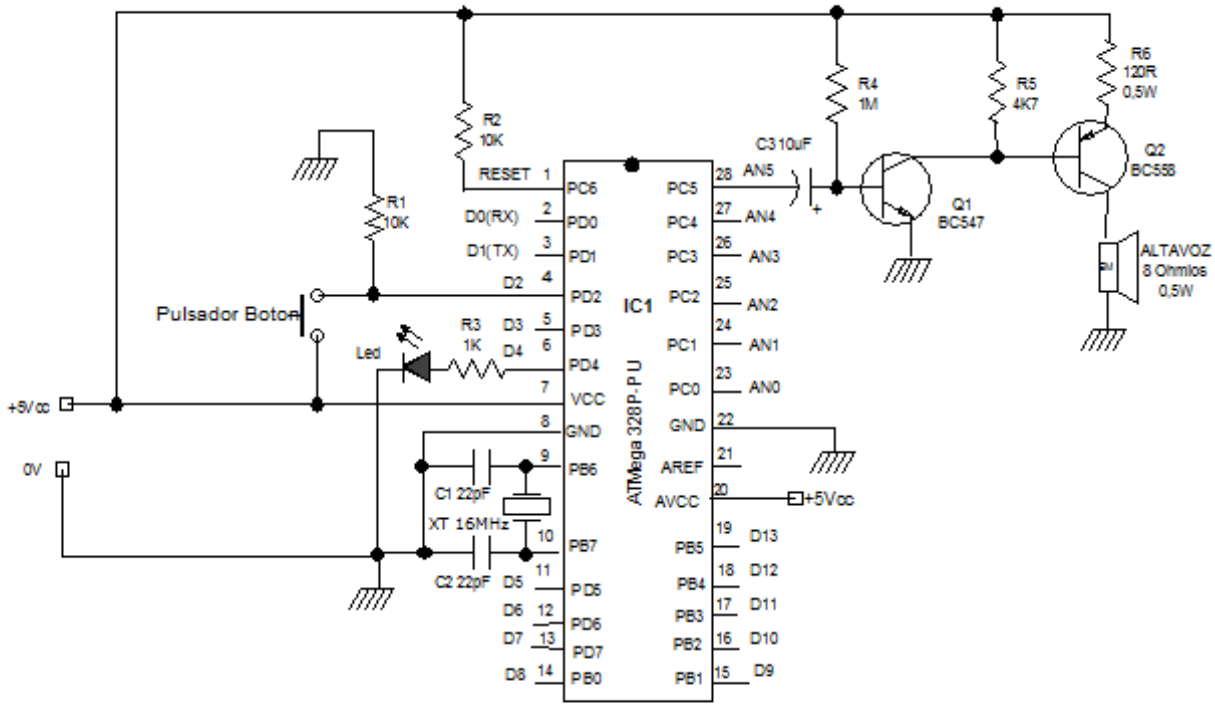
## Programa 3. Señalización con led y tonos

En esta programación se utiliza un diodo led y el sonido de unos tonos cuando se pulsa un pulsador botón. Para escuchar bien el sonido de los tonos se ha utilizado un pequeño amplificador a la salida del puerto analógico AN5. Cuando pulsamos el botón se escucha el sonido del tono ampliado y se enciende un diodo led, que se queda asín durante unos 8 segundos que posteriormente se apagan.

```
/*Pulsador botón que activa un led y suena tonos*/
int piezo=A5;   // declaramos la variable piezo para puerto analógico A5
const int botonPin=2;       // declara D2 el pin botón
const int ledPin=4;         // declara D4 pin LED
int Estadoboton=0;          // declara variable para Estadoboton

void setup() {
pinMode (piezo, OUTPUT);    // inicializa piezo como salida
pinMode(ledPin, OUTPUT);    // inicializa el pin LED como salida
pinMode(botonPin, INPUT);   // inicializa el pin boton como entrada:
}
void loop() {
Estadoboton = digitalRead(botonPin);  //lee el estado del valor pin botón

if (Estadoboton == HIGH) {//cuando pulsa  botón si estado botón es igual alto
digitalWrite(ledPin, HIGH);           //  enciende el led
  for (int i=100; i<=4000; i=i+50){   //hacemos un temporizador
    tone (piezo,320,100);             //activa tono de sonido
    tone (piezo,420,50);              //activa tono de sonido diferente
    delay(500);                       // espera medio segundo
  }
  }
else {                                // de lo contrario
digitalWrite(ledPin, LOW);            //apaga el led
 noTone(piezo);                       // apaga el tono
}
}
```




# CIRCUITO DE LLAMADA CON SEÑALIZACIÓN DE TONOS Y LED

| Plano: | Fecha: 08/11/2023 | Nº de Hojas: 1/1 |
|---|---|---|
| P-0127 | Dibujado: Jose M. Castillo Castillo | |

## Programa 4. Interruptor mediante sensor de temperatura

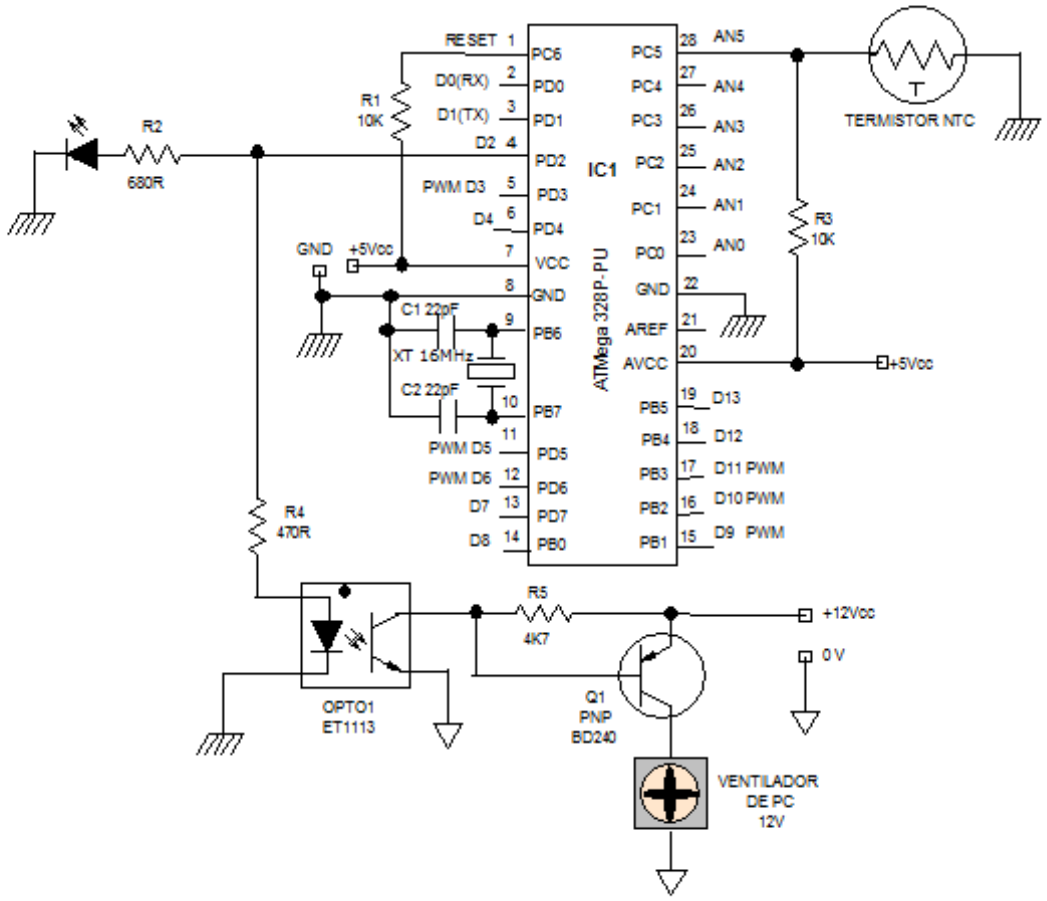
Esta programación consiste en programar, para el control de temperatura, un termistor NTC, resistencia dependiente de la temperatura, para cuando una determinada temperatura alcance un nivel alto, por ejemplo la temperatura de un PC interno, se ponga en funcionamiento un pequeño ventilador de 12 voltios que ventile y se desconecte cuando la temperatura vuelva a su estado normal.

Para obtener el valor deseado para una determinada temperatura y se ponga el ventilador en marcha se añade a la programación la función **Serial.begin(9600);** y **Serial.println(sensor);** donde se van visualizando en el monitor serie de Arduino los valores analógicos del sensor, escogiendo el rango más favorables para activar el ventilador para una determinada temperatura. En la programación se ha utilizado el rango para valorsensor por debajo de 100 para activar el ventilador.

En el circuito eléctrico se ha utlizado un optoacoplador para aislar y separar la salida del pin del microcontrolador y los +12 voltios que trabaja el ventiliador.

```
/*programacion de un sensor de temperatura que activa un ventilador*/
int sensor= A5;     //asignamos a la variable sensor el pin analógico A5
int led= 2;        // asignamos a la variable led el pin digital D2
int valorsensor;   //creamos variable

void setup() {
pinMode (sensor, INPUT);   //configuramos el pin sensor como entrada
pinMode (led, OUTPUT);     //configuramos el pin led como salida
}
void loop() {
valorsensor=analogRead(sensor); //guardamos lectura del sensor en valorsensor
  if (valorsensor>100) {     //si valorsensor es mayor de 100
    digitalWrite(led, LOW);  //desconecta el led
    delay(1000);             //espera 1 segundo
  }
  else {                    //de lo contrario
  digitalWrite(led,HIGH);   //activa la salida y enciende el led
 delay(1000);               //espera 1 segundo
}
}
```




# INTERRUPTOR MEDIANTE SENSOR DE TEMPERATURA

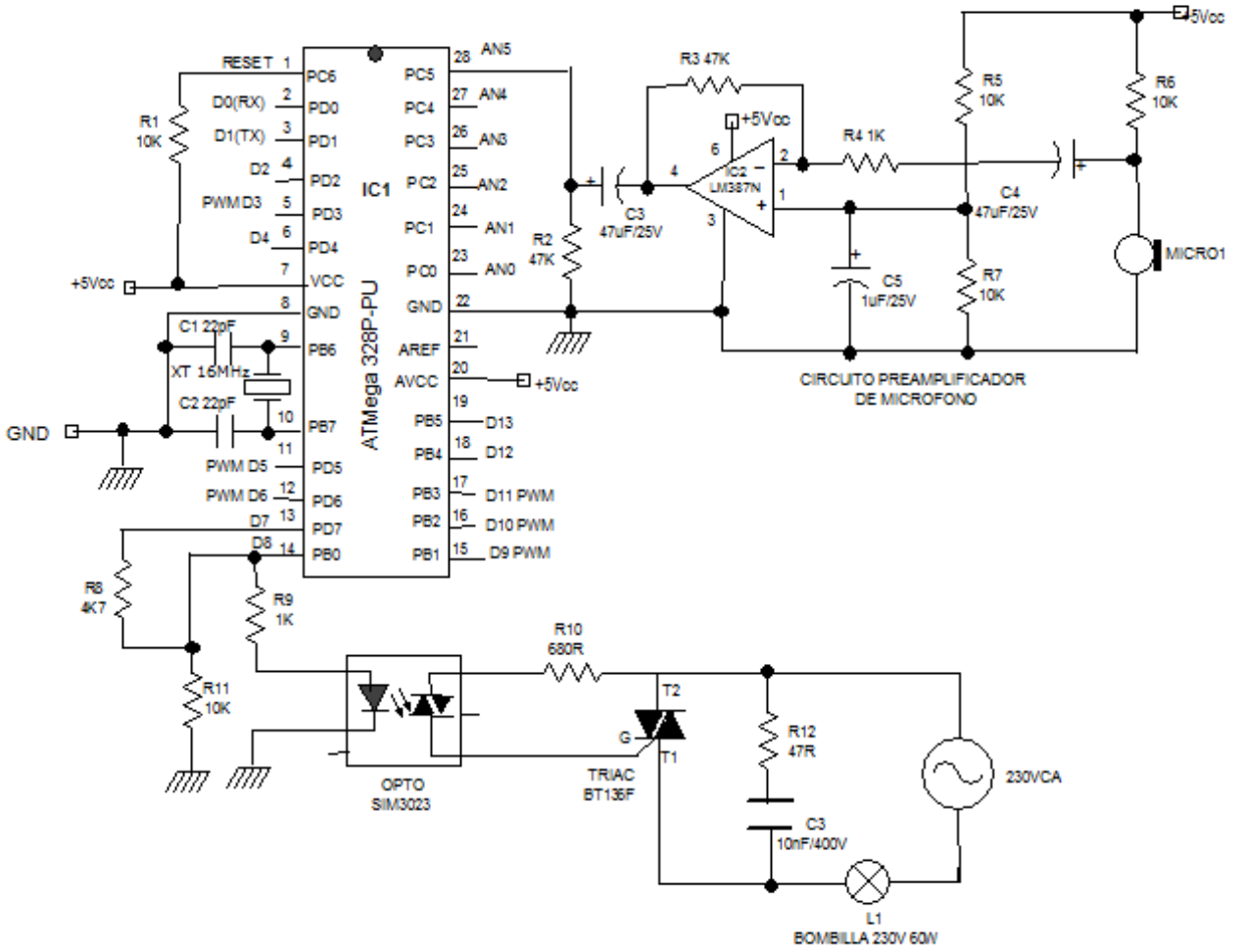
| Plano: | Fecha: 01/05/2023 | Nº de Hojas: 1/1 |
|---|---|---|
| P-0115 | Dibujado: Jose M. Castillo Castillo | |

## Programa 5. Interruptor mediante detector de sonidos

En este programa se ha tenido en cuenta la necesidad de utilizar un circuito preamplificador de micrófono para detectar el sonido y especificar en la programación el rango de sensibilidad que deseamos que se active la salida según el sonido, para ello, se puede visualizar a través del monitor serie de Arduino. Por ejemplo, si damos una palmada, se activa permanente, si volvemos a dar la palmada se apaga, como de un interruptor se tratase.

```
/*Sensor de sonido*/
int led=8;        // asignamos variable led al pin digital D8
int led2=7;       //asignamos variable led2 al pin digital D7
int sonido=A5;    //asignamos variable sonido al pin analogico A5
int detec;       //asignamos variable a detec
int comp;        //asignamos variable comp

void setup() {
  pinMode (led, OUTPUT);     //configuramos el pin led de salida
  pinMode (led2, INPUT);     //configuramos el pin led2 de salida
  pinMode (sonido, INPUT);   //configuramos el pin sonido de entrada
  Serial.begin(9600);        //configuramos el monitor serie
}
void loop() {
 detec=analogRead(sonido); //guardamos el valor analógico leído en sonido
 Serial.println(detec);      //visualizamos en el monitor serie los datos que
contiene la variable detec y elegimos entre el rango de detección
 comp=digitalRead(led2);   //leemos el valor del pin digital led2
 if (detec>=140 && comp==LOW){ //si detec es mayor o igual a 140 y comp esta en
bajo
  digitalWrite(led, HIGH); //enciende el led
  delay(5000);             //espera 5 segundo para estabilizar los datos
 }
if (detec>=140 && comp==HIGH) { //si detec es meyor o igual a 140 y comp esta
en alto
digitalWrite(led, LOW);  //apaga el led
delay(5000);      //esperamos 5 segundos para estabilizar los datos
}
}
```




# CIRCUITO INTERRUPTOR MEDIANTE DETECTOR DE SONIDO

| Plano: | Fecha: 01/05/2010 | Nº de Hojas: 1/1 |
|---|---|---|
| P-0105 | Dibujado: Jose M. Castillo Castillo | |

## Programa 6. Detector de movimientos con activación de luz

En esta programación consiste en un volumétrico infrarrojo que al detectar movimiento hace activar un circuito de luz con una bombilla de 230Vca y se queda encendida durante un minuto y medio y se apaga posteriormente si no existe movimientos.

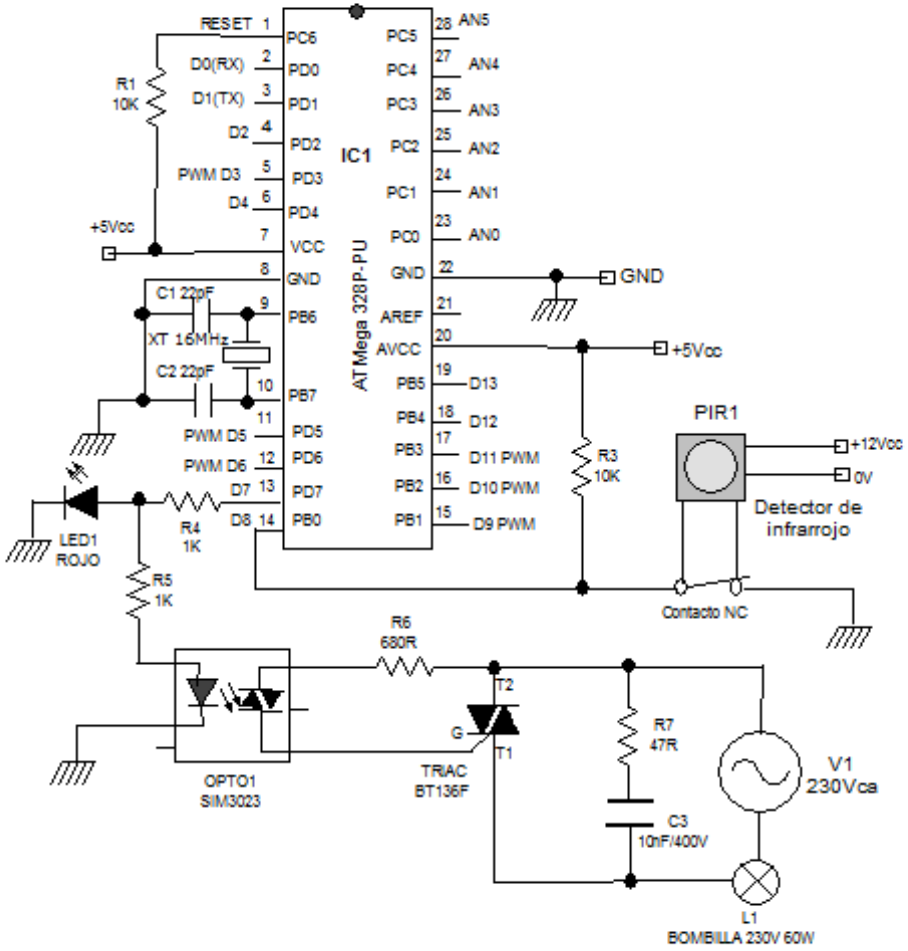
El detector volumétrico de infrarrojo cuya salida, cuando se conecta a la alimentación de 12 voltios, se encuentra normalmente cerrada, y cuando se produce una detección la salida cambia y pasa a estar abierta.

Para este caso se ha seleccionado un puerto digital D8 y le hemos llamado “detec” que se activará con un nivel alto (5 voltios) cuando exista un movimiento, de lo contrario estará a un nivel bajo (0 voltios) cuando no detecte movimiento.

Para señalizar esto se ha pensado a la salida utilizar un diodo led rojo y la conexión a un optoacoplador para encender una bombilla de 230 voltios que estará aproximadamente encendida un minuto y 38 segundos, según el bucle repetitivo, cuando se produce movimiento.

```
/*Código para detectar movimiento y encender una luz durante un determinado
tiempo*/
int pin=7;                    //asignamos el pin con el pin digital 7
int detec=8;                  //asignamos detec al pin digital 8
int movi;                     //asignamos la variable movi

void setup() {
  pinMode (pin, OUTPUT);      //configuramos el pin de salida
  pinMode (detec, INPUT);     // configuramos detect de entrada
}
void loop() {
  movi=digitalRead(detec);    //lee el valor de detec y guardalo en movi
  if (movi==HIGH) {           //si movi esta a nivel alto
  for (int w=0; w<500; w++) {  // creamos un bucle repetitivo que dura 1 minuto
y 38 segundos
  digitalWrite(pin, HIGH);  //ponemos el pin a nivel alto y enciendo led
  delay(200);     //esperamos 200 milisegundos que se añade a variable w
}
}
 digitalWrite(pin, LOW);    //termina el bucle y apagamos led
}
```




# CIRCUITO DETECTOR DE MOVIMIENTO CON ACTIVACIÓN DE LUZ

| Plano: | Fecha: | Nº de Hojas: |
|---|---|---|
| P-0105 | 10/11/2023 | 1/1 |
| | Dibujado: Jose M. Castillo Castillo | |

## Programa 7. Control de un volumétrico de infrarrojo mediante una fotorresistencia LDR

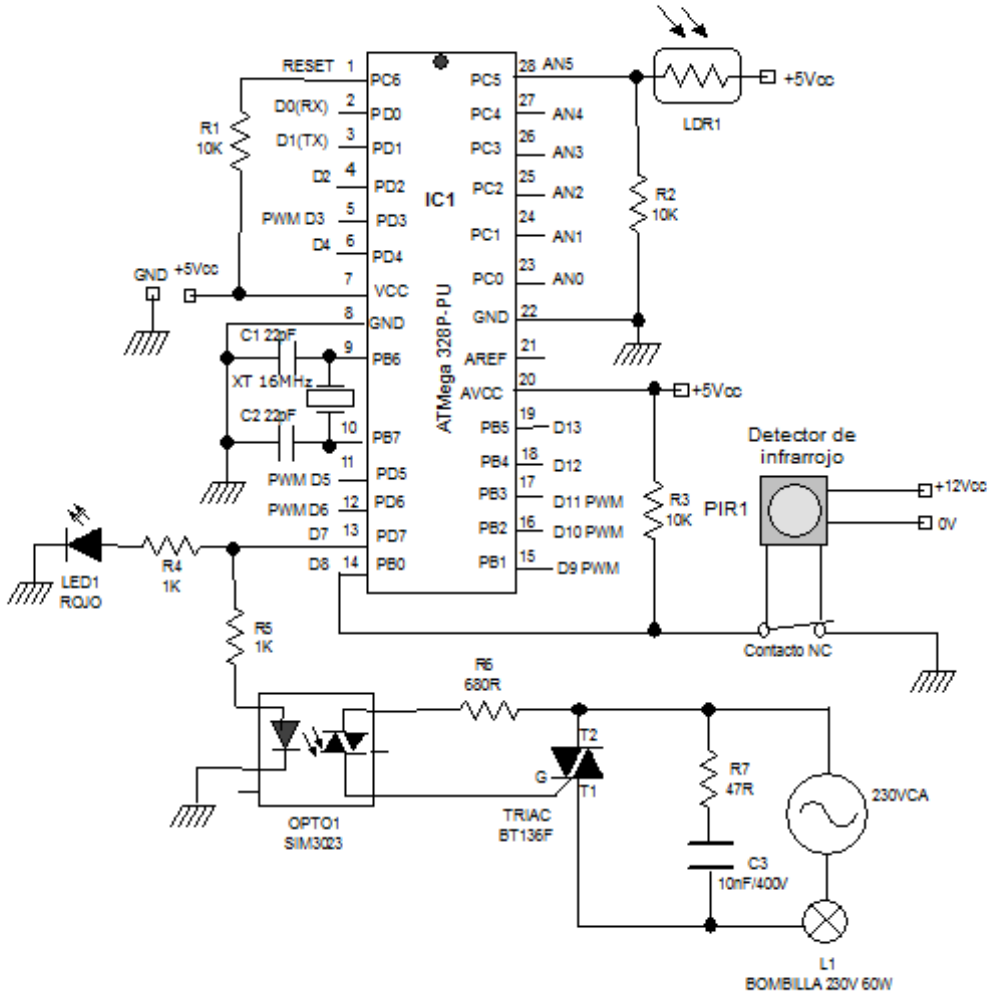
Esta programación es un añadido a la programación 6. Aquí se añade una fotocélula **LDR** para discriminar cuando hay luz y no la haya. Cuando no hay apenas luz natural hace activar el sistema de detención por infrarrojo para encender una bombilla y cuando la luz natural sea lo suficientemente alta hará que la fotocélula no active el detector PIR1.

La **LDR** va colocada a la entrada de un puerto analógico puesto que tiene una variedad de valores en el que se tiene que definir un valor entre 0 a 599 para la oscuridad, se conecta el sistema de detección, y entre 600 a 1023 para la luz, se desconecta el sistema de detección. Estos valores se puede obtener activando el monitor serie de Arduino **Serial.begin(9600);** y **Serial.println(valor);**

```
/*Código con una LDR y detector movimiento para encender una luz durante un
determinado tiempo*/
int ldr=A5;                 //asociamos el pin analogico A5 a la LDR
int pin=7;                  // asociamos el pin con el pin digital 7
int detec=8;                //asociamos detec al pin digital 8
int movi;                   //creamos la variable movi
int valor;                  // declaramos variable valor

void setup() {
pinMode (pin, OUTPUT);      //configuramos el pin de salida
pinMode (detec, INPUT);     // configuramos detect de entrada
pinMode (ldr, INPUT);       // configuramos la ldr como entrada

}
void loop() {
valor=analogRead(ldr);       //lee valor de la ldr y guardalo en valor
if (valor>=600) {             //si valor es mayor o igual a 600
digitalWrite(pin, LOW);     // desactiva la salida pin y apaga led
delay(100);                 // espera 100 milisegundos
}
else {                       //de lo contario haz lo siguiente
movi=digitalRead(detec);    //lee el valor de detec y guardalo en movi
if (movi==HIGH) {           //si movi esta a nivel alto
for (int w=0; w<500; w++) { // creamos temporizador 1 minuto y 38 segundos
digitalWrite(pin, HIGH);    //ponemos el pin a nivel alto y enciendo led
delay(200);          //esperamos 200 milisegundos que se añade a variable w
}
}
digitalWrite(pin, LOW);   //termina el bucle y apagamos pin led
}
}
```




# CIRCUITO DE CONTROL DE UN VOLUMETRICO PIR MEDIANTE LDR

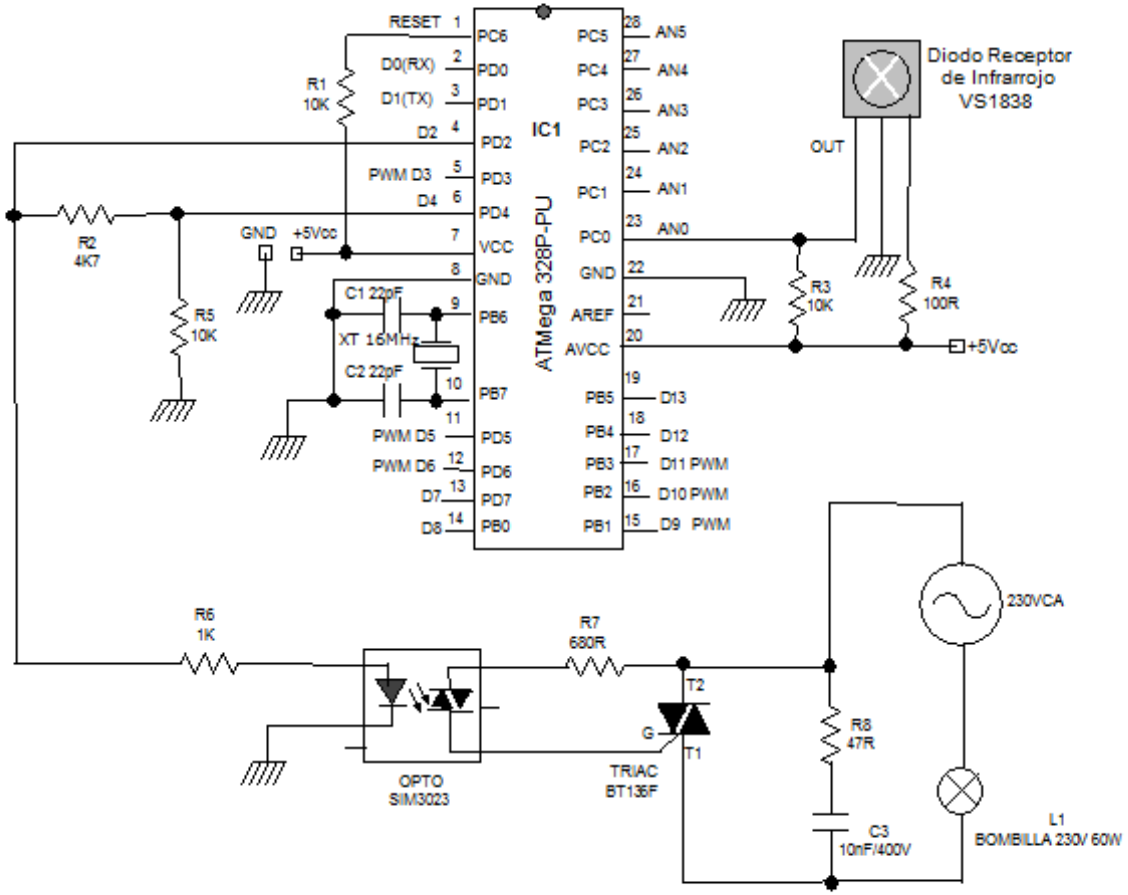
| Plano: | Fecha: | Nº de Hojas: |
|---|---|---|
| P-0105 | 10/12/2023 | 1/1 |
| | Dibujado: Jose M. Castillo Castillo | |

## Programa 8. Interruptor por control remoto

Este programa consiste en el control de la entrada de señales de un diodo receptor de infrarrojo a través de cualquier mando a distancia para encender o apagar una bombilla de corriente alterna de 230 voltios.

```
/* encendido y apagado por mando a distancia de infrarrojo*/
int valor;       //asignamos la variable
int valor2;      //asignamos la variable 2
int rea=4;       // asignamos el puerto digital 4 a rea
int sensor=A0;   //asignamos el puerto analógico a sensor
int led=2;       //asignamos el puerto digital 2 a led

void setup() {
pinMode (sensor, INPUT); // configuramos sensor de entrada
pinMode (led, OUTPUT);   //configuramos led de salida
pinMode (rea, INPUT);    //configuramos el pin rea como entrada
}
void loop() {
valor=analogRead(sensor);   // alamcena el valor del sensor analogico en valor
valor2=digitalRead(rea);         // alamcena el valor de rea digital en valor2
if (valor<100 && valor2==LOW) {   //condicionante  si valor menor de 100 y
valor2 bajo
digitalWrite(led, HIGH);     //encendemos led
delay(500);                  //espera medio segundo
  }
 if (valor<100 && valor2==HIGH) {//condicionante si valor menor de 100 y
valor2 alto
digitalWrite (led, LOW);        // apagamos led
delay(500);                      //espera medio segundo
}
}
```




# CIRCUITO INTERRUPTOR POR CONTROL REMOTO

| Plano: | Fecha: 21/12/2023 | Nº de Hojas: 1/1 |
|---|---|---|
| P-0105 | Dibujado: Jose M. Castillo Castillo | |

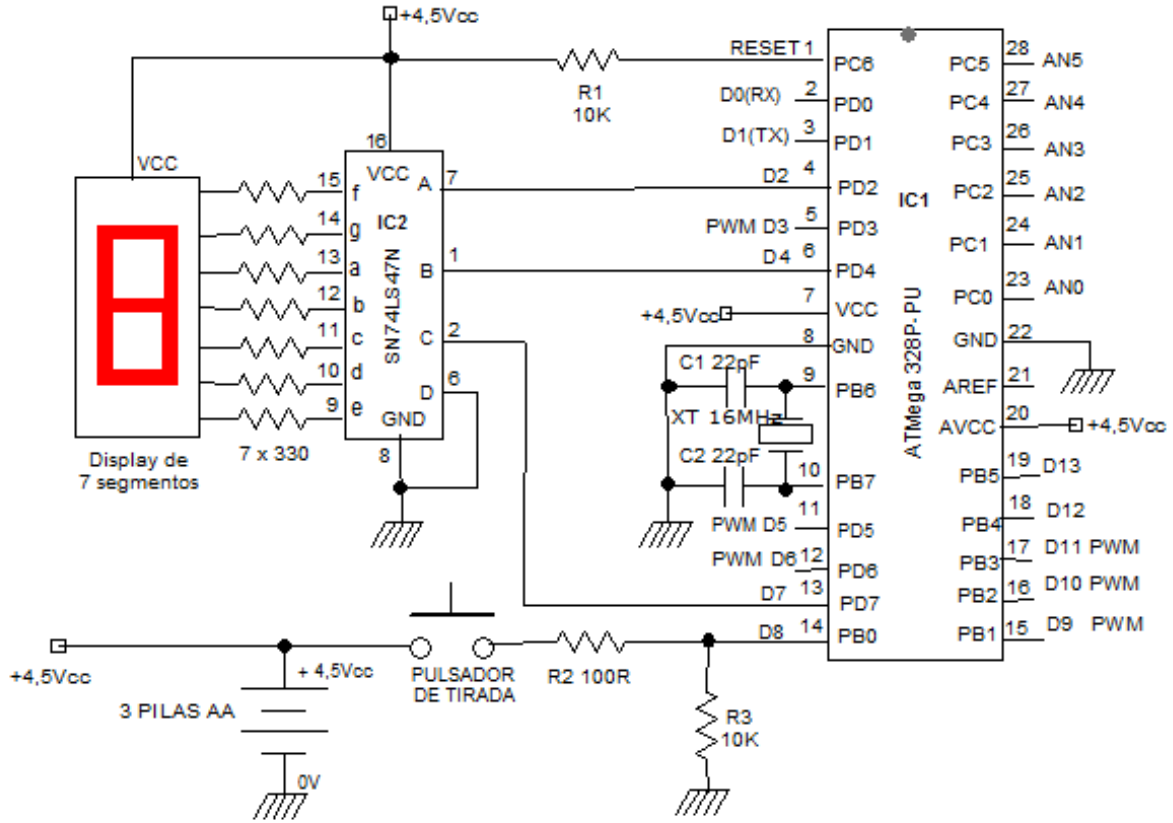
## Programa 9. Dado electrónico

Este programa consiste en el control aleatorio de seis números del 1 al 6 que se visualiza mediante un decodificador de 7 segmentos en un display. Es el típico dado electrónico.

```
/*Programacion para un dado electronico*/
int A=2;                  //asignamos A al pin digital D2
int B=4;                  //asignamos B al pin digital D4
int C=7;                  //asignamos C al pin digital D7
int boton=8;              //asignamos boton al pin digital D8
int estadopulsador;       //creamos variable
int maximo=6;             //asignamos la variable maximo a 6
int randomNumber;         //creamos variable

void setup() {
pinMode (A, OUTPUT);      //configuramos A como salida
pinMode (B, OUTPUT);      //configuramos B como salida
pinMode (C, OUTPUT);      //configuramos C como salida
pinMode (boton, INPUT);   //configuramos boton como entrada
}
void loop() {
estadopulsador=digitalRead(boton);  //lee el estado del pulsador
if (estadopulsador==HIGH) {         // si estadopulsador esta en alto
for (int w=0; w<5; w++) {           // bucle repetitivo hasta 5
dado1();                            //ve a la función dado1
delay(100);                         //espera 100 milisegundos
dado2();                            //ve a la función dado2
delay(100);                         //espera 100 milisegundos
dado3();                            //ve a la función dado3
delay(100);                         //espera 100 milisegundos
dado4();                            //ve a la función dado4
delay(100);                         //espera 100 milisegundos
dado5();                            //ve a la función dado5
delay(100);                         //espera 100 milisegundos
dado6();                            //ve a la función dado6
}
randomSeed(millis());               //se genera numeros aleatorios
randomNumber=random(maximo);        //guarda el valor obtenido de random en
ramdomNumber
if (randomNumber==1) {              //si es igual a 1
  dado1();                          //ve a la función dado1
}
if (randomNumber==2) {              //si es igual a 2
  dado2();                          //ve a la función dado2
}
if (randomNumber==3) {              //si es igual a 3
  dado3();                          //ve a la función dado3
}
if (randomNumber==4) {              //si es igual a 4
  dado4();                          //ve a la función dado4
}
if (randomNumber==5) {              //si es igual a 5
  dado5();                          //ve a la función dado5
}
```

```
if (randomNumber==6) {              //si es igual a 6
  dado6();                          //ve a la función dado6
}
}
}
void dado1() {                      // funcion dado1
  digitalWrite (A, HIGH);           // pon a nivel alto el pin A
  digitalWrite (B, LOW);            // pon a nivel bajo el pin B
  digitalWrite (C, LOW);            // pon a nivel bajo el pin C
}
void dado2() {                      // funcion dado2
  digitalWrite (A, LOW);            // pon a nivel bajo el pin A
  digitalWrite (B, HIGH);           // pon a nivel alto el pin B
  digitalWrite (C, LOW);            // pon a nivel bajo el pin C
}
void dado3() {                      // funcion dado3
  digitalWrite (A, HIGH);           // pon a nivel alto el pin A
  digitalWrite (B, HIGH);           // pon a nivel alto el pin B
  digitalWrite (C, LOW);            // pon a nivel bajo el pin C
}
void dado4() {                      // funcion dado4
  digitalWrite (A, LOW);            // pon a nivel bajo el pin A
  digitalWrite (B, LOW);            // pon a nivel bajo el pin B
  digitalWrite (C, HIGH);           // pon a nivel alto el pin C
}
void dado5() {                      // funcion dado5
  digitalWrite (A, HIGH);           // pon a nivel alto el pin A
  digitalWrite (B, LOW);            // pon a nivel bajo el pin B
  digitalWrite (C, HIGH);           // pon a nivel alto el pin C
}
void dado6() {                      // funcion dado6
  digitalWrite (A, LOW);            // pon a nivel bajo el pin A
  digitalWrite (B, HIGH);           // pon a nivel alto el pin B
  digitalWrite (C, HIGH);           // pon a nivel alto el pin C
}
```




# CIRCUITO DE UN DADO ELECTRÓNICO PROGRAMADO EN ARDUINO

| Plano: | Fecha: 22/01/2024 | Nº de Hojas: 1/1 |
|---|---|---|
| ? 0105 | Dibujado: Jose M. Castillo Castillo | |

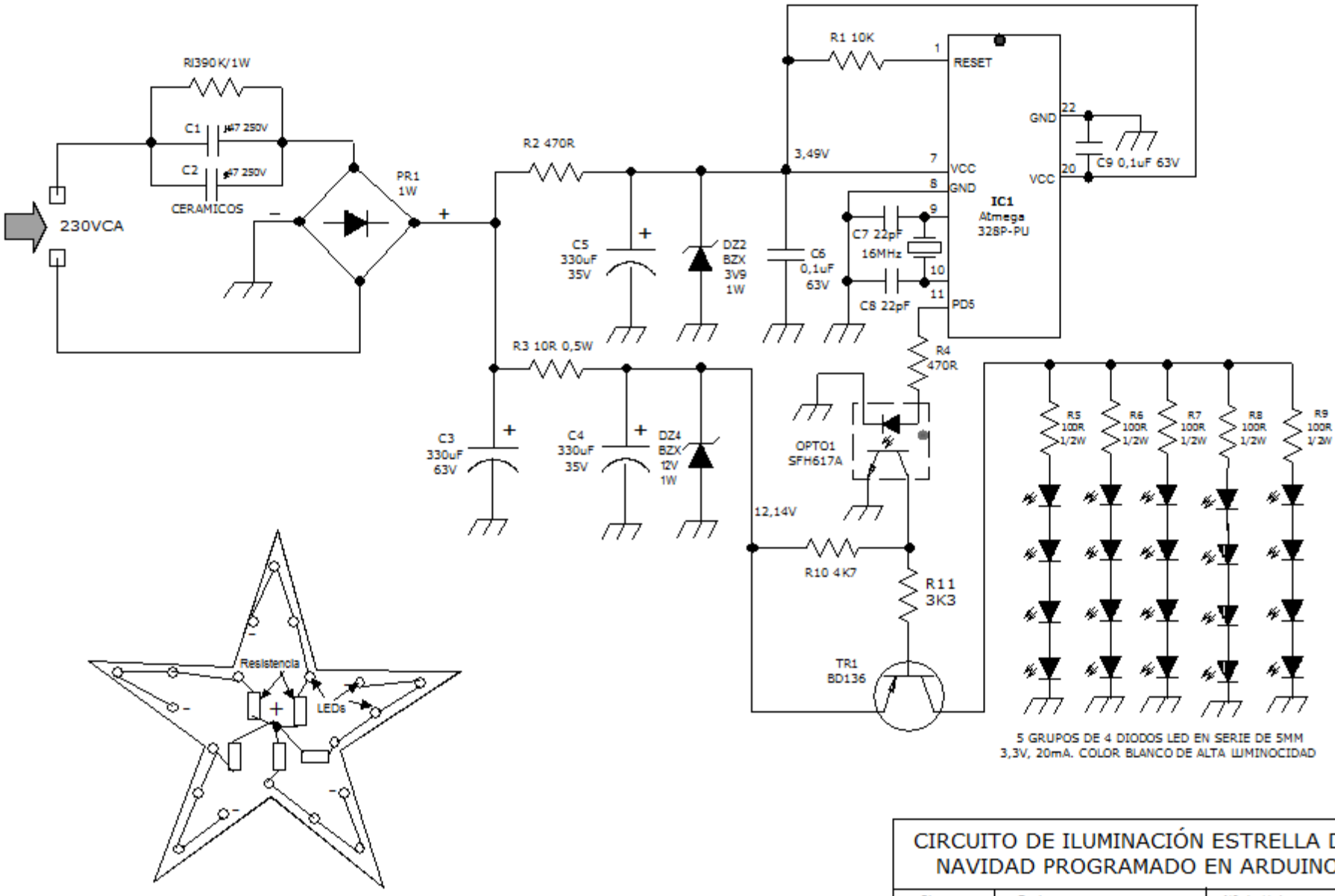
## Programa 10. Programación de la iluminación de una estrella de navidad

En esta programación solamente se ha utilizado una salida digital D5 para realizar el efecto de iluminación de una estrella de navidad, y consiste en una programación aleatoria de cuatro grupos de funciones que realizan diferentes efectos de iluminación cada una durante un determinado tiempo.

```
/* programación de la iluminacion de una estrella de navidad*/
int led=5;              //asignamos variable led al puerto digital 5
int maximo=4;           //asignamos a la variable máximo el valor 4
int aleatorio;          //asignamos la variable aleatorio

void setup() {
 pinMode(led, OUTPUT); // configuramos el led pin digital 5 como salida
}
void loop() {
randomSeed(millis());     //se genera numeros aleatorios
aleatorio=random(maximo); //guarda el valor obtenido de random en aleatorio
if (aleatorio==1) {       // si aleatorio es igual a 1
  efecto1();              //ve a la función efecto1
}
if (aleatorio==2) {       // si aleatorio es igual a 2
  efecto2();              //ve a la función efecto2
}
if (aleatorio==3) {       //si aleatorio es igual a 3
  efecto3();              //ve a la función efecto3
}
if (aleatorio==4) {       //si aleatorio es igual a 4
  efecto4();              //ve a la función efecto4
}
}
void efecto1() {           //función efecto1
  for(int w=0; w<4; w++) {  //bucle repetitivo
   digitalWrite(led, HIGH); //enciende el led
   delay(2000);             //espera dos segundos
  }
  }
void efecto2() {            //función efecto2
  for(int a=0; a<5; a++) {  //bucle repetitivo
   digitalWrite(led, LOW);  //apaga el led
   delay(1000);             //espera un segundo
   digitalWrite(led, HIGH); //enciende el led
   delay(100);              //espera 100 milisegundos
  }
  }
void efecto3() {            //función efecto3
  for(int b=0; b<5; b++) {  //bucle repetitivo
   digitalWrite(led, HIGH); //enciende el led
   delay(200);              //espera 200 milisegundos
   digitalWrite(led, LOW);  //apaga el led
   delay(100);              //espera 100 milisegundos
  }
```

```
  }
void efecto4() {             //función efecto4
  for(int c=0; c<5; c++) {  // bucle repetitivo
   digitalWrite(led, HIGH); //enciende el led
   delay(70);               //espera 70 milisegundos
   digitalWrite(led, LOW);  //apaga el led
   delay(50);               //espera 50 milisegundos
  }
  }
```



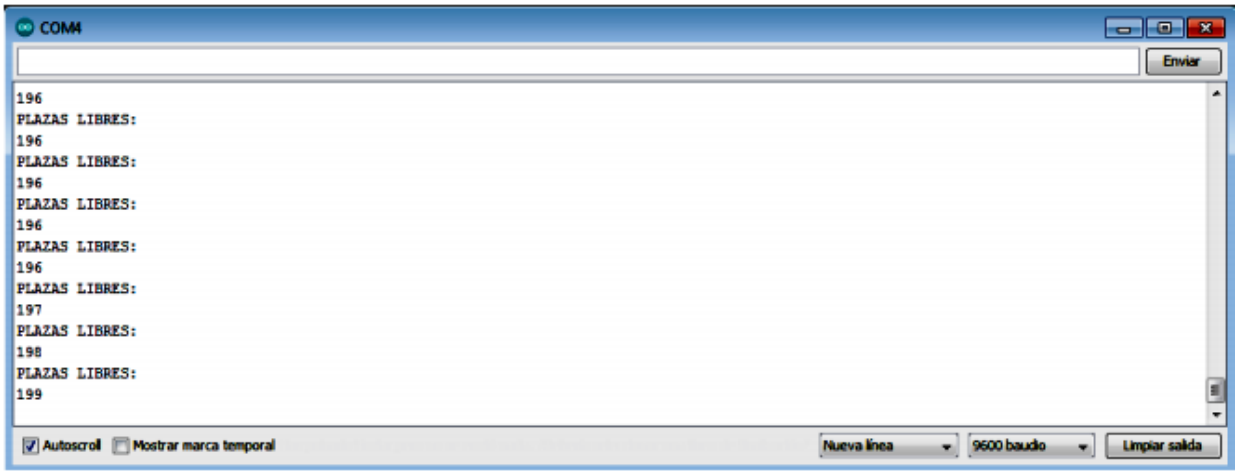

# CIRCUITO DE ILUMINACIÓN ESTRELLA DE NAVIDAD PROGRAMADO EN ARDUINO

| Plano: | Fecha: 14/12/2023 | Nº de Hojas: 1/1 |
|---|---|---|
| P-1199 | Dibujado: Jose M. Castillo Castillo | |

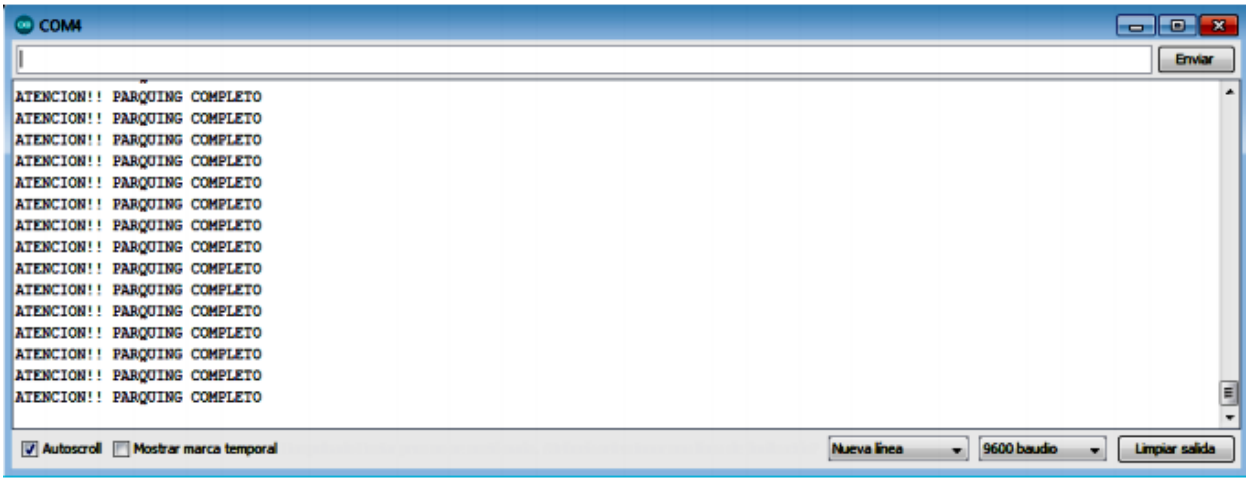
## Programación 11. Control de entrada y salida de vehículos en un Parking

Esta programación consiste en controlar la entrada y salida de los vehículos que entran y salen de un Parking y tiene como principal tarea detectar el número de vehículos que entran y salen del parking informando continuamente del número de plazas libres y avisando cuando el parking está completo. Para ello, también se va a disponer exteriormente de un semáforo que nos indica que el Parking está libre o completo, en nuestro caso, son dos diodos LED de color verde para libre y rojo para completo.

Utilizaremos el monitor serie de Arduino para visualizar las plazas libres y las que se van ocupando.




El máximo número de plazas son 200 y existe dos direcciones la **Entrada del Parking** y la **Salida del Parking**. La **Entrada** se decrementa y la **Salida** se incrementa el número de plazas, apareciendo en la pantalla del monitor serie el número de plazas libres y el led de color verde estará encendido.



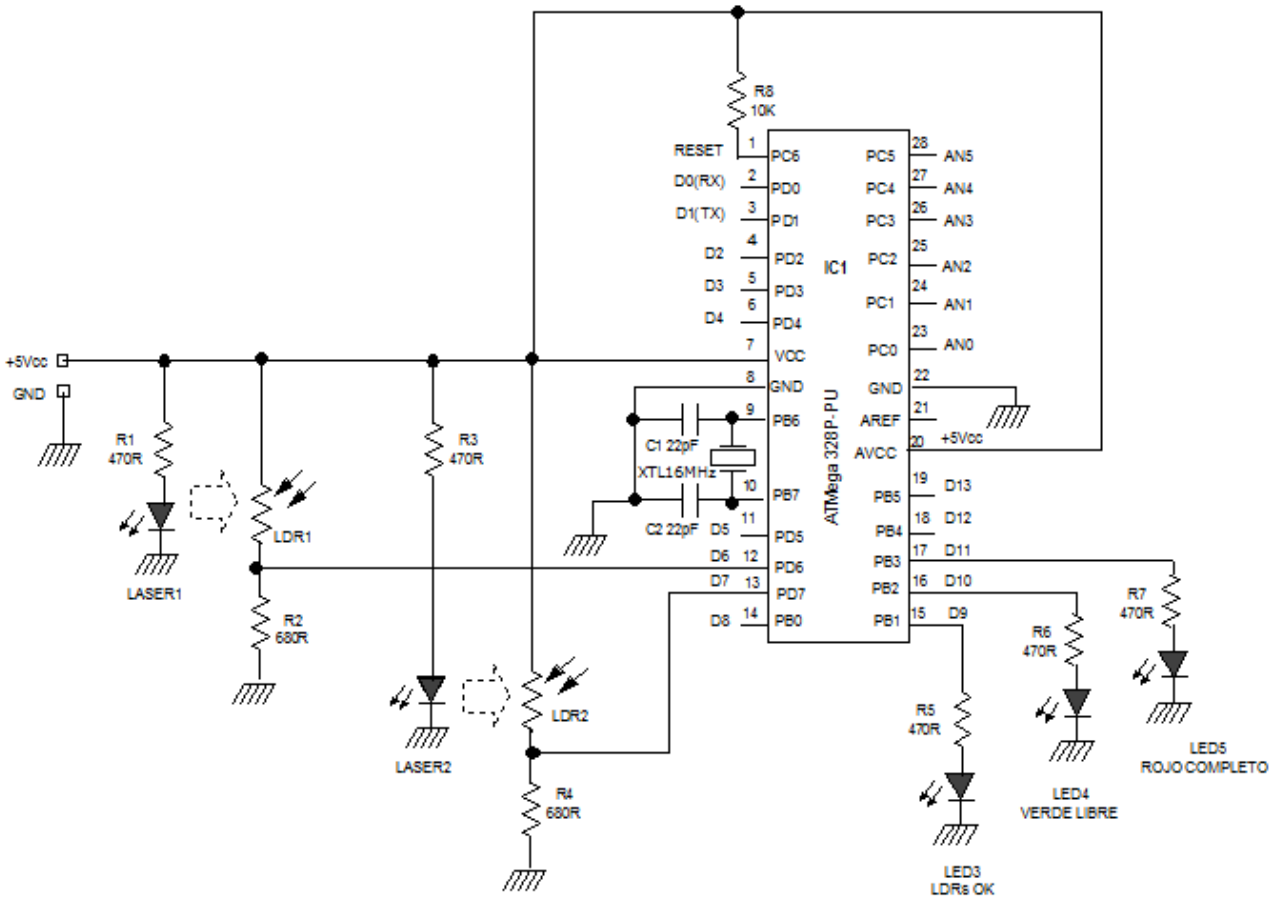
Cuando todas las plazas se han ocupado aparece en la pantalla del monitor serie el aviso de Parking Completo y se enciende el led rojo y se apaga el led verde.

```
/* Control de entrada y salida de vehiculos a un parking*/
  int ledrojo=11;  // pin D11 para el led rojo parking completo
  int ledverde=10; // pin D10 para el led verde plazas libres
  int ledverde2=9; // pin D9 para el led verde de OK fotoresistencias
  int ldrE=7;      //pin D7 de conexion LDR2 de entrada
  int ldrS=6;      //pin D6 de conexion LDR1 de salida
  int contadorE=0; // contador de vehiculos de entrada
  int contadorS=0; // contador de vehiculos de salida
  int contador=0;  //contador total
  int valorE;      //almacena el estado de la lDR2 de entrada
  int valorS;      //almacena el estado de la LDR1 de salida

void setup() {
  Serial.begin (9600);         //configuramos el monitor serie
  pinMode (ledrojo, OUTPUT);   //declaramos el pin led rojo de salida
  pinMode (ledverde, OUTPUT);  //declaramos el pin led verde de salida
  pinMode (ldrE, INPUT);       //declaramos el pin ldrE de entrada
  pinMode (ldrS, INPUT);       // declaramos el pin ldrS de salida
  pinMode (ledverde2, OUTPUT); //declaramos este pin para señalizar barrera
LDR OK
}
void loop() {

  valorE=digitalRead(ldrE);    //se lee el valor de la LDR de entrada
  if (valorE==LOW) {           // si la LDR de entrada detecta corte de luz
    contadorE++;               //almacena cada vez que se corte la luz
    delay(1000);               //espera un segundo
  }
   valorS=digitalRead(ldrS);   //se lee el valor de la LDR de salida
   if (valorS==LOW) {          //si la LDR de salida detecta corte de luz
    contadorS++;               //almacena cada vez que se corte la luz
    delay(1000);               //esperamos un segundo
  }
  if (valorE==HIGH && valorS==HIGH) { //si las dos LDRs estan activas
    digitalWrite(ledverde2, HIGH);  //enciende el lED verde de OK
  }
  else {                            //de lo contrario
    digitalWrite(ledverde2, LOW);  //apaga el LED verde de OK
  }
 contador=contadorE-contadorS;  //contador es la diferencia entre contadorE y
contadorS
 if (contador==200) {        //si contador es igual a 200 maximo de plazas
  Serial.println ("ATENCION!! PARKING COMPLETO"); // se visualiza en el PC
  delay(1000);                    //esperamos un segundo
  digitalWrite (ledrojo, HIGH);   //se enciende el led rojo de completo
  delay(1000);                    //esperamos un segundo
```

```
  digitalWrite (ledverde, LOW);   //se apaga el led verde de libre
  delay(1000);                    //esperamos un segundo
}
  else {                          // de lo contrario
    digitalWrite (ledverde, HIGH);   //enciende el led verde plazas libres
    digitalWrite (ledrojo, LOW);    //apaga el led rojo de completo
    delay(1000);                       //esperamos un segundo
    Serial.println ("PLAZAS LIBRES:"); //visualiza en el PC
    Serial.println (200-contador);   //visualiza en el PC las plazas libres
    delay(1000);                     //esperamos un segundo
   }
}
```




CONTROL DE ENTRADA Y SALIDA DE VEHICULOS EN UN PARKING PROGRAMADO EN ARDUINO

| Plano: | Fecha: 04/12/2022 | Nº de Hojas: 1/1 |
|---|---|---|
| P-0126 | Dibujado: Jose M. Castillo Castillo | |

## Programa 12. Contador de 00-99 con aviso acústico de cambio

En esta programación se ha tenido que hacer una estructura repetitiva con la instrucción de control **while**, que es aquella en que el cuerpo del bucle se repite mientras se cumpla una determinada condición. Cuando se ejecuta la instrucción **while**, la primera cosa que sucede es evaluar la condición (una expresión booleana).

Para nuestro caso en esta programación se ha introducido la instrucción **while** para cuando el pulsador no se haya pulsado y otro **while** cuando el pulsador se haya pulsado y entre estas dos instrucciones habrá un contador que empezará a contar desde 00 y cada vez que se presiona el pulsador se mostrará el siguiente número en los dos display de 7 segmentos para finalizará con el número 99.

El bucle **(while(digitalRead(pulsador)==LOW)** sirve para que la función digitalRead lea repetidamente el valor del pin digital en el que está conectado el pulsador (pin D10). Mientras que este pin tenga valor **LOW** significa que el pulsador no se ha presionado. El bucle deja de repetirse cuando se presiona el pulsador, puesto que entra valor **HIGH**, permitiendo que la ejecución continúe en la siguiente instrucción.

La siguiente instrucción es **contador++.** Simplemente incrementa en una unidad el valor de la variable contador. Si ésta tenia 00, pasará a tener 01, si esta tenía 01 pasará a tener 02 y así sucesivamente.

La siguiente instrucción es un condicional **if** que nos permite cuando el contador llegue al número 99 se ponga el contador a 00.

El siguiente bucle está formado por **(while(digitalRead(pulsador)==HIGH**) y sirve para que la ejecución no continue mientras entre valor **HIGH** por el pin digital D10 del pulsador. Esto evita que se cuenten los rebotos y las pulsaciones largas y contará como una única pulsación.

Por último se ha utilizado la instrucción **tone** para que en cada pulsación se produzca unos tonos de aviso en cada cambio de número.

```
/*Programacion de un contador de 00 a 99 con aviso acústico en el cambio*/
int AU=2;    //asignamos la variable AU de las unidades al pin digital 2
int BU=3;    //asignamos la variable BU de las unidades al pin digital 3
int CU=4;    //asignamos la variable CU de las unidades al pin digital 4
int DU=5;    //asignamos la variable DU de las unidades al pin digital 5
int AD=6;    //asignamos la variable AD de las decenas al pin digital 6
int BD=7;    //asignamos la variable BD de las decenas al pin digital 7
int CD=8;    //asignamos la variable CD de las decenas al pin digital 8
int DD=9;    //asignamos la variable DD de las decenas al pin digital 9
const int pulsador=10; //asignamos constante pulsador al pin digital 10
int contador=0;     // asignamos la variable contador a 0
int piezo=A5;       // asignamos la variable piezo al pin analógico A5
int temp=200;       // asignamos la variable temp al valor de 200
int tonos=500;      // asignamos la variable tonos para el retardo 500
int duracion=1000;  // asignamos la variable duración para tonos
int frecuencia=250; // asignamos la vaiable frecuencia para tonos

void setup() {
  pinMode (AU, OUTPUT);       //configuramos el pin AU de salida
  pinMode (BU, OUTPUT);       //configuramos el pin BU de salida
  pinMode (CU, OUTPUT);       //configuramos el pin CU de salida
  pinMode (DU, OUTPUT);       //configuramos el pin DU de salida
  pinMode (AD, OUTPUT);       //configuramos el pin AD de salida
  pinMode (BD, OUTPUT);       //configuramos el pin BD de salida
  pinMode (CD, OUTPUT);       //configuramos el pin CD de salida
  pinMode (DD, OUTPUT);       //configuramos el pin DD de salida
  pinMode (pulsador, INPUT);  //configuramos el pin pulsador de entrada
  pinMode (piezo, OUTPUT);    //configuramos el pin analogico A5 de salida
}
void loop() {

   while (digitalRead(pulsador)==LOW);   //mientras pulsador no se pulse
   contador++;                           //contador +1 cuando se pulsa
   if (contador==100) {                  //si contador llega a 100
   contador=0;                           //pon el contador a 0
   }
   while (digitalRead(pulsador)==HIGH);  //mientras pulsador se pulse

if (contador==1){                       //si contador es igual a 1
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, LOW);
    digitalWrite(CU, LOW);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, LOW);
    digitalWrite(CD, LOW);
    digitalWrite(DD, LOW);
```

```
    delay(temp);
}
if (contador==2){                          //si contador es igual a 2
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, HIGH);
    digitalWrite(CU, LOW);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, LOW);
    digitalWrite(CD, LOW);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==3){                          //si contador es igual a 3
    tone (piezo,frecuencia,duracion);    // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, HIGH);
    digitalWrite(CU, LOW);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, LOW);
    digitalWrite(CD, LOW);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==4){                          //si contador es igual a 4
    tone (piezo, frecuencia,duracion);  // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, LOW);
    digitalWrite(CU, HIGH);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, LOW);
    digitalWrite(CD, LOW);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==5){                          //si contador es igual a 5
    tone (piezo,frecuencia,duracion);    // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, LOW);
    digitalWrite(CU, HIGH);
```

```
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, LOW);
    digitalWrite(CD, LOW);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==6){                          //si contador es igual a 6
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, HIGH);
    digitalWrite(CU, HIGH);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, LOW);
    digitalWrite(CD, LOW);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==7){                          //si contador es igual a 7
    tone (piezo,frecuencia,duracion);    // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, HIGH);
    digitalWrite(CU, HIGH);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, LOW);
    digitalWrite(CD, LOW);
    digitalWrite(DD, LOW);
    delay(temp);
 }
if (contador==8){                         //si contador es igual a 8
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, LOW);
    digitalWrite(CU, LOW);
    digitalWrite(DU, HIGH);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, LOW);
    digitalWrite(CD, LOW);
    digitalWrite(DD, LOW);
    delay(temp);
 }
```

```
if (contador==9){                        //si contador es igual a 9
    tone (piezo,frecuencia,duracion);    // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, LOW);
    digitalWrite(CU, LOW);
    digitalWrite(DU, HIGH);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, LOW);
    digitalWrite(CD, LOW);
    digitalWrite(DD, LOW);
    delay(temp);
 }
if (contador==10) {                     // si contador es igual a 10
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, LOW);
    digitalWrite(CU, LOW);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, LOW);
    digitalWrite(CD, LOW);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==11) {                      // si contador es igual a 11
    tone (piezo,frecuencia,duracion);    // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, LOW);
    digitalWrite(CU, LOW);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, LOW);
    digitalWrite(CD, LOW);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==12) {                      //si contador es igual a 12
    tone (piezo,frecuencia,duracion);    // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, HIGH);
    digitalWrite(CU, LOW);
    digitalWrite(DU, LOW);
    delay(temp);
```

```
    digitalWrite(AD, HIGH);
    digitalWrite(BD, LOW);
    digitalWrite(CD, LOW);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==13) {                    //si contador es igual a 13
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, HIGH);
    digitalWrite(CU, LOW);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, LOW);
    digitalWrite(CD, LOW);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==14) {                    //si contador es igual a 14
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, LOW);
    digitalWrite(CU, HIGH);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, LOW);
    digitalWrite(CD, LOW);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==15) {                     //si contadro es igual a 15
    tone (piezo,frecuencia,duracion);    // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, LOW);
    digitalWrite(CU, HIGH);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, LOW);
    digitalWrite(CD, LOW);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==16) {                     //si contador es igual a 16
    tone (piezo,frecuencia,duracion);    // tono de aviso de cambio numero
```

```
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, HIGH);
    digitalWrite(CU, HIGH);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, LOW);
    digitalWrite(CD, LOW);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==17) {                     //si contador es igual a 17
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, HIGH);
    digitalWrite(CU, HIGH);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, LOW);
    digitalWrite(CD, LOW);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==18) {                     //si contador es igual a 18
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, LOW);
    digitalWrite(CU, LOW);
    digitalWrite(DU, HIGH);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, LOW);
    digitalWrite(CD, LOW);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==19) {                     // si contador es igual a 19
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, LOW);
    digitalWrite(CU, LOW);
    digitalWrite(DU, HIGH);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, LOW);
```

```
    digitalWrite(CD, LOW);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==20) {                     //si contador es igual a 20
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, LOW);
    digitalWrite(CU, LOW);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, HIGH);
    digitalWrite(CD, LOW);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==21) {                     //si contador es igual a 21
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, LOW);
    digitalWrite(CU, LOW);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, HIGH);
    digitalWrite(CD, LOW);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==22) {                     //si contador es igual a 22
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, HIGH);
    digitalWrite(CU, LOW);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, HIGH);
    digitalWrite(CD, LOW);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==23) {                     //si contador es igual a 23
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
```

```
    digitalWrite(BU, HIGH);
    digitalWrite(CU, LOW);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, HIGH);
    digitalWrite(CD, LOW);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==24) {                     //si contador es igual a 24
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, LOW);
    digitalWrite(CU, HIGH);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, HIGH);
    digitalWrite(CD, LOW);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==25) {                     //si contador es igual a 25
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, LOW);
    digitalWrite(CU, HIGH);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, HIGH);
    digitalWrite(CD, LOW);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==26) {                     //si contador es igual a 26
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, HIGH);
    digitalWrite(CU, HIGH);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, HIGH);
    digitalWrite(CD, LOW);
    digitalWrite(DD, LOW);
```

```
    delay(temp);
}
if (contador==27) {                     //si contador es igual a 27
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, HIGH);
    digitalWrite(CU, HIGH);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, HIGH);
    digitalWrite(CD, LOW);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==28) {                     //si contador es igual a 28
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, LOW);
    digitalWrite(CU, LOW);
    digitalWrite(DU, HIGH);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, HIGH);
    digitalWrite(CD, LOW);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==29) {                     //si contador es igual a 29
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, LOW);
    digitalWrite(CU, LOW);
    digitalWrite(DU, HIGH);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, HIGH);
    digitalWrite(CD, LOW);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==30) {                     //si contador es igual a 30
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, LOW);
    digitalWrite(CU, LOW);
```

```
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, HIGH);
    digitalWrite(CD, LOW);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==31) {                     //si contador es igual a 31
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, LOW);
    digitalWrite(CU, LOW);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, HIGH);
    digitalWrite(CD, LOW);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==32) {                     //si contador es igual a 32
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, HIGH);
    digitalWrite(CU, LOW);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, HIGH);
    digitalWrite(CD, LOW);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==33) {                     //si contador es igual a 33
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, HIGH);
    digitalWrite(CU, LOW);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, HIGH);
    digitalWrite(CD, LOW);
    digitalWrite(DD, LOW);
    delay(temp);
}
```

```
if (contador==34) {                    //si contador es igual a 34
    tone (piezo,frecuencia,duracion);  // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, LOW);
    digitalWrite(CU, HIGH);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, HIGH);
    digitalWrite(CD, LOW);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==35) {                    //si contador es igual a 35
    tone (piezo,frecuencia,duracion);  // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, LOW);
    digitalWrite(CU, HIGH);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, HIGH);
    digitalWrite(CD, LOW);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==36) {                    //si contador es igual a 36
    tone (piezo,frecuencia,duracion);  // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, HIGH);
    digitalWrite(CU, HIGH);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, HIGH);
    digitalWrite(CD, LOW);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==37) {                    //si contador es igual a 37
    tone (piezo,frecuencia,duracion);  // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, HIGH);
    digitalWrite(CU, HIGH);
    digitalWrite(DU, LOW);
    delay(temp);
```

```
    digitalWrite(AD, HIGH);
    digitalWrite(BD, HIGH);
    digitalWrite(CD, LOW);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==38) {                     //si contador es igual a 38
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, LOW);
    digitalWrite(CU, LOW);
    digitalWrite(DU, HIGH);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, HIGH);
    digitalWrite(CD, LOW);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==39) {                     //si contador es igual a 39
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, LOW);
    digitalWrite(CU, LOW);
    digitalWrite(DU, HIGH);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, HIGH);
    digitalWrite(CD, LOW);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==40) {                     //si contador es igual a 40
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, LOW);
    digitalWrite(CU, LOW);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, LOW);
    digitalWrite(CD, HIGH);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==41) {                     //si contador es igual a 41
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
```

```
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, LOW);
    digitalWrite(CU, LOW);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, LOW);
    digitalWrite(CD, HIGH);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==42) {                     //si contador es igual a 42
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, HIGH);
    digitalWrite(CU, LOW);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, LOW);
    digitalWrite(CD, HIGH);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==43) {                     //si contador es igual a 43
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, HIGH);
    digitalWrite(CU, LOW);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, LOW);
    digitalWrite(CD, HIGH);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==44) {                     //si contador es igual a 44
    tone (piezo,frecuencia,duracion);  // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, LOW);
    digitalWrite(CU, HIGH);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, LOW);
```

```
    digitalWrite(CD, HIGH);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==45) {                     //si contador es igual a 45
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, LOW);
    digitalWrite(CU, HIGH);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, LOW);
    digitalWrite(CD, HIGH);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==46) {                     //si contador es igual a 46
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, HIGH);
    digitalWrite(CU, HIGH);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, LOW);
    digitalWrite(CD, HIGH);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==47) {                     //si contador es igual a 47
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, HIGH);
    digitalWrite(CU, HIGH);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, LOW);
    digitalWrite(CD, HIGH);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==48) {                     //si contador es igual a 48
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
```

```
    digitalWrite(BU, LOW);
    digitalWrite(CU, LOW);
    digitalWrite(DU, HIGH);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, LOW);
    digitalWrite(CD, HIGH);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==49) {                     //si contador es igual a 49
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, LOW);
    digitalWrite(CU, LOW);
    digitalWrite(DU, HIGH);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, LOW);
    digitalWrite(CD, HIGH);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==50) {                     //si contador es igual a 50
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, LOW);
    digitalWrite(CU, LOW);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, LOW);
    digitalWrite(CD, HIGH);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==51) {                     //si contador es igual a 51
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, LOW);
    digitalWrite(CU, LOW);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, LOW);
    digitalWrite(CD, HIGH);
    digitalWrite(DD, LOW);
```

```
    delay(temp);
}
if (contador==52) {                     //si contador es igual a 52
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, HIGH);
    digitalWrite(CU, LOW);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, LOW);
    digitalWrite(CD, HIGH);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==53) {                     //si contador es igual a 53
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, HIGH);
    digitalWrite(CU, LOW);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, LOW);
    digitalWrite(CD, HIGH);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==54) {                     //si contador es igual a 54
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, LOW);
    digitalWrite(CU, HIGH);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, LOW);
    digitalWrite(CD, HIGH);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==55) {                     //si contador es igual a 55
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, LOW);
    digitalWrite(CU, HIGH);
```

```
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, LOW);
    digitalWrite(CD, HIGH);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==56) {                     //si contador es igual a 56
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, HIGH);
    digitalWrite(CU, HIGH);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, LOW);
    digitalWrite(CD, HIGH);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==57) {                     //si contador es igual a 57
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, HIGH);
    digitalWrite(CU, HIGH);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, LOW);
    digitalWrite(CD, HIGH);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==58) {                     //si contador es igual a 58
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, LOW);
    digitalWrite(CU, LOW);
    digitalWrite(DU, HIGH);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, LOW);
    digitalWrite(CD, HIGH);
    digitalWrite(DD, LOW);
    delay(temp);
}
```

```
if (contador==59) {                     //si contador es igual a 59
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, LOW);
    digitalWrite(CU, LOW);
    digitalWrite(DU, HIGH);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, LOW);
    digitalWrite(CD, HIGH);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==60) {                     //si contador es igual a 60
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, LOW);
    digitalWrite(CU, LOW);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, HIGH);
    digitalWrite(CD, HIGH);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==61) {                     //si contador es igual a 61
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, LOW);
    digitalWrite(CU, LOW);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, HIGH);
    digitalWrite(CD, HIGH);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==62) {                     //si contador es igual a 62
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, HIGH);
    digitalWrite(CU, LOW);
    digitalWrite(DU, LOW);
    delay(temp);
```

```
    digitalWrite(AD, LOW);
    digitalWrite(BD, HIGH);
    digitalWrite(CD, HIGH);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==63) {                     //si contador es igual a 63
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, HIGH);
    digitalWrite(CU, LOW);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, HIGH);
    digitalWrite(CD, HIGH);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==64) {                     //si contador es igual a 64
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, LOW);
    digitalWrite(CU, HIGH);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, HIGH);
    digitalWrite(CD, HIGH);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==65) {                     //si contador es igual a 65
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, LOW);
    digitalWrite(CU, HIGH);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, HIGH);
    digitalWrite(CD, HIGH);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==66) {                     //si contador es igual a 66
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
```

```
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, HIGH);
    digitalWrite(CU, HIGH);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, HIGH);
    digitalWrite(CD, HIGH);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==67) {                     //si contador es igual a 67
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, HIGH);
    digitalWrite(CU, HIGH);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, HIGH);
    digitalWrite(CD, HIGH);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==68) {                     //si contador es igual a 68
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, LOW);
    digitalWrite(CU, LOW);
    digitalWrite(DU, HIGH);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, HIGH);
    digitalWrite(CD, HIGH);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==69) {                     //si contador es igual a 69
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, LOW);
    digitalWrite(CU, LOW);
    digitalWrite(DU, HIGH);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, HIGH);
```

```
    digitalWrite(CD, HIGH);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==70) {                     //si contador es igual a 70
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, LOW);
    digitalWrite(CU, LOW);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, HIGH);
    digitalWrite(CD, HIGH);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==71) {                     //si contador es igual a 71
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, LOW);
    digitalWrite(CU, LOW);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, HIGH);
    digitalWrite(CD, HIGH);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==72) {                     //si contador es igual a 72
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, HIGH);
    digitalWrite(CU, LOW);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, HIGH);
    digitalWrite(CD, HIGH);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==73) {                     //si contador es igual a 73
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
```

```
    digitalWrite(BU, HIGH);
    digitalWrite(CU, LOW);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, HIGH);
    digitalWrite(CD, HIGH);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==74) {                     //si contador es igual a 74
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, LOW);
    digitalWrite(CU, HIGH);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, HIGH);
    digitalWrite(CD, HIGH);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==75) {                     //si contador es igual a 75
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, LOW);
    digitalWrite(CU, HIGH);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, HIGH);
    digitalWrite(CD, HIGH);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==76) {                     //si contador es igual a 76
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, HIGH);
    digitalWrite(CU, HIGH);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, HIGH);
    digitalWrite(CD, HIGH);
    digitalWrite(DD, LOW);
```

```
    delay(temp);
}
if (contador==77) {                     //si contador es igual a 77
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, HIGH);
    digitalWrite(CU, HIGH);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, HIGH);
    digitalWrite(CD, HIGH);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==78) {                     //si contador es igual a 78
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, LOW);
    digitalWrite(CU, LOW);
    digitalWrite(DU, HIGH);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, HIGH);
    digitalWrite(CD, HIGH);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==79) {                     //si contador es igual a 79
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, LOW);
    digitalWrite(CU, LOW);
    digitalWrite(DU, HIGH);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, HIGH);
    digitalWrite(CD, HIGH);
    digitalWrite(DD, LOW);
    delay(temp);
}
if (contador==80) {                     //si contador es igual a 80
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, LOW);
    digitalWrite(CU, LOW);
```

```
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, LOW);
    digitalWrite(CD, LOW);
    digitalWrite(DD, HIGH);
    delay(temp);
}
if (contador==81) {                     //si contador es igual a 81
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, LOW);
    digitalWrite(CU, LOW);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, LOW);
    digitalWrite(CD, LOW);
    digitalWrite(DD, HIGH);
    delay(temp);
}
if (contador==82) {                     //si contador es igual a 82
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, HIGH);
    digitalWrite(CU, LOW);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, LOW);
    digitalWrite(CD, LOW);
    digitalWrite(DD, HIGH);
    delay(temp);
}
if (contador==83) {                     //si contador es igual a 83
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, HIGH);
    digitalWrite(CU, LOW);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, LOW);
    digitalWrite(CD, LOW);
    digitalWrite(DD, HIGH);
    delay(temp);
}
```

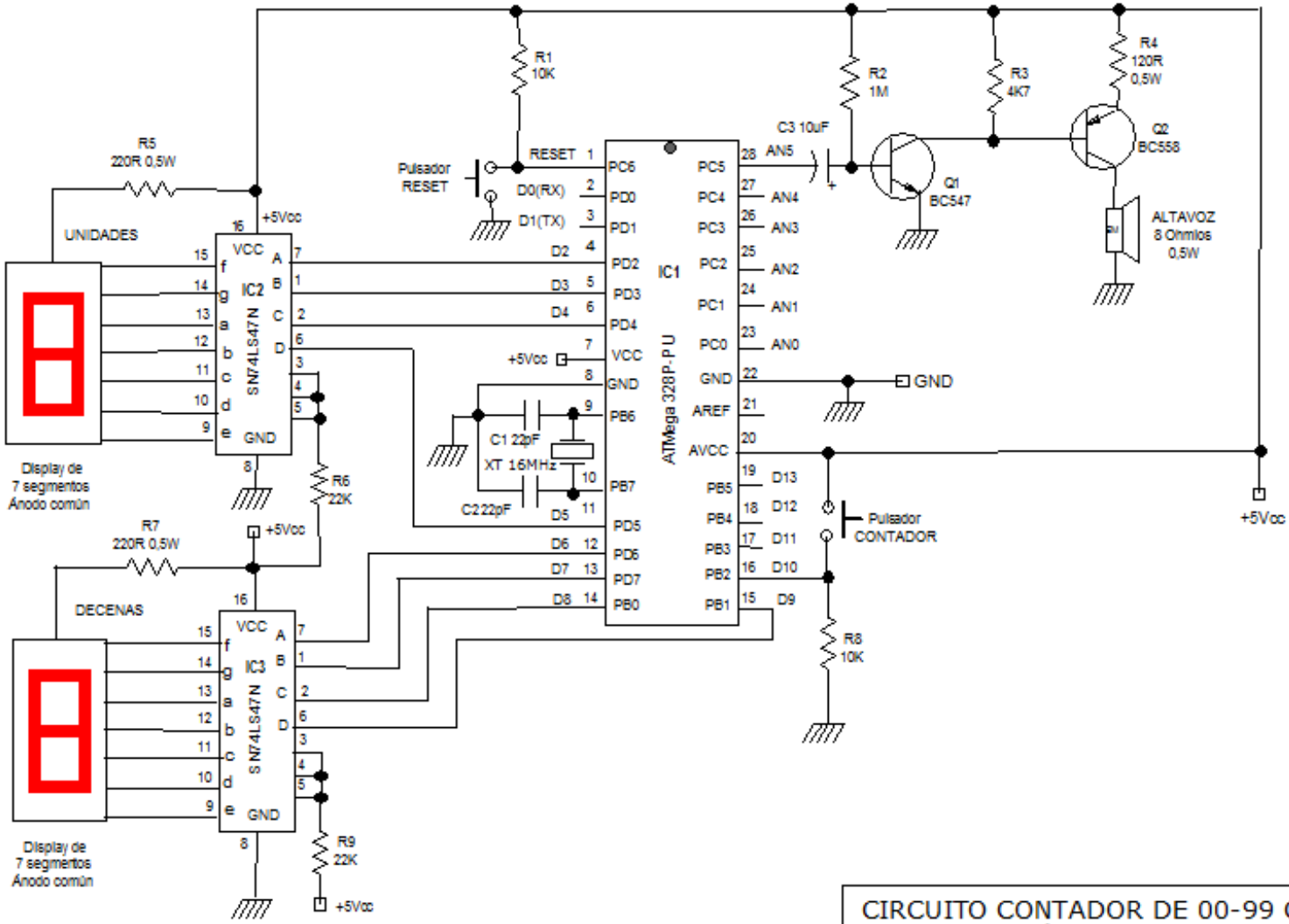
```
if (contador==84) {                    //si contador es igual a 84
    tone (piezo,frecuencia,duracion);  // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, LOW);
    digitalWrite(CU, HIGH);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, LOW);
    digitalWrite(CD, LOW);
    digitalWrite(DD, HIGH);
    delay(temp);
}
if (contador==85) {                    //si contador es igual a 85
    tone (piezo,frecuencia,duracion);  // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, LOW);
    digitalWrite(CU, HIGH);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, LOW);
    digitalWrite(CD, LOW);
    digitalWrite(DD, HIGH);
    delay(temp);
}
if (contador==86) {                    //si contador es igual a 86
    tone (piezo,frecuencia,duracion);  // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, HIGH);
    digitalWrite(CU, HIGH);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, LOW);
    digitalWrite(CD, LOW);
    digitalWrite(DD, HIGH);
    delay(temp);
}
if (contador==87) {                    //si contador es igual a 87
    tone (piezo,frecuencia,duracion);  // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, HIGH);
    digitalWrite(CU, HIGH);
    digitalWrite(DU, LOW);
    delay(temp);
```

```
    digitalWrite(AD, LOW);
    digitalWrite(BD, LOW);
    digitalWrite(CD, LOW);
    digitalWrite(DD, HIGH);
    delay(temp);
}
if (contador==88) {                     //si contador es igual a 88
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, LOW);
    digitalWrite(CU, LOW);
    digitalWrite(DU, HIGH);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, LOW);
    digitalWrite(CD, LOW);
    digitalWrite(DD, HIGH);
    delay(temp);
}
if (contador==89) {                     //si contador es igual a 89
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, LOW);
    digitalWrite(CU, LOW);
    digitalWrite(DU, HIGH);
    delay(temp);
    digitalWrite(AD, LOW);
    digitalWrite(BD, LOW);
    digitalWrite(CD, LOW);
    digitalWrite(DD, HIGH);
    delay(temp);
}
if (contador==90) {                     //si contador es igual a 90
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, LOW);
    digitalWrite(CU, LOW);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, LOW);
    digitalWrite(CD, LOW);
    digitalWrite(DD, HIGH);
    delay(temp);
}
if (contador==91) {                     //si contador es igual a 91
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
```

```
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, LOW);
    digitalWrite(CU, LOW);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, LOW);
    digitalWrite(CD, LOW);
    digitalWrite(DD, HIGH);
    delay(temp);
}
if (contador==92) {                     //si contador es igual a 92
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, HIGH);
    digitalWrite(CU, LOW);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, LOW);
    digitalWrite(CD, LOW);
    digitalWrite(DD, HIGH);
    delay(temp);
}
if (contador==93) {                     //si contador es igual a 93
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, HIGH);
    digitalWrite(CU, LOW);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, LOW);
    digitalWrite(CD, LOW);
    digitalWrite(DD, HIGH);
    delay(temp);
}
if (contador==94) {                     //si contador es igual a 94
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, LOW);
    digitalWrite(CU, HIGH);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, LOW);
```

```
    digitalWrite(CD, LOW);
    digitalWrite(DD, HIGH);
    delay(temp);
}
if (contador==95) {                     //si contador es igual a 95
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, LOW);
    digitalWrite(CU, HIGH);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, LOW);
    digitalWrite(CD, LOW);
    digitalWrite(DD, HIGH);
    delay(temp);
}
if (contador==96) {                     //si contador es igual a 96
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
    digitalWrite(BU, HIGH);
    digitalWrite(CU, HIGH);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, LOW);
    digitalWrite(CD, LOW);
    digitalWrite(DD, HIGH);
    delay(temp);
}
if (contador==97) {                     //si contador es igual a 97
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, HIGH);
    digitalWrite(CU, HIGH);
    digitalWrite(DU, LOW);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, LOW);
    digitalWrite(CD, LOW);
    digitalWrite(DD, HIGH);
    delay(temp);
}
if (contador==98) {                     //si contador es igual a 98
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, LOW);
```

```
    digitalWrite(BU, LOW);
    digitalWrite(CU, LOW);
    digitalWrite(DU, HIGH);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, LOW);
    digitalWrite(CD, LOW);
    digitalWrite(DD, HIGH);
    delay(temp);
}
if (contador==99) {                     //si contador es igual a 99
    tone (piezo,frecuencia,duracion);   // tono de aviso de cambio numero
    delay(tonos);
    digitalWrite(AU, HIGH);
    digitalWrite(BU, LOW);
    digitalWrite(CU, LOW);
    digitalWrite(DU, HIGH);
    delay(temp);
    digitalWrite(AD, HIGH);
    digitalWrite(BD, LOW);
    digitalWrite(CD, LOW);
    digitalWrite(DD, HIGH);
    delay(temp);
}
}
```




# CIRCUITO CONTADOR DE 00-99 CON AVISO ACÚSTICO DE CAMBIO.

| Plano: | Fecha: 21/12/2023 | Nº de Hojas: 1/1 |
|---|---|---|
| P-0136 | Dibujado: Jose M. Castillo Castillo | |