UNIT 3: EXPRESIONES, TEMPORIZACIONES Y SONIDO

OBJETIVOS

En primer lugar vas a profundizar en el concepto de lo que son variables, constantes y expresiones en general. De algo de esto ya hablábamos en la unidad anterior. Ahora toca asentar esos conocimientos.

También te voy a presentar nuevas funciones del lenguaje de programación Arduino. Son funciones que te permitirán generar temporizaciones y sonidos. Con ellas podrás diseñar programas bastante más versátiles y llamativos.

TEORÍA

- EXPRESIONES.
  - Constantes.
  - Variables.
  - Alcance de las variables.
  - Operaciones.
- TEMPORIZACIONES
  - Función: delayMicroseconds()
  - Función: delay()
  - Función: micros()
  - Función: millis()
- EL SONIDO
- FUNCIONES DE SONIDO

PRÁCTICAS

- EJEMPLO 1: Intermitencia
- EJEMPLO 2: Juego de luces
- EJEMPLO 3: Semáforo V1
- EJEMPLO 4: Timbre electrónico
- EJEMPLO 5: Melodias
- EJEMPLO 6: Semáforo V2

"The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein."

PRACTICE MATERIALS

- *A USB cable Ordenador portátil o de sobremesa.*
- *Entorno de trabajo Arduino IDE que se incluye en el material complementario y que se supone instalado y configurado.*
- *Tarjeta controladora Arduino UNO*
- *Cable USB.*

"The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein."

TABLE OF CONTENTS

TEORÍA **4**

1. EXPRESIONES 4
A. CONSTANTES 4
B. VARIABLES 5
C. ALCANCE DE LAS VARIABLES 8
D. OPERACIONES 9
2. TEMPORIZACIONES 11
A. THE DELAYMICROSECONDS FUNCTION 11
B. FUNCIÓN DELAY() 11
C. FUNCIÓN MICROS() 12
D. FUNCIÓN MILLIS() 12
3. EL SONIDO 12
4. SOUND FUNCTIONS 15

PRÁCTICA **17**

1. EJEMPLO 1: INTERMITENCIA 17
2. EJEMPLO 2: JUEGO DE LUCES 18
3. EJEMPLO 3: SEMÁFORO V1 18
4. EJEMPLO 4: TIMBRE ELECTRÓNICO 19
5. EJEMPLO 5: MELODÍAS 19
6. EJEMPLO 6: SEMÁFORO V2 19

REFERENCIAS **20**

"The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein."

TEORÍA

1. EXPRESIONES

A estas alturas habrás observado que las funciones que has estudiado hasta el momento, y la mayoría de las funciones del lenguaje Arduino que te quedan por ver, necesitan de uno o más parámetros u operadores para su funcionamiento. Van cerrados entre los paréntesis de la función y separados entre sí mediante comas “,”.

Haciendo un repaso:

- **pinMode(pin, mode)**: Hay que indicar el número de la patilla a la que haces referencia y si es de entrada o de salida.
- **digitalRead(pin)**: Hay que indicar el número de la patilla que deseas leer.
- **digitalWrite(pin,value)**: Hay que indicar el número de patilla por la que quieres sacar el valor.

Estos parámetros los puedes expresar de diferentes formas, de hecho lo hiciste en algunos de los ejemplos de la Unidad anterior. Llamaremos “expresión” a la forma en que vamos a indicar esos parámetros.

Básicamente hay tres:

- **Constante**: El valor del parámetro lo proporcionas directamente en la propia función. Por ejemplo, digitalRead(4), lee directamente el estado de la patilla de entrada número 4.
- **Variable**: El valor del parámetro lo has definido previamente en una variable. Por ejemplo, digitalRead(Pulsador), lee la patilla de entrada que está definida en la variable “Pulsador”. El controlador debe localizar esa variable en su memoria RAM y extraer el número de la patilla a leer. Si no la localiza se producirá un error.
- **Operación**: El valor del parámetro se obtiene como consecuencia de realizar cualquier tipo de operación aritmético / lógica entre variables y/o constantes. Por ejemplo, digitalRead((1+1)*2), lee el estado de la patilla que resulta de calcular (1+1)*2, la 4.

A. CONSTANTES

Cada vez que el controlador ejecute esta instrucción, siempre se lee la misma patilla, la 4. Este hecho NO se puede variar dinámicamente durante la ejecución del programa. La única forma de cambiarlo es que modifiques el programa y lo vuelvas a grabar nuevamente en la memoria del controlador.

Otro ejemplo. Imagina que debes realizar un programa para calcular la longitud de una circunferencia en función de un diámetro determinado. La ecuación a resolver sería: $l = d * \pi$, donde:.

l = longitud de la circunferencia. **d** = diámetro de la circunferencia **π** = 3.141592

"The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein."

A la vista de este ejemplo, tú mismo puedes responder:

¿Quién es la constante? _______________ ¿Y la variable? __________________

Las constantes son datos que forman parte del propio programa ya que se encuentran integrados en las propias instrucciones o funciones del mismo. Se guardan en la memoria de programa FLASH del controlador. Por lo tanto, no se pueden modificar (salvo si grabamos de nuevo el programa), no son volátiles y se mantienen incluso cuando se desconecta la tensión de alimentación.

Otros ejemplos:

digitalRead(12); *//leer pin 12*

digitalWrite(6,HIGH); *//poner pin 6 a salida "1"*

pinMode(9,OUTPUT); *//Configurar pin 9 como salida*

B. VARIABLES

Las variables son espacios que se reservan en la memoria RAM del controlador para guardar diferentes tipos de datos. Te las puedes imaginar como a una serie de cajas o contenedores donde puedes guardar cualquier información para usarla posteriormente siempre que te haga falta.

Recuerda. La memoria RAM es una memoria que, de la forma adecuada, la puedes leer y escribir tantas veces como quieras. Su contenido SI se puede modificar dinámicamente durante la ejecución de un programa. Es volátil y todo su contenido desaparece cuando falta la tensión de alimentación.

En algunos de los ejemplos anteriores, ya has usado variables. Primero tenían que ser declaradas para poder ser posteriormente usadas. La declaración se realiza asignándole un nombre y si acaso un contenido, según la siguiente sintaxis:

type name = value

type: Establece el tipo de información que va a contener la variable. Puede ser números, letras o textos. Enseguida los verás..

name: Es el nombre que asignas a esa variable, a ese contenedor, y el que deberás emplear en lo sucesivo para hacer uso de su contenido. Lo ideal es que no emplees nombres muy largos y que te recuerden un poco el tipo de información que contienen. Debe empezar siempre por una letra, no por un número y no debe haber espacios en blanco.

value: Es el contenido o valor que metemos en la variable. Es opcional. Puedes crear una variable vacía, sin nada. En este caso haces una reserva en la memoria RAM del controlador para meterle un valor cuando lo creas oportuno en tu programa.

"The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein."

En la siguiente tabla tienes el resumen de los tipos más corrientes de variables:

TYPE	BYTES	DESCRIPTION	EJEMPLOS
char	1	Almacena un valor del tipo carácter (8 bits) que debe ir entre comilla simple (‘). Es con signo y codifica números del rango -128 a +127..	char N = ‘1’ char letter =‘A’
byte	1	Almacena un número de 8 bits sin signoen el rango de 0 a 255	byte C = 23 byte result = C +18
int	2	Es el tipo por defecto. Almacena un número de 16 bits con signo en el rango de -32768 a +32767	int C = 2453 int value = C * 10 number = 2453
unsigned int or word	2	Almacena un número de 16 bits sin signo	unsigned int valor = 60000 word valor = 4328 * 10
long	4	y con un rango de 0 a 65535	long counter = -123456789
unsigned long	4	Almacena un número de 32 bits con signo en el rango de -2.147.483.648 a +2.147.483.647	unsigned long speedOfLight = 299792468
float	4	Almacena un número de 32 bits sin signo en el rango de 0 a 4.292.967.295	float pi = 3.1416 float L = 4 * pi

A la vista está que, según el tipo de variable que definas, consumirás una mayor o menor cantidad de memoria RAM. Te aconsejo que definas correctamente las variables según el tipo de información que vayan a contener. Por ejemplo, si defines una variable del tipo “long”, consumes 4 bytes de memoria RAM. Su empleo es poco eficiente si la información que va a contener está comprendida únicamente entre 0 y 255. Mejor la defines del tipo “byte” que consume sólo 1. La memoria RAM es limitada. Si la usas indebidamente puedes llevarte una sorpresa cuando se agote.

Por otro lado si declaras una variable de, por ejemplo tipo byte, y le introduces un valor superior a 255, la variable se desborda y se pierde su información. Es algo parecido a lo que le pasará al cuentakilómetros de un coche cuando se desborda su capacidad. Empieza otra vez desde 0.

Otro tipo de variables son las llamadas “Arrays”. Un array es una colección de variables de cualquiera de los tipos que acabas de estudiar. Para acceder al contenido de una variable en particular, dentro de la colección, debes indicar el índice de la misma. Mira estos ejemplos.

"The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein."

- **int Table[4]**

Crea un array vacío llamado “Tabla” y reserva espacio para guardar 4 números del tipo int. Ocupa por tanto 8 bytes de la memoria RAM del controlador.

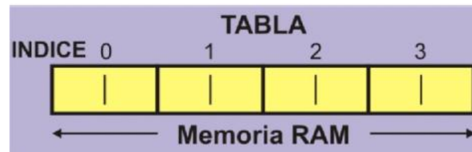

Figure 1

- **int Table[4]={5678,750,1234}**

Crea el array “Tabla” con espacio para guardar 4 números del tipo int, en el cual almacena los tres valores indicados.

Figure 2

- **int Suma = Table[0] + Table[2]**

Extrae del array “Tabla” los valores de la variable 0 (5678) y de la variable 2 (1234). Hace la suma y almacena el resultado (6912) en la variable “Suma”.

- **unsigned int Values[12]**

Crea el array “Valores” que reserva espacio para 12 variables del tipo entero sin signo. Ocupa un total de 24 bytes de la memoria RAM.

- **Values[5] = 12345 * 3**

En la variable número 4 del array “Valores” se introduce el número resultante de hacer el cálculo (37035).

- **char Text[8] = “ARDUINO”**

Crea el array “Texto” del tipo carácter. Ocupa 8 bytes de la memoria RAM en los que se almacena la cadena “ARDUINO\o” (\o es un código de control que indica fin de la cadena).

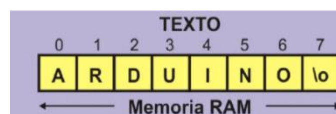

Figure 3

"The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein."

- **char Letter = Text[2]**

Extrae la variable número 2 del array “Text” (letra D) y la almacena en la variable “Letter” de tipo char.

C. ALCANCE DE LAS VARIABLES

Según el lugar del programa en donde declares las variables, estas pueden ser “globales” o bien “locales”. Este asunto de momento no te afecta de forma inmediata, pero puede ser importante cuando empieces a realizar programas de cierta complejidad y tamaño.

Globales: Son las variables que se declaran o definen fuera de todas las funciones que haya en el programa, incluidas las funciones setup() y loop(). Algunos de los ejemplos anteriores empleaban este tipo de variables. Si te fijas en la figura, las variables “Dato” y “Resultado” son globales. Pueden ser usadas por todas las funciones que hubiera en el programa.

Locales: Son variables que se declaran y crean dentro de una determinada función, y sólo pueden ser usadas por ella. Se evita así que una variable que se declara en una función pueda ser utilizada y/o modificada por otra función distinta de ella.

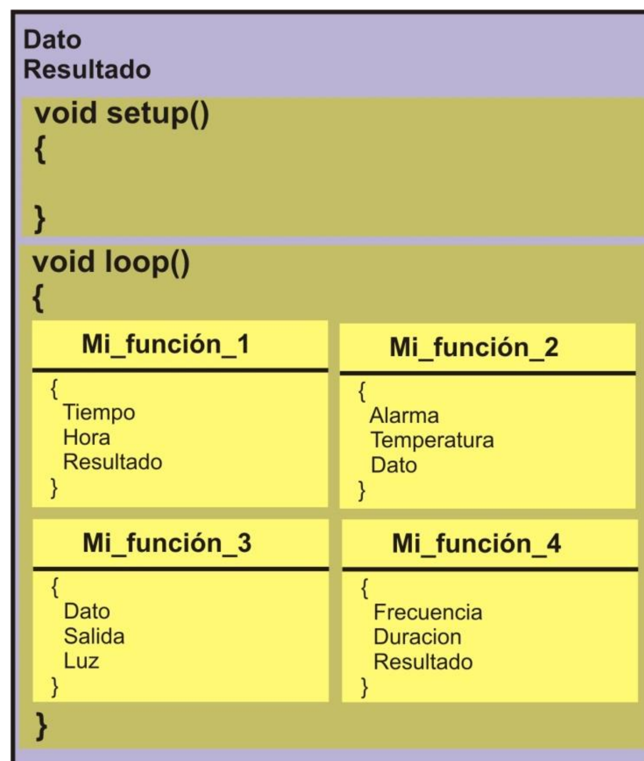

Figure 4

"The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein."

Cuando finaliza la ejecución de una función todas las variables locales desaparecen hasta que volvamos a usar esa misma función. Según la figura, la función “Mi_Función_1” tiene como locales a las variables: “Tiempo”, “Hora” y “Resultado”.

Responde a las siguientes preguntas::

1. La función “Mi_Función_2” ¿puede hacer uso de la variable “Luz”?__________

2. “Mi_Función_1” ¿Puede hacer uso de la variable “Hora”? _________________

3. “Mi_Función_4” ¿Puede hacer uso de la variable “Dato”? ________________

D. OPERACIONES

Seguro que ya has adivinado que tanto los parámetros que acompañan a las sentencias del lenguaje Arduino, como los valores que cargas en las variables, se pueden obtener como resultado de hacer todo tipo de operaciones aritméticas entre variables y/o constantes.

Las operaciones matemáticas típicas son:

= **Adignación**

+ **Suma**

- **Resta**

* **Multiplicación**

/ **Division**

% **Módulo (es el resto de una división entre dos números enteros)**

Al igual que como lo hacemos nosotros, cualquier cálculo se realiza de izquierda a derecha. La multiplicación y la división tienen prioridad sobre la suma o la resta. Puedes usar los paréntesis para establecer la prioridad que desees en una operación. Mira y analiza los siguientes ejemplos:

```
-> 3 + 5 – 2 = 6

-> 20 – 2 × 3 = 14 and not 54

-> (20 - 2) × 3 = 54 and not 14

-> 20 / 2 × 3 = 30

-> 24 / (2 × 3) = 4
```

```
-> (12 - 2) + 5 × 3 = 25 and not 45

-> ((12 - 2) + 5) × 3 = 45 and not 25

->13 % 4=1

-> 7 % 5 =2

-> 5 % 5 = 0
```

"The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein."

Con lo que has visto en este apartado, a partir de ahora podrás expresar los parámetros que emplean algunas funciones del lenguaje Arduino, de una forma mucho más flexible y potente. Fíjate en los siguientes ejemplos y trata de responder a las preguntas

```
byte Light=3;

pinMode(Light×2,OUTPUT);

digitalRead(Light+1);

digitalWrite((Light-1)×3),HIGH);
```

Qué patilla se configura como salida? ______________________________________

¿Qué patilla se lee? ____

¿Qué patilla se pone a nivel “1”? ____

```
byte A = 250;

byte B; byte C;

unsigned int D;

A = A + 3;

B = (A - 200) × 2 / 4;

C = A × 2 + 20;

D = (A × 100) % 2;
```

A = ____________; B = ____________; C = ____________; D = ____________

"The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein."

2. TEMPORIZACIONES

Aunque te pueda parecer paradójico, hay muchas ocasiones en las que interesa que el controlador no haga absolutamente nada útil. Es decir, que pierda una determinada cantidad de tiempo.

El lenguaje de programación Arduino dispone de unas funciones que detienen la ejecución de un programa durante un cierto tiempo establecido por ti. Quizá ahora no le veas sentido, pero espera un poco a los diferentes ejemplos.

A. THE DELAYMICROSECONDS FUNCTION

Realiza una temporización de tantos microsegundos como indiques. Ten en cuenta que: 1 segundo = 1000 milésimas de segundo (mS) y que 1 milésima de segundo = 1000 millonésimas de segundo. Puesto que: 1 segundo = 1.000.000 de millonésimas de segundo (µS), 1 µS = 0.000001 segundos.

Sintaxis:

delayMicroseconds(n);

n: Indica el número de microsegundos que deseas temporizar. Se trata de un número entero sin signo (unsigned int) de 16 bits. En estos momentos el valor más grande que se aconseja para realizar una temporización con cierta precisión

Ejemplos:

A = 100;

delayMicroseconds(A); *//detiene el programa 100 µS = 0.1 mS = 0.0001 seconds*

delayMicroseconds(A×10-100); *//detiene el programa 900 µS = 0.9 mS = 0.0009 seconds*

B. FUNCIÓN DELAY()

Realiza una temporización de tantas milésimas de segundos como indiques. Ten en cuenta que: 1 segundo = 1000 milisegundos, o sea, 1 milisegundo (mS) = 0.001 segundos.

Sintaxis:

delay(n);

n: Indica la cantidad de milésimas de segundo (mS) que deseas temporizar. Es un número largo sin signo de 32 bits, (unsigned long), capaz de representar desde 0 hasta 4.294.967.295 (2^32- 1).

Ejemplos:

A=100; delay(A); *// detiene el programa 100 mS = 0.1 second delay(1000)*
*// detiene el programa 1000 mS = 1 segundo delay(A*600);*
// detiene el programa 60000 mS = 60 seconds = 1 minute

"The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein."

Recuerda. Cuando realizas una temporización, bien sea con la función delayMicroseconds() o bien con la función delay(), el controlador deja de ejecutar instrucciones de tu programa hasta que dicha temporización finaliza.

C. FUNCIÓN MICROS()

Devuelve el número de microsegundos transcurridos desde que el Arduino realizó la secuencia de inicio y comenzó a ejecutar tu programa, hasta el momento actual. Devuelve un valor largo sin signo (unsigned long) capaz de representar aproximadamente unos 70 minutos de funcionamiento. Recuerda que hay 1000 µS en una milésima de segundo (mS) y 1000000 de µS en un segundo.

Sintaxis:

var = micros();

var: Es la variable donde se almacenan los micro segundos (µS) transcurridos desde que se reinició el sistema.

D. FUNCIÓN MILLIS()

Devuelve el número de milésimas de segundos (mS) transcurridos desde que el Arduino realizó la secuencia de inicio y comenzó a ejecutar tu programa, hasta el momento actual. Devuelve un valor largo sin signo (unsigned long) capaz de representar aproximadamente unos 50 días de funcionamiento. Recuerda que hay 1000 mS en una en un segundo.

Sintaxis:

var = millis();

var: Es la variable donde se almacenan las milésimas de segundo (mS) transcurridas desde que se reinició el sistema.

3. EL SONIDO

¿Qué ocurre cuando tiras una piedra a una charca? Que al entrar en contacto con el agua se modifica su presión y esto provoca una serie de ondas que se van desplazando hasta desvanecerse. El sonido es algo parecido. Basta con que un dispositivo sea capaz de vibrar y producir cambios de presión u ondas en el aire. Estas ondas llegan al tímpano de nuestro oído, donde nuestro cerebro las percibe como sonidos. Cuando hablamos o cantamos, nuestras cuerdas vocales son las encargadas de producir esos cambios de presión u ondas que se desplazan por el aire hasta llegar a nuestro tímpano.

También hay otros aparatos, por todos conocidos, capaces de producir cambios de presión en el aire y generar por tanto ondas que transportan el sonido: altavoces, auriculares, zumbadores, sirenas, etc…

"The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein."

Aunque con formas, características y aplicaciones diferentes, todos ellos basan su funcionamiento en tres componentes básicos: membrana, bobina e imán permanente. La membrana puede ser de papel, cartón, plástico, etc… debidamente tratado. Está físicamente unida a la bobina, y esta a su vez está introducida sobre un imán permanente.

Al aplicar una señal eléctrica a la bobina, se crea un campo magnético que interacciona con el campo magnético del imán permanente. Estos campos a veces se atraen entre sí y otras se repelen. Esta atracción o repulsión de la bobina respecto al imán "arrastra" o "empuja" la membrana a la que está unida. Estos movimientos de la membrana producen cambios de presión, vibraciones u ondas. Ya tenemos el sonido viajando por el aire. Además, esas vibraciones u ondas son una fiel reproducción de la señal eléctrica que las originó.

La señal eléctrica que da origen a todo este proceso debe ser forzosamente una señal variable, que transite constantemente de nivel "1" a nivel "0", a una determinada velocidad o frecuencia. No valen señales fijas. Imagina una señal que está permanentemente a nivel "1" o a nivel "0". En este caso, la bobina entra o sale del imán permanente, pero ahí se queda. La membrana no vibra y no se generan ondas sonoras.

El oído humano es capaz de percibir y escuchar vibraciones que van desde los 20 Hz hasta los 20000 Hz (20 KHz) o ciclos por segundo. Esto es la frecuencia o el número de veces por segundo que una señal pasa de "1" a "0".

Las diferentes frecuencias las percibimos como "tonos". Una frecuencia baja produce un sonido con un tono más grave, mientras que cuanto más alta sea la frecuencia más agudo será el tono.

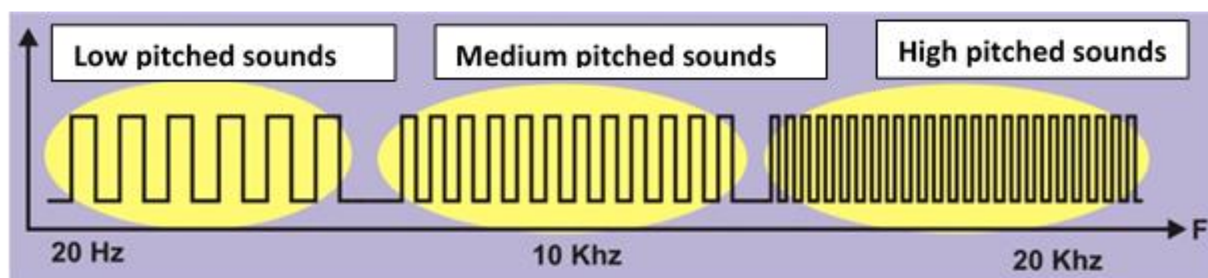

Figure 5

Las vibraciones cuya frecuencia está por debajo de los 20 Hz se denominan "infrasonidos", y por encima de los 20 KHz "ultrasonidos". El oído humano no es capaz de percibir estos sonidos, pero algunos animales sí.

Vamos a hacer unos cálculos. Supón que deseas generar una señal sonora cuya frecuencia es de 100 Hz:

1. A partir de la frecuencia F deseada (100 Hz), calculamos el periodo T, o el tiempo que dura un ciclo

$$T = \frac{1}{F} = \frac{1}{100} = 0.01\ s = 10\ mS = 10000\ \mu S$$

2. Ahora calculamos el semiperiodo S, o lo que es igual, el tiempo que la señal va a estar a nivel "1" y a nivel "0".

$$S = \frac{T}{2} = \frac{0.01}{2} = 0.005\ s = 5\ mS = 5000\ \mu S$$

"The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein."

3. Con los conocimientos que tienes ya podrías confeccionar un programa que genere por el altavoz conectado en la patilla 2, la frecuencia deseada. Estudia la siguiente solución:

```
void setup()
{
pinMode(2,OUTPUT);                  // Output pin 2 (speaker)
}
// T Period for total of 10000 µS at an F frequency of 100 HZ
void loop()
{
digitalWrite(2,HIGH);               //Pin 2 at level "1"
delayMicroseconds(5000);            //Pauses for 5000 µS
digitalWrite(2,LOW);                //Pin 2 at level "0"
delayMicroseconds(5000);            //Pauses for 5000 µS
}
```

Observa que la patilla 2 se mantiene a nivel "1" (HIGH) durante 5000 µS y a nivel "0" (LOW) otro tanto. Es decir, por la patilla 2 se produce un ciclo T de nivel "1" y de nivel "0" que dura 10000 µS. Como el ciclo se repite constantemente, en un segundo caben 100 de esos ciclos (1000000/10000). Esto es la frecuencia F.

Graba el programa y verifica su correcto funcionamiento, que consistirá en un constante (y molesto) sonido a una frecuencia de 100 Hz.

Modifica el programa para obtener sonidos con tonos de diferentes frecuencias F, cuyos periodos T y semiperiodos S habrás calculado previamente en la siguiente tabla.:

F	T	S	F	T	S
50 Hz			2.5 KHz		
200 Hz			5 KHz		
500 Hz			10 KHz		
1 KHz			15 KHz		
2 KHz			20 KHz		

"The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein."

4. SOUND FUNCTIONS

Afortunadamente el lenguaje Arduino de programación te va facilitar enormemente la generación de todo tipo de sonidos. No vas a tener que calcular ni el periodo T ni el semiperiodo S. Simplemente indicarás la frecuencia F deseada y él se encargará de lo demás.

- **Función Tone()**

Genera un sonido con la frecuencia F deseada, sobre la patilla de salida que indiquemos y con la duración que queramos.

Sintaxis:

tone(pin, frequency, duration);

pin: Representa sobre qué patilla de salida vas a sacar el sonido.

frequency: Es un valor entero sin signo (unsigned int) que representa la frecuencia del sonido. Se expresa en Hz (herzios). Recuerda que el oído humano sólo percibe sonidos que van desde los 20 Hz hasta los 20 KHz.

duration: Este parámetro es opcional. Es un valor largo sin signo (unsigned long) que representa, en milisegundos, el tiempo que dura la salida del sonido. Si no se indica, el sonido se mantiene indefinidamente o hasta que se ejecute la función noTone().

Ejemplos:

int Pin=11;

int F = 1200;

int D = 3000;

tone(2,F,D); *// Genera un tono de 1200 Hz durante3 segundos en el pin 2*

tone(6,F*10,500); *// Genera un tono de 12 KHZ durante 0.5 segundos en el pin 2.*

tone(Pin,200); *// Genera un tono de 200 Hz en el pin 11*

- **Función noTone():**

Finaliza la salida del tono que esté sonando en ese momento por la patilla indicada.

Sintaxis:

noTone(pin);

pin: *Representa a la patilla cuyo sonido vas a anular.*

"The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein."

Ejemplo:

```
tone(2,13000);        //Genera un tono de 13 KHz en el pin 2.
….                    //El tono se mantiene.

noTone(2);            //El tono para.
```

"The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein."

PRÁCTICA

1. EJEMPLO 1: Intermitencia

Este es el programa de ejemplo más sencillo que te puedes imaginar. Consiste en producir una intermitencia sobre el led blanco, conectada con la patilla 6. Echa un vistazo a la solución de la figura.

La variable "Tiempo" contiene al valor a temporizar, 500 mS = 0.5 ". La patilla 6, la del led blanco, se configura como salida.

El cuerpo principal del programa, en la función loop(), se limita a activar y desactivar la patilla 6 a intervalos regulares determinados por la variable "Tiempo", 500 mS.

```
int Tiempo = 500;            //Valor de la temporización

//Sentencias iniciales de configuración
void setup()
{
  pinMode(6, OUTPUT);        //La patilla 6 del led blanco se confi
}

void loop()
{
  digitalWrite(6,HIGH);      //El led blanco se ilumina
  delay(Tiempo);             //Temporiza
  digitalWrite(6,LOW);       //El led blanco se apaga
  delay(Tiempo);             //Temporiza
}
```

Figure 6

- **Tu turno**

Con este ejemplo puedes llevar a cabo varias actividades. Te sugiero un par de ellas.:

You can try varying the pause interval using the "*Time*" variable. I suggest you shorten it. Bring it down bit by bit until you get to 10 mS. What do you notice?

1. Puedes probar a variar el intervalo de temporización mediante la variable "Tiempo". Te recomiendo que lo vayas disminuyendo. Vete poco a poco bajándolo hasta llegar a unos 10 mS. ¿Qué observas?

   Aparentemente el led blanco está siempre encendido. Sin embargo no es así. El led se está encendiendo y apagando muy rápidamente. Lo que ocurre es que la retina de nuestro ojo no es capaz de percibir esos cambios tan rápidos y nos da la sensación óptica de que está permanentemente encendido.

   ¡¡ Qué elástico es el tiempo !! Lo que para ti es inmediato para el controlador no lo es. El controlador tiene que esperar 10 mS con el led encendido y otros 10 mS con el led apagado, pero tú no lo ves.

"The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein."

2. En lugar de emplear la función delay() prueba a utilizar la función delaymicroseconds(). Con ella puedes conseguir temporizaciones muy pequeñas, del orden de las millonésimas de segundos (µS), y con bastante precisión.

3. También puedes modificar el programa para emular el funcionamiento de un faro. Por ejemplo, el led blanco emite un destello de 0.1” de duración cada 2”

2. EJEMPLO 2: Juego de luces

Este ejemplo te resultará bastante atractivo. Cuatro leds se van a encender secuencialmente, de uno en uno, empezando por el led blanco y durante un cierto tiempo cada uno. Te dará la sensación de movimiento de derecha a izquierda.

El programa es quizá algo largo, pero muy sencillo. Activa el led blanco, temporiza, lo apaga y activa el led verde. Temporiza, apaga el verde y enciende el ámbar. Temporiza, apaga el ámbar y enciende el rojo. Temporiza, apaga el rojo y el ciclo se vuelve a repetir.

- **Tu turno**

1. Modifica el tiempo con objeto de ir más rápido o más lento en el desplazamiento de derecha a izquierda.

2. Consigue que el desplazamiento vaya de derecha a izquierda y seguidamente de izquierda a derecha.

3. EJEMPLO 3: Semáforo V1

Con lo que has aprendido ya puedes plantearte hacer un pequeño proyecto. Te propongo hacer la simulación del funcionamiento de un sencillo semáforo. Este ejemplo es una primera versión de otros semáforos que irás mejorando poco a poco.

El programa como tal no aporta ninguna nueva función. Simplemente se trata de encender y apagar los leds verde, ámbar y rojo que simulan el semáforo, en la secuencia adecuada y con los tiempos que desees. Sin embargo puedes observar un detalle..

- **Conclusiones**

Fíjate que todas las funciones del programa que establecen la secuencia de funcionamiento del semáforo se han incluido en la función setup(). La función principal loop() está vacía.

Esto es lícito. Ya estudiaste que, aunque es obligatorio ponerla, la función loop() puede no tener ninguna función. Por otra parte, todas las funciones que se incluyan en la función setup(), sólo se ejecutan una vez. Efectivamente, cada vez que quieras que tu semáforo realice una nueva secuencia, tendrás que pulsar el botón RESET de reinicio. Compruébalo.

"The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein."

4. EJEMPLO 4: Timbre electrónico

Otro sencillo proyecto. En esta ocasión vas a simular el funcionamiento de un timbre. Cada vez que se acciona el pulsador el timbre emite un sonido con tres tonos diferentes. Es la primera vez que vas a usar la función tone() y podrás comprobar que es sumamente sencilla. Basta con indicar la patilla de salida, la frecuencia y la duración (opcional).

En la figura tienes la solución al programa. Se genera un primer tono de 1000 Hz durante 0.2". Luego, tras una temporización de 0.4", se genera otro tono de 1500 Hz durante 0.3". Finalmente, 0.4" después se genera un tercer tono de 2000 Hz durante 0.4". Los tonos se obtienen por la patilla 2 del controlador, que es donde está conectado el pequeño altavoz.

Observa también que todas las funciones están incluidas en la función setup(), por lo que la secuencia se ejecuta cada vez que pulsas RESET.

- **Tu turno**

Vas a mejorar el timbre añadiéndole una señal luminosa. Te propongo que cada vez que pulses el RESET, además de la secuencia descrita anteriormente, se encienda el led blanco. Cuando finalice la secuencia, una vez que se ha generado el tercer tono, el led deberá apagarse

5. EJEMPLO 5: Melodías

Para los que saben solfeo (que no es mi caso). Este ejemplo combina una serie de tonos de diferentes frecuencias y duraciones, para generar una conocida melodía. Simplificando te diré que a cada nota musical le corresponde una determina frecuencia. A partir de ellas se obtienen los sostenidos, bemoles y combinaciones de todos ellos. ¡¡ Prueba y experimenta!!

6. EJEMPLO 6: Semáforo V2

Aquí tienes otro pequeño proyecto. Se trata de una versión mejorada del ejemplo del semáforo. Le hemos añadido señales acústicas de aviso especiales para invidentes.

El programa quizá te parezca demasiado largo pero si lo analizas un poco verás que no es complicado. Es muy secuencial. Empieza activando la luz verde al tiempo que genera 6 tonos de 1 KHz y 0.5" cada uno. A continuación la luz ámbar con 6 tonos de 1 KHZ y 0.4" de duración. Finalmente, se enciende la luz roja con un tono de 1KHz durante 6".

Como todas las funciones del programa están incluidas en la función setup(), únicamente se ejecutarán una vez: cada vez que pulses RESET se inicia la secuencia.

- **Tu turno**

Para que a la hora de hacer las comprobaciones, la ejecución del programa no nos resulte muy larga en el tiempo, reconocerás que los tiempos en que se mantienen iluminadas cada luz no es el real, es mucho menor. Te sugiero que localices un semáforo en tu calle o barrio y cronometres los tiempos de encendido para cada luz. También puedes escuchar los tonos que se producen en cada caso y observar si hay intermitencias en las luces y el sonido. Se trata de que modifiques el programa para hacer que tu semáforo simule lo mejor posible a un semáforo real.

"The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein."

REFERENCIAS

BOOKS

[1]. **EXPLORING ARDUINO**, Jeremy Blum, Ed.Willey
[2]. Practical ARDUINO, Jonathan Oxer & Hugh Blemings, Ed.Technology in action
[3]. Programing Arduino, Next Steps, Simon Monk
[4]. Sensores y actuadores, Aplicaciones con ARDUINO, Leonel G.Corona Ramirez, Griselda S. Abarca Jiménez, Jesús Mares Carreño, Ed.Patria
[5]. ARDUINO: Curso práctico de formación, Oscar Torrente Artero, Ed.RC libros
[6]. 30 ARDUINO PROJECTS FOR THE EVIL GENIUS, Simon Monk, Ed. TAB
[7]. Beginning C for ARDUINO, Jack Purdum, Ph.D., Ed.Technology in action
[8]. ARDUINO programing notebook, Brian W.Evans

WEB SITES

[1]. https://www.arduino.cc/
[2]. https://www.prometec.net/
[3]. http://blog.bricogeek.com/
[4]. https://aprendiendoarduino.wordpress.com/
[5]. https://www.sparkfun.com

"The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein."