# UNIDAD 4: TOMA DE DECISIONES Y FUNCIONES DE CONTROL

## OBJETIVOS

Estudiar las funciones que controlan el flujo de la ejecución de un programa. También se llaman “funciones de control” y están disponibles en cualquier lenguaje de programación. Arduino no puede ser menos.

Los ejemplos que has realizado hasta eran programas compuestos por una serie de funciones que se ejecutaban todas ellas, de forma secuencial y desde la primera hasta la última. A partir de ahora los programas van a tener una cierta “inteligencia” y capacidad de decisión. Habrá funciones que se ejecutan o no si se dan determinadas condiciones.

## TEORÍA

- Operadores de comparación
- Operadores booleanos
- Operadores compuestos
- Función: if(…)
- Función: if(…) else
- Función: for(…)
- Función: while(…)
    - Otras formas de while
- Función: switch(…) / case
- Otras funciones de control
    - Función: do … while(…)
    - Función: break
    - Función: return
    - Función: go to

**PRÁCTICA**

- EJEMPLO_6_1: Timbre electrónico V2
- EJEMPLO_6_2: Juego de luces V2
- EJEMPLO_6_3: Semáforo V3
- EJEMPLO_6_4: Baliza electrónica
- EJEMPLO_6_5: Semáforo V4
- EJEMPLO_6_6: Ráfagas
- EJEMPLO_6_7: Timbre electrónico V3
- EJEMPLO_6_8: Contador
- EJEMPLO_6_9: Tiempo

***MATERIALES***

- ✓ 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.

# TABLA DE CONTENIDOS

# TEORÍA

Los seres humanos, algunos más que otros, nos pasamos el día tomando decisiones. Somos capaces de analizar nuestro entorno y situación y, en función de nuestros intereses, sentimientos, capacidad, intuición, obligaciones, etc… tomamos decisiones que nos llevan a actuar de diferentes formas según nos convenga.

Los programas que has desarrollado hasta el momento son programas muysecuenciales. Todas las instrucciones se van ejecutando una tras otra, desde laprimera hasta la última y sin ninguna otra consideración.

Sin embargo, Arduino, al igual que cualquier otro controlador, tiene capacidad de tomar decisiones y ejecutar los programas o tareas apropiadas para cada caso. Ahora bien, te puedes imaginar que Arduino no tiene sentimientos ni intuición. Ni tan siquiera es inteligente. Es una máquina que sólo sabe trabajar con números. Las decisiones se toman en base a ellos: cálculos aritméticos/lógicos, comparaciones entre números, estados de las señales de entrada, valores analógicos leídos de un sensor, etc…

## 1. OPERADORES DE COMPARACIÓN

Ya sabes que Arduino puede hacer operaciones aritméticas como sumar y restar diferentes valores. También sabes que esos valores se puede expresar en forma de constantes o de variables o en una combinación de ambas.

Bueno, pues Arduino también sabe hacer comparaciones entre valores o entre los resultados que ofrecen ciertas funciones. A continuación tienes los diferentes operadores de comparación así como los símbolos que los representan:

| OPERADOR | SIMBOLO | OPERADOR | SIMBOLO |
|---|---|---|---|
| **Igualdad** | = = | **Distinto de** | != |
| **Menor que** | < | **Mayor que** | > |
| **Menor o igual que** | <= | **Mayor igual que** | >= |

Tras realizar una comparación, sea del tipo que sea, solo se obtienen dos resultados posibles:“Verdadero”(“true”) o “Falso” (“false”). Mira y analiza los siguientes ejemplos:

**Suponiendo que...**

```
char Letter = 'J';          int B = 12345;
byte A = 13;                float PI = 3.1416
```

**Entonces...**

```
Letter == 'J'               //True
Letter != 'Q'               //True
18 < A                      //False
A == 8 + 5                  //True
B >= A                      //True
PI * 2 > 8.16               //False
B – 1000 <= A * 12          //False
digitalRead(4)==1           //True if pin 4 is on level "1"
```

## 2. OPERADORES BOOLEANOS

Incluso es posible relacionar entre sí varias de las comparaciones anteriores. Para ello Arduino dispone de tres operadores lógicos, también llamados “booleanos”, que seguro que te recuerdan a algunos de los ejemplos que hiciste. Aquí los tienes junto con sus correspondientes símbolos:

| OPERADOR | SIMBOLO |
|---|---|
| **NOT** | ! |
| **AND** | && |
| **OR** | 11 |

De igual manera, como consecuencia de relacionar dos o más expresiones mediante estos operadores lógicos, también se generan dos posibles resultados:"Verdadero" o "Falso". Te aconsejo que hagas uso de los paréntesis para agrupar cada relación, facilitando así la lectura y evitar errores. Observa los siguientes ejemplos:

```
(Letter == 'X') && (A > 10)                          //False
(A == 10+3) && (B >= 12345) && (Letter != 'Q')       //True
(B > 12300) || (PI = 3.1412)                         //True
(A == B) || (A > 10 + 4)                             //False
!(A == B)                                            //True
(digitalRead(4) ==1) && (digitalRead(8)==1)          //True if pins, 4 and 8, are on level 1
```

## 3. OPERADORES COMPUESTOS

En muchas ocasiones vas a realizar operaciones muy sencillas con una variable y cuyo resultado va a parar a esa misma variable. Acuérdate de que dispones de los llamados "operadores compuestos" que simplifican esas expresiones. Los tienes resumidos en la siguiente tabla:

| OPERADOR | OPERACIÓN | EJEMPLO | EQUIVALE |
|---|---|---|---|
| ++ | Incrementa una unidad | X++ | X= X + 1 |
| - - | Decrementa una unidad | Y- - | Y = Y - 1 |
| + = | Suma | X+=Y | X = X + Y |
| -= | Resta | X-= 3 | X = X - 3 |
| *= | Producto | X *= Y | X = X * Y |

| | | | |
|---|---|---|---|
| /= | División | X /= 5 | X = X / 5 |

## 4. FUNCIÓN IF(...)

Es la función de control más importante y elemental.Permite evaluar y tomar decisiones en base aexpresiones. En informática, en los esquemasconocidos como “diagramas de flujo”, se emplea elsímbolo de la figura para representar una toma dedecisión. El controlador evalúa una expresión tancompleja o más que las que has visto antes. Si elresultado es “Verdadero” se ejecutan todas las funciones que haya encerradas entre las llaves “{ … }”. Si el resultado es “Falso” no seejecutan y el programa sigue su curso.

Recuerda. Cualquier expresión puede estar formada por operacionesaritméticas entre constantes y/o variables relacionadas entre sí por operadores decomparación, que a su vez se pueden relacionar mediante los operadores lógicos obooleanos. Tómate tu tiempo para meditar sobre esto último y repasa los ejemplosanteriores. Es importante y … ¡¡no tienes prisa!!

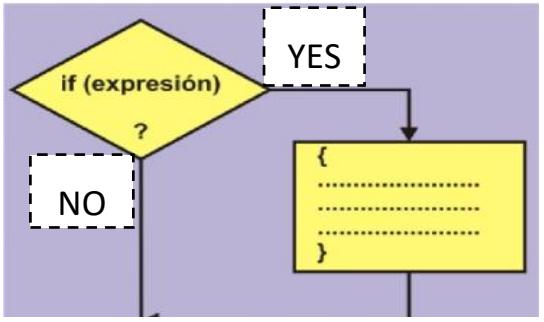



**Figura 1**

**Sintaxis:**

*if(expresión)*

*{*

*….*

*….*

*}*

*Expresión*: Establece la condición que el controlador Arduino debe evaluar. Puede serla comparación entre datos constantes y/o variables. Puede haber una o más comparaciones relacionadas entre sí mediante los operadores lógicos. También se pueden comparar los resultados obtenidos al ejecutar operaciones aritméticas u otras funciones del propio lenguaje Arduino.

*Llaves*: Como si fueran un "bocadillo", las llaves encierran las funciones que el controlador debe ejecutar en caso de que se cumpla la condición ("Verdadero"). No hacen falta ponerlas si únicamente se debe ejecutar una única función.

**Ejemplo:**

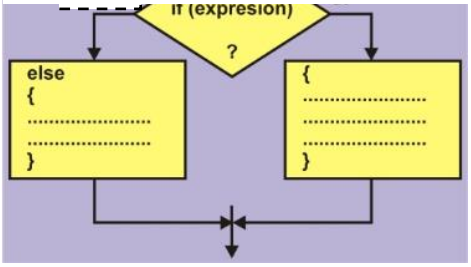
```
 void loop()
{
if((A>B) || (C < 25))                    //If the condition is true…
{
      digitalWrite(6,HIGH);              //Switches on pin 6
C=25;                                    //A value of 25 is stored in the C variable
}
....                                     //Continues the execution....
}
```

**Consejo**: Cuando emplees funciones de control en tus programas, es una buena costumbre alinear las llaves de forma que se vea claramente el emparejamiento de la que abre, con la llave que cierra. En el ejemplo se distinguen claramente las llaves que abren y cierran la función loop(), y las que se ejecutan si se cumple la condición if().

## 5. FUNCIÓN IF(…) ELSE

Derivada de la función if(…) anterior, aquí tienes otrafunción de control. Con ella puedes determinarclaramente qué hacer si no se cumple la condición(else). Mira la figura. Se evalúa la expresión ocondición. Si se cumple, “Verdadero”, se ejecutantodas las funciones que se encuentren entre las llaves“{ … }”. Igual que se hacía con la función if(…).

Figure 2




Si la condición NO se cumple, “Falso”, se ejecutan todas las funciones que hayencerradas tras las llaves else {…}. Ejecutadas las funciones en cualquiera de los doscasos (Verdadero o Falso), el programa continua su curso

**Sintaxis**:

*if(expresión)*

*{*

*….*

*….*

*}*

*else*

*{*

*….*

….

*}*

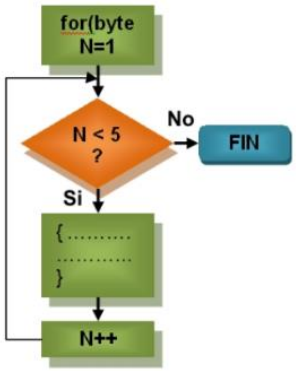
*Expresión*: Establece la condición que el controlador Arduino debe evaluar. Puede ser la comparación entre datos constantes y/o variables. Puede haber una o más comparaciones relacionadas entre sí mediante los operadores lógicos. También se pueden comparar los resultados obtenidos al ejecutar operaciones aritméticas u otras funciones del propio lenguaje Arduino.

*Llaves*: Contienen todas las funciones que el controlador debe ejecutar tanto encaso de que se cumpla la condición (if), como si no se cumple (else). No es necesario ponerlas si sólo se ejecuta una función para cada caso.

**Ejemplo**

```
if(digitalRead(4) == 1)          //If pin 4 is on "1" …
  {
digitalWrite(6,HIGH);                  //Enables pin 6
tone(2,2000,400);                      //Generates a 2 KHz tone on pin 2
}
else                             //…and if not…
digitalWrite(6,LOW);             //Disables output 6
….                               //Continues executing the program
…
```

## 6. FUNCIÓN FOR()




Se trata de una función que permite realizar bucles controlados. Un bucle es un conjunto de funciones cuya ejecución se repite un número determinado de veces.

Una imagen vale más que mil palabras. Mira el diagrama de flujo de la figura. Se declara la variable "N" con un valor inicial, por ejemplo, 1. Se evalúa y, si es menor que5, se ejecutan todas las funciones incluidas entre las llaves. El valor de la variable "N" se modifica automáticamente. En el ejemplo se incrementa en una unidad (N++). Vuelve a repetirse la evaluación de la condición y, si se sigue cumpliendo, se repite nuevamente la ejecución de las funciones.

El proceso se repite hasta que la condición sea falsa. En el ejemplo de la figura las funciones cerradas entre las llaves se ejecutan 4 veces: mientras "N" sea menor de 5.

**Sintaxis**:

*for(inicio,condición,modificador)*

{

….

….

}

***inicio***: Es una expresión que permite establecer el valor inicial de una variable. Solo se ejecuta una vez, al principio del bucle.

*condición*: Es la condición a evaluar. Si se cumple, "Verdadero", se ejecutan todas las funciones encerradas entre las llaves. Si no se cumple, "Falso", el bucle finaliza y la ejecución del programa sigue su curso. La condición se evalúa cada vez que se repite el bucle.

modificador: Es una expresión que permite modificar el valor de la variable con objeto de alcanzar la condición. Esta expresión se ejecuta cada vez que se repite el bucle.

*Llaves*: Encierran, a modo de "bocadillo", todas las funciones que forman el bucle y que deben ejecutarse un número determinado de veces.

**Ejemplo:**

```
for (int N = 1; N < 5; N=N+1)          //Establishes the loop conditions
{
digitalWrite(6,HIGH);                  //Enables pin 6
delay (150);                           //Pauses the program for 0.15"
digitalWrite(6,LOW);                   //Disenables pin 6
delay (1000);                          //Pauses the program for 1"
}
….                                     //Continues executing the program
```

La secuencia de encendido y apagado de la patilla 6, con sus correspondientes temporizaciones, se repite 4 veces.

Para establecer el inicio, la condición y el modificador de cualquier bucle for() puedes emplear todo tipo de expresiones como las que has estudiado al principio de esta Unidad: aritméticas, lógicas o booleanas y de comparación entre variables y/o constantes. Suponiendo…

```
int A = 5;
for(byte N=A+3; N <= A * 2 + 8; N = N + 3)
{
….
```

….

}

**Ahora tú**:

¿Cuál es el valor inicial?_____

¿Y el valor final?_____

¿En cuánto se incrementa cada vez?_____

¿Cuántas veces se repite el bucle?_____

## 7. FUNCIÓN WHILE()

Es una variante de la función for(). También se emplea por tanto para realizar bucles en donde un conjunto de funciones se ejecutan un número determinado de veces. Bucles y más bucles…

**Sintaxis**:

*while(condición)*

*{*

….

….

*}*

*condición*: Es la expresión condicional. Mientras sea verdadera se ejecutan las funciones del bucle, las que están encerradas entra las llaves. Si la condición es falsa el bucle finaliza y la ejecución del programa sigue su curso.

**Ejemplo:**

```
int N = 6 while (N > 0)                    //While N is greater than 0
…
{
digitalWrite(6,HIGH);                      //Enables pin 6
delay (150);                               //Pauses for 0.15”
digitalWrite(6,LOW);                       //Disenables pin 6
delay (1000);                              //Pauses for 1”
n--;                                       //The next value for N ( N = N – 1)
}
….
….                                         //Continues executing the program
```

## A. OTRAS FORMAS DE WHILE()

Es muy frecuente emplear la función while() como un bucle contenido en sí mismo. No lleva llaves y el bucle consiste en ejecutar la propia función hasta que la condición sea falsa. Analiza los siguientes ejemplos y observa dónde se pone el ‘;’.

**while(digitalRead(4)==1);**

La función while() se ejecuta constantemente mientras la patilla 4 esté a nivel“1”. Dicho en otras palabras, el ejemplo espera a que la patilla 4 pase a valer nivel “0”para continuar con el curso de la ejecución.

**while(! digitaRead(4));**

Similar al anterior pero al contrario. while() se ejecuta constantemente mientras la patilla 4NO esté a nivel “1”, es decir esté a nivel “0”. Espera por tanto a que la patilla 4 pase a valer nivel “1” para continuar con el curso de la ejecución.

**while(digitalRead(4)==0);**

Equivale justamente al ejemplo anterior. Espera a que la patilla 4 pase a valer“1” para seguir con el curso del programa

**while(1);**

Bucle infinito. La condición es siempre verdadera (1), por lo que la función while(1) se ejecuta constantemente de forma indefinida. No se ejecutará ninguna función que haya después de esta.

## 8. FUNCIÓN SWITCH() / CASE

Esta función te va a permitir elegir entre varios “caminos” para ejecutar diferentes grupos de funciones. Se compara el valor de una variable con diferentes valores indicados en múltiples funciones “case”. Cuando coincide con una de ellas se ejecutan las funciones correspondientes.

La palabra clave “break” finaliza la función “switch()”. Normalmente la usarás al final de cada caso o “case”. Es algo así como:

**“En caso de que el valor sea X ejecuta estas funciones. En caso de que valga Y ejecuta estas otras. Si vale Z estas otras, etc…”**

**Sintaxis**:

*switch(variable)*

*{*

*case X:*

*….;*

*break;*

*case n:*

*….;*

*….;*

*break;*

*default:*

*….;*

*}*

*variable*: Es la variable cuyo valor se va a comparar con los indicados en las diferentes “case”..

*case*: Establecen todos los valores con los que se va a comparar el contenido dela variable. Cuando coincide con uno de ellos se ejecutan todas las funciones comprendidas entre ése “case” y la expresión “break”.

*default*: Es opcional. Si lo usas, en caso de que no haya ninguna coincidencia, se ejecuta todas sus funciones.

**Ejemplo:**

```
switch(A)                          //Variable to be compared
{
case 1:                            //If the value of A is 1...
digitalWrite(6,HIGH);              //Enables pin 6
tone(2,200,200);                   //Tone on pin 2
break;                             //Exit
case 3:                            // If the value of A is 3…
B=digitalRead(7);                  //Reads pin 7
break;                             //Exit
case 124:                          //If the value of A is 124…
B=12*4;                            //The value of B is 48
digitalWrite(11,HIGH)              //Enables pin 11
break;                             //Exit
default:                           //If nothing else matches…
digitalWrite(6.LOW);               //Disables pin 6
```

```
digitalWrite(11,LOW);          //Disables pin 11
}
```

## 9. OTRAS FUNCIONES DE CONTROL

Las funciones de control que te presentado son las que más importantes y las que más vas a utilizar. No obstante el lenguaje Arduino de programación dispone de alguna otra. Su empleo lo dejo a tu criterio. De momento no quiero que te agobies. Quizá, conforme vayas aprendiendo y mejorando tus técnicas de programación, veas su utilidad.

### A. FUNCIÓN DO...WHILE()

Es muy parecida a la función while(). La diferencia se encuentra en que la condición para repetir o no las funciones de un bucle, se establece al final del mismo, no al principio.

**Sintaxis**:

```
do
{
….
} while(condition)
```

### B. FUNCIÓN BREAK

Esta función es una forma de “romper” cualquier bucle for(), while() o do…while() aunque no se dé la condición que hayas establecido. También la has usado para salir o finalizar la función switch() / case.

**Sintaxis**:

*break;*

### C. FUNCIÓN RETURN

Finaliza y retorna de cualquier función creada por el usuario. Si lo deseas puede devolver un valor al programa que llamó e hizo uso a esa función. Ya aprenderás más adelante a crear tus propias funciones.

**Sintaxis**:

*return;*

*return value;*

*value*: this is the value the function returns when it goes back to the calling program. It’s optional and it may be a constant or form part of a variable.

### D. FUNCIÓN GOTO

Traslada, de forma incondicional, la ejecución del programa al lugar indicado.

**Sintaxis**:

*test:*

*….*

*Go to test:* //Salta incondicionalmente al punto del programa en el que se encuentra la etiqueta indicada (“test”)..

# PRÁCTICA

## 10. EJEMPLO 1: TIMBRE ELECTRÓNICO V2

Se trata de una versión mejorada del ejemplo que viste anteriormente. En aquella ocasión había que reiniciar todo el sistema cada vez que querías hacer sonar el timbre. Esta versión es más correcta y una buena ocasión para utilizar la función if(). El programa chequea si se activa la patilla 4 en la que te recuerdo, está conectado el pulsador. Cuando esto sucede se generan dos tonos consecutivos de400 y 300 Hz cada uno. Si no se activa la patilla 4 el programa no hace nada.

Fíjate en las flechas rojas de la figura. Ya te aconsejé sobre lo conveniente que resulta alinear las llaves, de forma que se vea claramente el emparejamiento de la llave que abre con la que cierra.

```
void setup()
{
  pinMode( 2 , OUTPUT);  //D2 Salida del altavoz
}

void loop()
{
 if(digitalRead(4))    //Si se activa la patilla 4...
 {
  tone(2,400);         //Tono de 400 Hz
  delay(300);          //Durante 0,3"
  tone(2,300,300);     //Tono de 300 Hz durante 0.3"
 }
}
```

**Figura 3**

## 11. EJEMPLO 2: JUEGO DE LUCES V2

Otro viejo conocido, el juego de luces. En esta versión el desplazamiento se produce de derecha a izquierda o viceversa en función de que la patilla 4 esté a nivel “0” o a nivel “1”. Es una muy buena ocasión para el uso de la funciónif(..) else..

.

## 12. EJEMPLO 3: SEMÁFORO V3

Seguimos mejorando alguno de los ejemplos que hiciste anteriormente. Ahora le toca al semáforo. En esta versión la secuencia se inicia cuando el peatón acciona un pulsador conectado con la patilla 4. No hay que accionar el RESET para reiniciar el sistema como lo hacías en las versiones anteriores. Si no se acciona el pulsador el semáforo permanece en rojo. Otro buen ejemplo de cómo usar la función if(…)else.

Te sugiero que compares este ejemplo con sus versiones previas. Hay pequeñas diferencias, pero las hay, y poco a poco vamos mejorando nuestro semáforo.

## 13. EJEMPLO 4: BALIZA ELECTRÓNICA

Aquí tienes un buen ejemplo práctico que además te introduce en el uso de la función for(…).Trata de simular el funcionamiento de una baliza o faro electrónico que cada cierto tiempo produce un determinado número de destellos.

En la figura tienes elprograma principalincluido en lafunción loop(). Mediantefor(…) seestablece el valor inicial de la variable (N=1), las veces que ha de ejecutarse el bucle hasta que Nalcance el valor final (N<Destellos)y la modificación de N por cada vezque se ejecuta el bucle (N++).

```
int Tiempo = 150;          //Tiempo de encendido y apagado
int Destellos = 5;         //Nº de destellos deseados

void setup()
{
pinMode(6,OUTPUT);          //D2 Salida
}

void loop()
{
for(byte N=1;N<=Destellos;N++)
  {
  digitalWrite(6,HIGH);       //El led blanco se enciende
  delay(Tiempo);              //Temporiza
  digitalWrite(6,LOW);        //El led blanco se apaga
  delay(Tiempo);              //Temporiza
  }
```

**Figura 4**

Si lo miras con cuidado descubrirás que el bucle se repite 5 veces. En cada una de ellas el led blanco se enciende durante 0.15” y luego se apaga durante otros 0.15”.Finalizado el bucle se realiza una temporización de 1.5” y el proceso se vuelve a repetir.

- **Ahora tú**

Es un programa muy sencillo. Quizá te atrevas (espero que sí) a modificar el número de destellos, los tiempos de encendido y apagado o el tiempo entre una ráfaga de destellos y la siguiente. Prueba con diferentes valores.

## 14. EJEMPLO 5: SEMÁFORO V4

Como ya conoces el funcionamiento de la función for(…), es el momento ideal para diseñar la última versión del proyecto del semáforo, la V4. Funcionalmente no aporta nada nuevo respecto a la V3, sin embargo técnicamente es la mejor de todas las propuestas. Mira la solución.

Si te fijas bien en el programa verás porqué es la mejor solución. Efectivamente, la secuencia de los 6 tonos que se deben emitir cada vez que se encienden la luz verde y la luz ámbar, se realiza mediante sendos bucles for(…).Consecuencia de ello es un programa con menos funciones, que gasta menos memoria flash de programa y es por tanto más eficiente.

## 15. EJEMPLO 6: RÁFAGAS

Los bucles for(…) se pueden anidar. Esto es, dentro de un bucle for(…) puede haber otros bucles for(…), y dentro de cada uno de estos, puede haber otros y así sucesivamente.

Este programa es un sencillo ejemplo de anidamiento de dos bucles for(…) para producir destellos en dos luces cada

Por cada 5 destellos en el led blanco, se produce un destello en el led rojo. La secuencia finaliza al de 6 de estos destellos.

```
void loop()
{
  if(digitalRead(4))  //Si la entrada D4 es verdad (nivel "1") ...
  {
    for(byte R=1; R<=6;R++)     //Mientras R <= 6
    {
      for(byte B=1;B<=5;B++)    //Mientras B <= 5 ...
        {
        digitalWrite(6,HIGH);   //Activa el led blanco
        delay(80);              //Temporiza
        digitalWrite(6,LOW);    //Desactiva el led oblanc
        delay(100);             //Temporiza
        }                       //Siguiente valor para B
      digitalWrite(11,HIGH);    //Activa el led rojo
      delay(100);               //Temporiza
      digitalWrite(11,LOW);     //Desactiva el led rojo
      delay(120);               //Temporiza
```

**Figura 5**

Dentro de la función if(…), se encuentran dos bucles for(…) que se ejecutarán cada vez que se accione el pulsador conectado en la patilla 4. El más interno de los bucles se controla con la variable “B” que evoluciona desde 1 hasta 5. Produce una ráfaga de 5 destellos en el led blanco.

El bucle for(…) externo se controla con la variable “A” que evoluciona desde 1hasta 6. Como envuelve al bucle interno éste se repetirá 6 veces. Por cada una de ellas el led rojo produce un destello

**Cuando se anidan dos o más bucles for(…), se resuelve primero el bucle más interno siguiendo en orden hasta resolver, por último, el más externo.**

## 16. EJEMPLO 7: TIMBRE ELECTRÓNICO V3

Una última y definitiva versión de nuestro conocido timbre electrónico, la V3. Lo importante de este ejemplo es el tratamiento que hacemos con la señal de entrada en la patilla 4. No nos basta con accionar el pulsador e introducir un nivel “1”, no. También hay que soltarlo para volver a introducir el nivel “0” de reposo. Esta tarea se le conoce como “detección de un pulso”. Se emplea mucho en sistemas de control donde una señal de entrada no está formada por un nivel lógico estático como el “1” o el “0”. Hay muchos periféricos de entrada que proporcionan pulsos. Un pulso es una transición completa de nivel “0” a “1” y vuelta a “0” (0-1-0), o al contrario (1-0-1).

Para detectar estas situaciones (y otras muchas más), la función while() viene de maravilla. Observa el programa en la figura.

La primera función while() espera mientras la entrada digital número 4 esté a nivel “0”. La segunda mientras esté a nivel “1”.

Conclusión: hasta que no acciones el pulsador (nivel “1”) y luego lo sueltes (nivel “0”), no se emite sonido alguno. Pruébalo con cuidado y fíjate en ese detalle que lo diferencia respecto al timbre electrónico en su versión V2.

```
void setup()
{
}

void loop()
{
  while(! digitalRead(4));  //Espera mientras la patilla 4 está a "
  while(digitalRead(4));    //Espera mientras la patilla 4 está a "
  tone(2,400);              //Tono de 400 Hz
  delay(300);               //Durante 0,3"
  tone(2,300,300);          //Tono de 300 Hz durante 0.3"
}
```

**Figura 6**

## 17. EJEMPLO 8: CONTADOR

Una aplicación práctica. Imagina un sistema de acceso a un auditorio donde caben 10 personas.Un sensor, conectado en la patilla 4, detecta supaso y genera un “pulso” por cada una que entra. Cuando se llene, se emite una señal sonora como aviso de que empieza la función. Obsérvala función loop() de la figura. Las funciones contenidas en la función while() se ejecutan

```
void loop()
{
  byte Contador=1;          //Inicia la variable con 1
  while(Contador <= 10)
  {
    while(! digitalRead(4));  //Espera mientras la patilla 4 está
    while(digitalRead(4));    //Espera mientras la patilla 4 está
    Contador++;               //Pulso recibido, contador + 1
  }
  tone(2,400);              //Tono de 400 Hz
  delay(300);               //Durante 0,3"
  tone(2,300,300);          //Tono de 300 Hz durante 0.3"
}
```

**Figure 7**

mientras la variable “Contador” sea menor o igual que 10. Esta variable se va incrementando cada vez que se detecta un pulso 0-1-0 por la patilla 4. Para ello también empleamos sendas funcioneswhile().

Cuando se detectan los 10 pulsos, correspondientes al paso de otras tantas personas, se emite la señal sonora de aviso..

- **Ahora tú**

Una vez que grabes el programa comprueba su funcionamiento. Basta con que vayas aplicando impulsos por la patilla 4 Este simula el sensor detector de personas. ¿Funciona correctamente? ¿Sí? Pues yo no lo creo…

Si te fijas bien verás que parece que cuenta de menos. Es decir, que al pasar 5 o6 personas ya se dispara la señal sonora. Este fallo se debe a un problema conocido como “efecto rebote” típico de los pulsadores e interruptores. Aunque realices una única pulsación, la patilla de entrada del controlador no recibe un único pulso 0-1-0, recibe varios. Esto es debido a que los contactos metálicos del pulsador tardan un tiempo en estabilizarse cada vez lo accionas.

Aunque este tiempo suele cifrarse en unos pocos milisegundos, nuestro Arduino es muchísimo más rápido y detecta todos esos pulsos. Es decir, una única pulsación puede ser interpretada como uno, dos o más pulsos.

Una forma de evitarlos es intercalando una pequeña temporización. Cuando se detecta un cambio de estado en el pulsador temporizamos unos 20 mS antes de esperar el siguiente cambio. Se evitan así los rebotes que pudiera haber durante ese lapsus de tiempo.

```
while(! digitalRead(4));          //Waits as long as pin 4 is on “0”
delay(20);                        //Anti-rebound pause
while(digitalRead(4));            //Waits as long as pin 4 is on "1"
delay(20);                        //Anti-rebound pause
Counter ++;                       //Pulse received, counter + 1
```

## 18. EJEMPLO 9: TIEMPO

¿Te acuerdas de la función millis()? Era aquella que permitía conocer el tiempo que va transcurriendo desde que se conecta el Arduino y/o se reinicia. Pues bien, la vamos a usar junto con la función switch(…) / case para medir diferentes lapsus de tiempo.

Efectivamente, tras la secuencia de reinicio del Arduino vamos a medir el tiempo que trascurre. Cuando pase 1 segundo se enciende el led blanco. Cuando hayan pasado 2 segundos se encenderá el led verde, al de 3segundos se enciende el led ámbar y, al de 4 segundos, se enciende el led rojo. Por último, cuando hayan transcurrido 8 segundos desde que se inició el sistema, todos los leds quedarán apagados y se generará un tono. Observa y estudia la función loop() del programa de la figura.

```
void loop()
{
 A=millis();              //Lee el tiempo transcurrido tras el RESE
 switch(A)                //Analiza la variable
 {
   case 1000:               //Si vale 1000 (1") ...
     digitalWrite(6,HIGH); //Activa el led blanco
     break;
   case 2000:               //Si vale 2000 (2") ...
     digitalWrite(6,LOW); //Desactiva led blanco
     digitalWrite(9,HIGH); //Activa el led verde
     break;
   case 3000:               //Si vale 3000 (3") ...
     digitalWrite(9,LOW);  //Desactiva led verde
     digitalWrite(10,HIGH); //Activa el led ámbar
     break;
   case 4000:               //Si vale 4000 (4") ...
     digitalWrite(10,LOW); //Desactiva led ámbar
     digitalWrite(11,HIGH); //Activa el led rojo
     break;
   case 8000:               //Si vale 8000 (8") ...
     digitalWrite(11,LOW); //Desactiva led rojo
     tone(2,1000,300);      //Tono de 3 KHz durante 0.3"
     break;
 }
}
```

**Figura 8**

La función A=milis() carga la variable “A” con los milisegundos que van trascurriendo desde que se inicia el sistema. La función switch(A) hace un análisis de su contenido. En caso de que valga 1000 (case 1000:) se activa el led blanco. Ha pasado 1 segundo (1000 mS). En caso de que valga 2000 (case 2000:) se activa el verde, con 3000 el ámbar y con 4000 el rojo. Por último, en caso de que valga 8000(case 8000:) habrán transcurrido 8 segundos. Todos los leds quedan apagados y se emite un tono de 1 KHz.

El programa se sigue ejecutando, pero el tiempo transcurrido ya ha superado los 8000 ms (8 segundos), por lo que la variable “A” no cumplirá ninguna de las cinco condiciones.

¿Sabes cuánto tendrías que esperar para que se vuelvan a dar esas condiciones?

Pues vamos a calcularlo. La función millis() devuelve un entero largo sin signo(unsigned long)de 32 bits. Esto es, devuelve un valor de entre 0 y 4.294.967.295milésimas de segundo, lo que equivale a 4.294.967 segundos. Si tienes en cuenta que un día tiene 86.400 segundos (24 * 60 * 60), tendrán que transcurrir unos 50 días para que la función millis() se desborde y vuelva a empezar desde 0.

Si quieres volver a ver la secuencia de encendido de los leds puedes esperar todo ese tiempo o bien reinicias el sistema. Lo que prefieras…

# REFERENCIAS

LIBROS

[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

PÁGINAS WEB

[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