UNIDADE 3: EXPRESSÕES, PAUSAS E SONS

OBJETIVOS

O primeiro passo que vamos dar nesta unidade é observar atentamente as variáveis, constantes e expressões, em geral. Abordámo-las sucintamente na unidade anterior, nesta unidade vamos analisá-las em detalhe.

Vamos também apresentar algumas funções novas da programação em linguagem Arduino que vão permitir criar pausas e sons. Vão ter a possibilidade de as utilizar para criar programas consideravelmente mais versáteis e interessantes.

SECÇÃO TEÓRICA

- EXPRESSÕES
  - Constantes
  - Variáveis
  - Âmbito variável
  - Operações
- PAUSAS
  - A função delayMicroseconds()
  - A função delay()
  - A função micros()
  - A função millis()
- SOM
- FUNÇÕES DO SOM

SECÇÃO PRÁTICA

- EXEMPLO 1: Luzes intermitentes
- EXEMPLO 2: Conjuntos de luzes
- EXEMPLO 3: O sinal de trânsito V1
- EXEMPLO 4: A campainha elétrica
- EXEMPLO 5: Melodias
- EXEMPLO 6: O sinal de trânsito V2

MATERIAIS

-Computador de secretária ou portátil

-Ambiente de trabalho Arduino IDE; debe incluir material suplementar já instalado e configurado.

-Placa de controlo Arduino UNO

-Um cabo USB

ÍNDICE

SECÇÃO TEÓRICA

1. EXPRESSÕES

Já se deve ter apercebido que todas as funções da linguagem Arduino que estudou até ao momento necessitam de um ou mais parâmetros para funcionarem. Estes parâmetros são colocados entre os parênteses das funções e separados por vírgulas, como esta "," . Vamos relembrá-los:

- **pinMode(pin, mode)**: tem de especificar o número do pin a que se refere e se é *input* ou *output*.
- **digitalRead(pin)**: tem de especificar qual o número do pin que pretende ler.
- **digitalWrite(pin,value)**: tem de especificar qual o número do pin que quer utilizar para estabelecer o valor.

Estes parâmetros podem ser definidos de formas diferentes, que é exatamente o que fez em alguns dos exemplos da unidade anterior. O modo como indica estes parâmetros denomina-se "expressão". Existem três tipos básicos de expressões:

- **Constante**: Estabelecemos o valor do parâmetro na própria função. Por exemplo, **digitalRead(4)** lê o valor de input do pin número 4.
- **Variável**: Já definiu o valor de um parâmetro com uma variável. Por exemplo, **digitalRead(pushbutton**) lê o pin de input definido na variável "pushbutton". O controlador tem de localizar esta variável na memória RAM e extrair o número do pin a ser lido. Se não extrair, gera-se uma mensagem de erro.
- **Operação**: O valor do parâmetro é estabelecido como resultado de qualquer tipo de operação aritmética ou lógica entre as variáveis e as constantes. Por exemplo, **digitalRead((1+1)*2)** lê o pin 4, uma vez que este é o resultado de (1+1)*2.

A. CONSTANTES

O valor de um programa que não se altere pode ser utilizado como "constante". Caso tenha de ler um botão que está permanentemente ligado ao pin 4, pode utilizar a função digitalRead(4).

Cada vez que o controlador executa esta função, lê sempre o mesmo pin: o número 4. **Não** há forma de a alterar **durante** a execução do programa; se realmente pretender alterar esta situação, terá de modificar o programa e gravá-lo novamente na memória de controlo. Aqui está outro exemplo: imagine que pretende criar um programa para calcular uma circunferência, baseando-se num determinado diâmetro. A equação seria **l = d * π.**

l = comprimento da circunferência **d** = diâmetro da circunferência **π** = 3.141592

À luz deste exemplo, pode identificar o seguinte:

Qual é a constante?_____

E qual é a variável? _____

As constantes são uma parte integral do programa uma vez que aparecem nas instruções e funções. Ficam armazenadas no programa da memória FLASH do controlo e, por isso, não podem ser modificadas, a não ser que o programa seja gravado de novo. Não são voláteis e mantêm o seu valor, mesmo quando se desliga o sistema.

Mais exemplos:

digitalRead(12); *//lê o pin de input 12*

digitalWrite(6,HIGH); *// determina o pin de output 6 no nível "1"*

pinMode(9,OUTPUT); *//configura o pin 9 como output*

B. VARIÁVEIS

As variáveis são espaços na memória RAM do controlador especialmente desenhados para armazenar diferentes tipos de dados. Imagine-os como uma série de caixas ou contentores onde pode armazenar qualquer tipo de informação que poderá utilizar mais tarde.

Não se esqueça que pode ler e escrever a memória RAM as vezes que quiser, desde que o faça corretamente. Sim, pode realmente modificar o seu conteúdo durante a execução de um programa. É volátil e o seu conteúdo desaparece assim que se desligar o sistema.

Utilizou variáveis em alguns dos exemplos da unidade anterior. Tem de as declarar antes de as utilizar. Para as declarar, primeiro terá de lhes atribuir um nome e alguns conteúdos, utilizando a seguinte sintaxe:

type name = value

type: Estabelece o tipo de dados que a variável contém. Podem ser números, letras ou textos. Vamos abordá-los diretamente.

name: Este é o nome que atribui à variável ou recetáculo; terá de a utilizar quando pretender aceder aos conteúdos. Recorra a nomes curtos que associe facilmente aos conteúdos. Comece sempre com uma letra, não com um número e não deixe espaços em branco.

value: Este é o conteúdo ou valor que colocamos na variável. É opcional: pode criar uma variável que esteja completamente vazia. Caso faça isto, prepare a memória RAM do controlador para lhe atribuir um valor quando for apropriado para o seu programa.

A Tabela 1 inclui um resumo do tipo de variáveis mais comuns.

Tabela 1. Tipos de variáveis mais comuns

TIPO	BYTES	DESCRIÇÃO	EXEMPLOS
char	1	Um char ocupa 1 byte de memória e armazena um valor de caracter (8 bits). Os caracteres são literamente escritos com aspas simples (‘). O tipo de dados char é um tipo sinalizado, o que significa que codifica número desde -128 a 127.	char N = ‘1’ char letter =‘A’
byte	1	Um byte armazena um número não atribuído de 8-bit e rende uma média de 0 a 255 (28-1).	byte C = 23 byte result = C +18
int	2	Os números inteiros constituem o seu tipo de dados prioritário para o armazenamento de números. Armazenam um valor sinalizado de 16-bit (2-byte) e produzem uma média de -32768 a +32767.	int C = 2453 int value = C * 10 number = 2453
unsigned int or word	2	Armazena um número não atribuído de 16-bit e produz uma média de 35 (216-1).	unsigned int valor = 60000 word valor = 4328 * 10
long	4	Estas são variáveis atribuídas que armazenam 32 bits (4 bytes) e produzem uma média de -2,147,483,648 a 2,147,483,647.	long counter = -123456789
unsigned long	4	Variáveis não atribuídas que armazenam 32 bits (4 bytes) e produzem uma média de 0 a 4,294,967,295 (2^32 - 1).	unsigned long speedOfLight = 299792468
float	4	Os números de ponto flutuante podem ser tão elevados como 3.4028235E+38 e tão baixos como -3.4028235E+38. São armazenados como 32 bits (4 bytes) de informação.	float pi = 3.1416 float L = 4 * pi

A maior ou menor quantidade de memória RAM que se utiliza depende do tipo de variável que se defina. Aconselhamos a que defina as variáveis de acordo com o tipo de informação que armazenam. Por exemplo, se definir um tipo de variável “long”, irá utilizar 4 bytes de memória RAM, mas não é eficiente recorrer a este tipo se a informação que vai guardar estiver entre os valores 0 e 255. Provavelmente será melhor defini-la como tipo “byte” que consome apenas 1. A memória RAM é limitada; se a utilizar de forma errada, pode esgotar-se e ter assim uma surpresa desagradável.

Por outro lado, se declarar uma variável do tipo “byte” e lhe atribuir um valor maior do que 255 a variável fica sobrecarregada e perde a informação. É um pouco o que acontece com o milómetro de um carro quando excede a sua capacidade: recomeça do 0.

Os denominados **“Arrays”** (matrizes) são outro tipo de variável. Uma matriz é um grupo de variáveis e pode incluir qualquer uma das consideradas na tabela apresentada anteriormente (Tabela 1). Para aceder aos conteúdos das variáveis é necessário indicar o número indexado correspondente. Observe estes exemplos.

- **int Table[4]**

  Cria uma matriz vazia chamada “Table” e poupa espaço para armazenar quatro números de tipo inteiros. Ocupa 8 bytes na memória RAM do controlador (Figura 1).

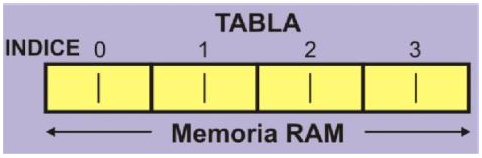

Figura 1

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

  Cria uma matriz “Table” com espaço suficiente para armazenar 4 tipos de números int.; é aqui que os três valores indicados se armazenam (Figura 2).

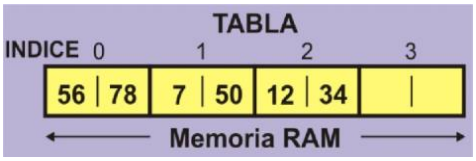

Figura 2

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

  Devolve os valores para a variável 0 (5678) e de variável 2 (1234) da matriz “Table”. Adiciona-os e guarda o resultado (6912) na variável “Sum.

- **unsigned int Values[12]**

  Cria a matriz “Values” que guarda espaço suficiente para 12 variáveis de tipo inteiro não atribuídas. Ocupam o total de 24 bytes na memória RAM.

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

  O resultado do cálculo (37035) é inserido na variável número 4 da matriz “Values”.

- **char Text[8] = “ARDUINO”**

  Cria um caracter matriz tipo “Text” que ocupa 8 bytes na memória RAM e armazena a cadeia “ARDUINO\o” (\o é um código de substituição que indica o fim da cadeia).

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

  Devolve a variável número 2 da matriz “Text” (Letra D) e guarda-a na variável “Letter” do tipo de dados char (Figura 3).

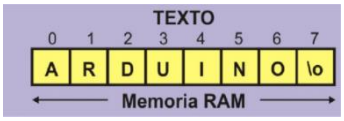

Figura 3

C. ÂMBITO VARIÁVEL

As variáveis podem ser “globais” ou “locais” dependendo de onde as declara no programa. Pode não interferir muito no momento, mas poderá ser importante mais tarde quando começar a criar programas maiores e mais complexos.

Globais: Estas variáveis são declaradas ou definidas à parte de todas as funções no programa, incluindo setup() e loop(). Alguns dos exemplos na unidade anterior utilizavam este tipo de variáveis. Observe a Figura 4. Vai verificar que as variáveis “Data” e “Result” são globais. Podem ser utilizadas por todas as funções no programa.

Locais: Estas variáveis são declaradas e criadas dentro de uma determinada função e só podem ser utilizadas por essa função. Desta forma evitamos que uma variável que seja declarada numa função seja utilizada ou modificada, ou ambos, noutra função.

Quando a execução de uma função termina, as variáveis locais desaparecem todas até se utilizar essa função novamente. De acordo com a Figura 4, “My_Function_1” contempla as seguintes variáveis locais: “Time”, “Hour” e “Result”.

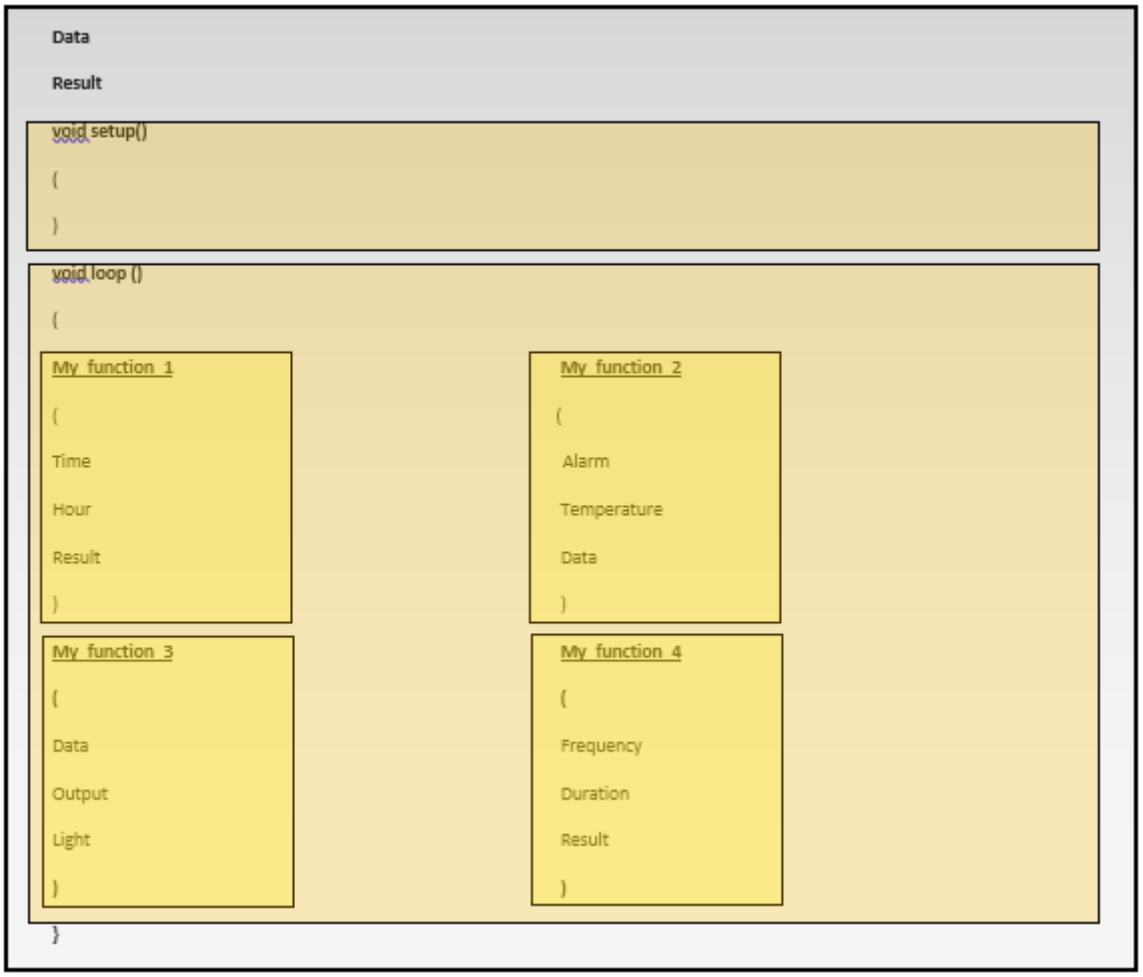

Figura 4

Responda às questões seguintes:

1. A “My_Function_2” pode usar a variável “Light”? _____

2. A “My_Function_1” pode usar a variável “Hour”? _____

3. A “My_Function_4” pode usar a variável “Data”? _____

D. OPERAÇÕES

De certeza que já se apercebeu que os parâmetros que se adequam às frases na linguagem Arduino, assim como os valores que atribuiu às variáveis podem ser obtidos como resultado de diversos tipos de operações aritméticas entre variáveis e/ou constantes.

Estas são as operações matemáticas mais comuns:

= **Equivalência**

+ **Adição**

- **Subtração**

***** **Multiplicação**

/ **Divisão**

% **Restante (o restante de uma divisão entre dois números inteiros)**

Efetuam-se os cálculos da esquerda para a direita. A multiplicação e a divisão têm prioridade sobre a adição e a subtração. Pode utilizar parênteses para estabelecer a prioridade que pretende numa operação. Observe os exemplos seguintes:

```
-> 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
```

Esta secção dar-lhe-á uma perspetiva muito mais flexível e capaz para expressar os parâmetros do que a utilizada por algumas funções Arduino. Observe os exemplos e responda às questões.

```
byte Light=3;

pinMode(Light×2,OUTPUT);

digitalRead(Light+1);

digitalWrite((Light-1)×3),HIGH);
```

Qual é o pin que está configurado como output? _____

Qual é o pin que é lido? _____

Qual é o pin que está registado com nível “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 = _____

2. PAUSAS

Pode parecer um pouco contraditório, mas é normalmente positivo que o controlador não faça absolutamente nada de útil, ou seja, que perca um pouco de tempo.

A linguagem de programação Arduino tem um número de funções que suspendem a execução de um programa por um período de tempo determinado. Pode não entender exatamente porquê neste momento, mas os exemplos vão clarificar este aspeto.

A. A FUNÇÃO DELAYMICROSECONDS

Suspende o programa pelo período de tempo especificado (em microssegundos). Há mil microssegundos num milissegundo e um milhão de microssegundos num segundo. Lembre-se do seguinte: 1 segundo = 1000 milissegundos (mS) e 1 milissegundo = 1000 milhões de segundo, dado que 1 segundo = 1,000,000 milhões de um segundo (µS) e 1 µS = 0.000001 segundos.

SIntaxe:

delayMicroseconds(n);

n: Indica o número de milissegundos que pretende suspender o programa (colocá-lo em pausa). É um integral de 16 bit não atribuído (unsigned int). Atualmente, o maior valor que irá produzir um atraso rigoroso é 16383 (aproximadamente 16 mS ou 0.016 segundos). Para atrasos mais longos do que alguns milhares de microssegundos, deve utilizar **delay()**; vamos analisar este aspeto já de seguida.

Exemplos:

A = 100;

delayMicroseconds(A); *//suspende o programa durante 100 µS = 0.1 mS = 0.0001 segundos*

delayMicroseconds(A×10-100); *//suspende o programa durante 900 µS = 0.9 mS = 0.0009 segundos*

B. A FUNÇÃO DELAY():

Suspende o programa (coloca-o em pausa) durante os milissegundos que indicar. Preste atenção ao seguinte: 1 segundo = 1000 milissegundos, ou por outras palavras, 1 milissegundo (mS) = 0.001 segundos.

Sintaxe:

delay(n);

n: indica o número de milissegundos (mS) que pretende colocar o programa em pausa. Referem-se como números extensos sem um número extenso de 32 bit atribuído ou não atribuído e o seu alcance de 0 a 4,294,967,295 (2^{32}- 1).

Exemplos:

A=100; delay(A); *//suspende o programa durante 100 mS = 0.1 segundo de atraso(1000) //suspende o programa durante 1000 mS = 1 segundo de atraso (A*600);*

//suspende o programa durante 60000 mS = 60 segundo = 1 minuto

Lembre-se que quando coloca o programa em pausa utilizando a função **delayMicroseconds()** ou a função **delay()**, o controlador só volta a executar as instruções do programa quando a pausa terminar.

C. A FUNÇÃO MICROS()

Devolve o número de microssegundos que passaram desde que o Arduino começou a correr o seu programa. Devolve um valor extenso não atribuído e o número irá ultrapassar o limite (retorna ao zero) depois de aproximadamente 70 minutos. Recorde que existem 1,000 µS (microssegundos) num milissegundo (mS) e 1,000,000 µS num segundo.

Sintaxe:

var = micros();

var: Esta é uma variável que armazena os microssegundos (µS) decorridos desde o reinício do sistema.

D. A FUNÇÃO MILLIS()

Devolve o número de milissegundos (mS) decorridos desde que o Arduino começou a correr o seu programa. Este número irá ultrapassar o limite (voltar ao zero), depois de aproximadamente 50 dias.

Devolve um valor extenso não atribuído e o número irá ultrapassar o limite (voltar ao zero), depois de aproximadamente 50 dias. Recorde que há 1,000 milissegundos (mS) num segundo.

Sintaxe:

var = millis();

var: Esta é a variável que armazena os milissegundos (mS) decorridos desde que o sistema foi reiniciado.

3. SOM

O que acontece quando se atira uma pedra num lago? A pedra atinge a água e altera a sua pressão, o que despoleta uma série de ondas que se espalham para o exterior e que vão desaparecendo gradualmente.

Com o som passa-se mais ou menos o mesmo. Precisamos apenas de um aparelho que emita vibrações e alterações de pressão no ar. Estas vibrações atingem os nossos tímpanos e o nosso cérebro entende-os como som. Quando conversamos ou cantamos, as nossas cordas vocais são responsáveis pela produção de alterações de pressão ou ondas que se espalham pelo ar até atingirem os tímpanos da/s pessoa/s com quem estamos a interagir.

Existem muitos aparelhos que provocam alterações na pressão do ar e que criam ondas que transportam o som: altifalantes, auriculares, intercomunicadores, sirenes, etc.

Todos estes aparelhos suportam-se nos mesmos componentes básicos: um diagrama, uma bobina e um íman permanente. O diafragma pode ser feito de papel, papelão, plástico ou qualquer outro material que tenha sido devidamente tratado. O diafragma é unido à bobina que, por sua vez, está montada sobre um íman permanente.

Um sinal elétrico é aplicado à bobina o que cria um campo magnético; este campo, por sua vez, interage com o campo magnético do íman permanente. Estes dois campos magnéticos podem atrair-se ou repelir-se uns aos outros. Essa atração ou repulsa entre a bobina e o íman “arrasta” ou “empurra” o diafragma que está unido ao íman. Os movimentos do diafragma provocam mudanças de pressão, vibrações ou ondas. E é assim que o som viaja pelo ar. Além disso, essas vibrações ou ondas são uma verdadeira reprodução do sinal elétrico que as havia gerado em primeiro lugar.

O sinal elétrico que está na origem desse processo deve ser variável e flutuar constantemente entre os níveis "1" e "0", a uma determinada velocidade ou frequência. Os sinais fixos não produzem o mesmo efeito. Imagine um sinal que esteja constantemente no nível "1" ou "0"; a bobina entra ou sai do íman permanente e é aí que fica; o diafragma não vibra e não são produzidas ondas de som.

Os seres humanos são capazes de ouvir frequências de som entre aproximadamente 20 Hz e 20 000 Hz (20 KHz), ou ciclos, por segundo, como se representa na Figura 5. Isto indica a frequência ou o número de vezes por segundo que um sinal passa de “1” para “0”.

Percebemos frequências como “arremessos”. Uma baixa frequência produz um som baixo; quanto maior a frequência, maior o som.

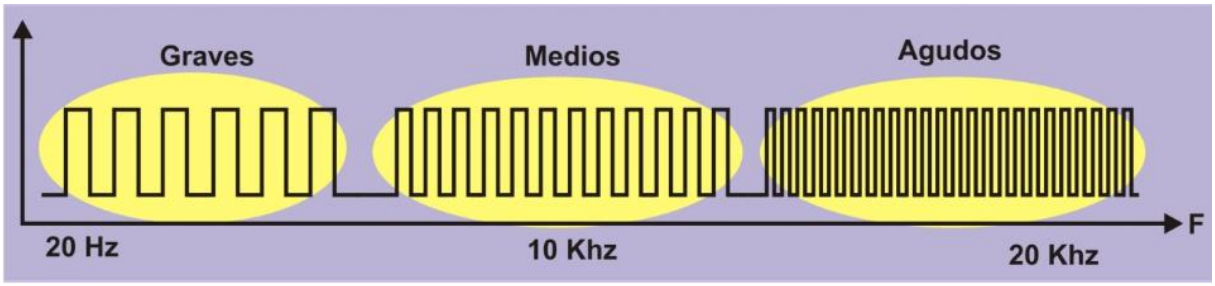

Figura 5

As vibrações com frequência inferior a 20 Hz denominam-se “*infra-sons*” e as de mais de 20 KHz chamam-se “*ultra-som*”. Os seres humanos não conseguem ouvir esses sons, no entanto, alguns animais conseguem.

Vamos fazer alguns cálculos. Suponha que quer gerar um sinal de som com uma frequência de 100 Hz:

1. Com base em **F**, a frequência desejada (100 Hz), calculamos o período **T** ou o tempo que um ciclo dura.

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

2. Agora calculamos o semiperíodo **S** ou, por outras palavras, o tempo em que o sinal permanece no nível "1" e no nível "0".

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

3. Com o que já aprendeu, agora poderia criar um programa que gerasse a frequência desejada através do altifalante conectado ao pin 2 na sua placa de BASIC I/O. Observe atentamente a seguinte solução:

```
void setup()
{
pinMode(2,OUTPUT);                    // Pin de output 2 (altifalante)
}
// Período T para o total de 10000 µS na frequência F de 100 HZ
void loop()
{
digitalWrite(2,HIGH);                 //Pin 2 no nível "1"
delayMicroseconds(5000);              //Suspende durante 5000 µS
digitalWrite(2,LOW);                  //Pin 2 no nível "0"
delayMicroseconds(5000);              //Suspende durante 5000 µS
}
```

Repare que o pin 2 permanece no nível "1" (HIGH) para 5000 µS e no nível "0" (LOW) durante o mesmo período. Por outras palavras, o ciclo "**T**" no nível "1" e no nível "0" é produzido no pin 2 e dura

10000 µS. Como é constantemente repetido, um segundo contém 100 ciclos (1000000/10000). Esta é a frequência **F**.

Registre o programa e assegure-se de que funciona corretamente. Se assim for, origina a emissão um som constante e irritante a uma frequência de 100 Hz.

Modifique o programa para obter sons com arremessos a partir de várias frequências **F** diferentes; calcule os seus períodos **T** e semi-períodos **S,** usando a tabela abaixo:

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		

4. FUNÇÕES DO SOM

A linguagem de programação Arduino permite-lhe que produza muito facilmente todos os tipos de sons. Não terá que resolver o período **P** ou o semi-período **S**. Tudo o que é necessário é indicar a frequência **F** e o Arduino trata do resto.

- **A função Tone()**

Gera um som com a frequência **F** necessária no pin de output e durante o período indicado.

Sintaxe:

tone(pin, frequency, duration);

pin: Indica em que pin irá gerar o som.

frequency: Este é um número inteiro não atribuído (unsigned int) que representa a frequência do som em Hz (hertz). Lembre-se que os seres humanos só conseguem ouvir frequências de som entre aproximadamente 20 Hz e 20 000 Hz (20 KHz) ou ciclos por segundo.

duration: Este parâmetro é opcional. É um número longo não atribuído (unsigned long number) que representa a duração do som em milissegundos (se não houver nenhuma indicação, o tom continua indefinidamente ou até a função noTone () ser executada).

Exemplos:

```
int Pin=11;

int F = 1200;

int D = 3000;
```

tone(2,F,D); *//Gera sons de 1200 Hz durante 3 segundos no pin 2*

tone(6,F*10,500); *// Gera sons de 12 KHZ durante 0.5 segundos no pin 2.*

tone(Pin,200); *//Gera um som indefinido de 200 Hz no pin 11*

- **A função noTone():**

Suspende um som num determinado pin.

Sintaxe:

noTone(pin);

pin: *representa em que pin o som será interrompido.*

Exemplo:

tone(2,13000); *//Gera um som indefinido de 13 KHz no pin 2.*

…. *//O som continua a soar.*

noTone(2); *//O som pára.*

SECÇÃO PRÁTICA

5. EXEMPLO 1: Luzes intermitentes

Este é o programa exemplo mais simples que pode imaginar: um LED branco intermitente, ligado ao pin 6 na placa de BASIC I/O. Observe a solução na Figura 6.

A variável “Tempo” contém a duração da pausa, 500 mS = 0,5 segundos. O Pin 6, ao qual o LED branco está ligado, está configurado como uma saída (output).

A parte principal do programa contida na função **loop ()** limita-se a ligar e desligar o pin 6 nos intervalos definidos pela variável “Tempo”, 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
}
```

Figura 6

- **Agora é a sua vez!**

Com este exemplo, pode optar por fazer um sem número de atividades diferentes. Aqui ficam algumas sugestões:

1. Pode tentar variar o intervalo de pausa usando a variável “Tempo”. Sugerimos que o diminua. Reduza-o pouco a pouco até chegar a 10 mS. De que é que se percebe?

   Parece que o LED branco está sempre ligado, mas isso não acontece. Na verdade, o LED liga-se e desliga-se muito rapidamente. De facto, a nossa retina não consegue detetar mudanças tão rápidas e cria uma ilusão de ótica: parece-nos que o LED está permanentemente aceso.

   O tempo é verdadeiramente um conceito elástico! O que parece constante para si, não o é da mesma forma para o controlador. Pode manter o LED aceso por 10 mS e mantê-lo desligado por outros 10 mS, mas não nos percebemos disso.

2. Em vez de usar a função delay(), tente usar outra: delaymicroseconds(). Poderá criar pausas extremamente curtas e precisas assim: apenas um milionésimo de segundo (µS).

3. Também pode modificar o programa para simular um farol. Por exemplo, pode fazer com que o LED emita um flash de luz de 0.1 segundo a cada 2 segundos.

6. EXEMPLO 2: Conjuntos de Luzes

Este é um bom exemplo. As quatro luzes LED no tabuleiro BASIC I/O vão ligar-se e desligar-se uma após a outra durante o mesmo período de tempo, começando com a branca. Dar-lhe-á uma sensação de movimento da direita para a esquerda.

O programa pode ser um pouco longo, mas é bastante simples. Acende a luz branca, faz uma pausa, desliga-a e liga o verde. Em seguida, faz uma pausa no verde, desliga-se e liga a âmbar. Em seguida, faz uma pausa na luz âmbar, desliga-a e liga o LED vermelho. Finalmente, faz uma pausa no vermelho, desliga-se e o ciclo inicia-se de novo.

- **Agora é a sua vez!**

1. Altere o tempo: a ideia é ir mais rápida ou mais lentamente à medida que se move da direita para a esquerda.

2. Primeiro, faça as luzes acenderem-se e desligarem-se da direita para a esquerda e, de seguida, da esquerda para a direita.

7. EXEMPLO 3: Sinal de Trânsito V1

E que tal estruturar um pequeno projeto com tudo o que aprendeu? Sugiro que simule um sinal de trânsito simples; esta será a primeira versão de uma série de sinais de trânsito que poderá ir melhorando à medida que vai evoluindo nos seus conhecimentos.

O programa em si não é inovador. Envolve apenas a ativação e desativação dos LEDs verde, âmbar e vermelho na ordem correta no tempo que indicar. No entanto, há já uma diferença que pode sinalizar neste momento.

- **Conclusão**

Repare que todas as funções do programa que estabelecem a sequência de trabalho do sinal de trânsito estão incluídas na função **setup()**; a função principal **loop()** está vazia.

Isto é perfeitamente normal. Exploramos este aspeto em lições anteriores, recorda-se? Deve incluir a função **loop()**, mesmo que não tenha nada nela. Por outro lado, todas as funções incluídas na função **setup()** são executadas apenas uma vez. Então, sempre que quiser que o seu sinal de trânsito execute uma nova sequência, terá que pressionar novamente o botão **RESET**; experimente.

8. EXEMPLO 4: Campainha elétrica

Aqui está outro projeto simples. Desta vez vai simular uma campainha elétrica. Sempre que pressionar o botão, a campainha emite um som composto de três campos diferentes.

Esta será a primeira vez que vai usar a função **tone()** e verá que é realmente muito simples. Tudo o que precisa de fazer é definir o pin de saída, a frequência e a duração (este último parâmetro é opcional).

A solução para o programa está patente na figura ao lado. Um tom inicial de 1000 Hz é gerado por 0.2 segundos. Em seguida, após uma pausa de 0.4 segundos, outro tom de 15000 Hz é gerado por 0.3 segundos. Por último, 0.4 segundos depois, um terceiro tom de 2000 Hz é gerado por 0.4 segundos.

Recebemos os tons no pin 2 do controlador.

Certifique-se de que todas as funções estão incluídas na função **setup()** para que a sequência seja executada sempre que pressionar **RESET**.

- **Agora é a sua vez!**

Vai melhorar a campainha, adicionando uma luz de sinal. Tal como na sequência descrita anteriormente, sugerimos que ligue o LED branco sempre que pressionar **RESET**. Assim que o terceiro tom é gerado e a sequência termina, a luz LED deve apagar-se.

9. EXEMPLO 5: Melodias

Aqui fica um exercício para quem gosta de música. Este exemplo combina uma série de notas musicais de diferentes frequências e durações que dão origem a uma melodia bem conhecida. Para simplificar, cada nota musical corresponde a uma certa frequência. Obtemos as notas nítidas e planas e as suas combinações a partir dessas frequências. Experimente e tente conciliar diferentes combinações!

10. EXEMPLO 6: Sinal de trânsito V2

Outro pequeno projeto para si. É uma versão melhorada do sinal de trânsito V1: adicionamos alguns sinais acústicos de aviso para os cegos.

O programa pode parecer um pouco longo, mas se observar com atenção, vai aperceber-se que não é assim muito complicado. É muito sequencial. Começa por acionar a luz verde ao mesmo tempo que gera seis notas de 1 KHz com 0.5 segundos de duração. De seguida, aciona a luz âmbar e gera seis 1 KHZ com notas de 0.4 segundos de duração. Por último, a luz vermelha acende e uma nota de 1KHz soa por 6 segundos.

Como todas as funções do programa estão incluídas na função **setup()**, essas funções são executadas apenas uma vez: cada vez que pressionar **RESET**, a sequência inicia-se.

- **Agora é a sua vez!**

Portanto, a execução do programa não dura muito tempo quando estamos a testá-lo. Vai aperceber-se que o tempo em que cada luz permanece acesa é realmente muito menor do que a sua durabilidade real.

Procure um sinal de trânsito na sua rua ou área de residência e registe durante quanto tempo cada luz permanece acesa. Também deve ouvir os sons de advertência que cada sinal faz e se as luzes e os sons são intermitentes ou não. A ideia é modificar o seu programa para que simule, o mais aproximadamente possível, a forma como os sinais de trânsito reais funcionam.

REFERÊNCIAS

LIVROS

[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

SÍTIOS 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