UNITÀ 3: ESPRESSIONI, PAUSE E SUONI

OBIETTIVI

La prima cosa che intendiamo fare è dare un'occhiata più da vicino a variabili, costanti e espressioni in generale.

Intendo anche presentarvi alcune nuove funzioni del linguaggio di programmazione Arduino; permetteranno di creare pause e suoni. Potrai utilizzarle per creare programmi molto più versatili e interessanti.

SESSIONE TEORICA

- ESPRESSIONI
  - Costanti
  - Variabili
  - Ambito variabile
  - Operazioni
- PAUSE
  - La funzione delayMicroseconds()
  - La funzione delay()
  - La funzione micros()
  - La funzione millis()
- SUONI
- FUNZIONI AUDIO

SESSIONE PRATICA

- ESEMPIO 1: Luci lampeggianti
- ESEMPIO 2: Set di luci
- ESEMPIO 3: Il segnale di traffico V1
- ESEMPIO 4: La campana elettrica
- ESEMPIO 5: Melodie
- ESEMPIO 6: Il segnale di traffico V2

MATERIALI PRATICI

- Portatile o Pc desktop

-Ambiente di lavoro Arduino IDE;questo dovrebbe includere il materiale supplementare già installato e coonfigurato;

- Arduino UNO controller board;

-Cavo USB.

SOMMARIO

SEZIONE TEORICA

1. ESPRESSIONI

A questo punto, probabilmente avete capito che tutte le funzioni di lingua Arduino studiate finora hanno bisogno di uno o più parametri per funzionare. Sono contenute all'interno delle parentesi delle funzioni e ci sono virgole "," tra di loro. Procediamo:

- **pinMode(pin, mode):** è necessario specificare il numero del pin cui ci si sta riferendo e se è un input o un output.
- **digitalRead(pin):** è necessario specificare il numero del pin che si desidera leggere.
- **digitalWrite(pin, value):** è necessario specificare quale numero di pin si desidera utilizzare per stabilire il valore.

È possibile esprimere questi parametri in modi diversi. Il modo in cui indichi questi parametri è chiamato "espressione". Ne esistono tre tipi di base:

2. **Costante:** stabiliremo il valore del parametro nella funzione stessa. Ad esempio, **digitalRead(4)** legge il valore del pin di input n° 4.
3. **Variabile:** Hai già definito il valore del parametro con una variabile. Ad esempio, **digitalRead(pushbutton)** legge il pin di input definito nella variabile "pushbutton". Il controller deve individuare questa variabile nella sua memoria RAM ed estrarre il numero del pin da leggere. In caso contrario, viene generato un messaggio di errore.
4. **Funzionamento**: Il valore del parametro viene stabilito come risultato di qualsiasi tipo di operazione aritmetica o logica tra le variabili e le costanti. Per esempio, **digitalRead((1 + 1) * 2)** legge il pin 4 , come risultato di (1 + 1) * 2.

A. COSTANTI

Un valore di programma che non cambia mai può essere utilizzato come "costante". Se si deve leggere un pulsante collegato in modo permanente al pin 4 è possibile utilizzare la funzione **digitalRead(4)**.

Ogni volta che il controllore esegue questa funzione, legge sempre lo stesso pin: numero 4. **Non** c'è modo di cambiarlo **durante** l'esecuzione del programma; se decidi di modificarla devi modificare il programma e registrarla nuovamente nella memoria del controller. Ecco un altro esempio: immaginate di voler creare un programma per calcolare la circonferenza basata su un certo diametro. L'equazione sarebbe **l = d * π**.

l = lunghezza della circonferenza.
d = diametro della circonferenza
π = 3.141592

Alla luce di questo esempio si può dedurre quanto segue:

Quale è la costante? _____

E quale è la variabile? _____

Le costanti sono parte integrante del programma come appaiono nelle istruzioni o nelle funzioni. Sono memorizzate nel programma di memoria FLASH del controller e quindi non possono essere modificate a meno che il programma non venga rieseguito. Non sono volatili e anche quando è scollegato mantengono il loro valore.

Esempi:

digitalRead(12); legge il pin di input 12
digitalWrite(6,HIGH); posiziona il pin di 6 sul livello "1"
pinMode(9,OUTPUT); Configura il pin 9 come output

B. VARIABILI

Le variabili sono spazi nella memoria RAM del controller appositamente progettati per memorizzare diversi tipi di dati. Pensate a loro come una serie di scatole o contenitori in cui è possibile memorizzare qualsiasi tipo di informazione e utilizzarlo più tardi quando necessario.

Non dimenticate che è possibile leggere e scrivere la memoria RAM quante volte volete. È possibile modificare il suo contenuto durante l'esecuzione di un programma. È volatile e il contenuto scompare quando l'alimentazione è scollegata.

Le variabili sono state utilizzate in alcuni esempi nell'unità precedente. Devi dichiararle prima di usarle. Per dichiararle, prima devi dare loro un nome e alcuni contenuti utilizzando la seguente sintassi:

Tipo nome = valore

tipo: Definisce il tipo di dati che contiene la variabile. Possono essere numeri, lettere o testi. Li guarderemo direttamente.

nome: questo è il nome assegnato alla variabile o al contenitore; è necessario utilizzarlo in futuro quando si desidera accedere ai contenuti. Provate ad usare nomi concisi che ricordino in qualche modo i contenuti. Iniziate sempre con una lettera, non un numero e non lasciate spazi vuoti.

valore: Questo è il contenuto o il valore che abbiamo inserito nella variabile. È opzionale: è possibile creare una variabile completamente vuota. Se si esegue questa operazione, provvedete ad assegnare nella memoria RAM del controller un valore quando è appropriato per il programma.

La seguente tabella contiene un riepilogo dei tipi più comuni di variabili:

TIPO	BYTES	DSCRIZIONE	ESEMPI
char	1	Un char richiede 1 byte di memoria e memorizza un carattere (8 bit). I caratteri letterali sono scritti tra virgolette singole ('). Il tipo di dati char è con segno, il che significa che codifica i numeri da -128 a 127.	char N = '1' char letter ='A'
byte	1	Un byte memorizza un numero senza segno a 8 bit e produce una gamma da 0 a 255 (28-1).	byte C = 23 byte result = C +18
int	2	Gli interi sono il tipo di dati primario per l'archiviazione dei numeri. Memorizzano un valore con segno a 16 bit (2 byte) e producono una gamma da -32768 a +32767	int C = 2453 int value = C * 10 number = 2453
unsigned int or word	2	Memorizza un numero senza segno a 16 bit e produce una gamma da 0 a 65535 (216-1)	unsigned int valor = 60000 word valor = 4328 * 10
long	4	Queste sono variabili con segno che memorizzano 32 bit (4 byte) e memorizzano una gamma da -2,147,483,648 a 2,147,483,647.	long counter = -123456789
unsigned long	4	Variabile senza segno che memorizza 32 bit (4 byte) e produce una gamma da 0 a 4,294,967,295 (2 ^ 32 - 1).	unsigned long speedOfLight = 299792468
float	4	I numeri a virgola mobile possono essere pari a 3.4028235E + 38 e fino a -	float pi = 3.1416 float L = 4 * pi

		3.4028235E + 38. Sono memorizzati come 32 bit (4 byte) di informazioni.	

È abbastanza chiaro che consumi una quantità maggiore o minore di memoria RAM in base al tipo di variabile definita. Ti consiglio di definire le variabili in base al tipo di informazioni che memorizzano. Ad esempio, se si definisce una variabile tipo "lunga" si consumano 4 byte di memoria RAM, ma non è particolarmente efficiente utilizzare una di queste se le informazioni da memorizzare sono solo tra 0 e 255. Probabilmente è meglio definirlo come un tipo "byte" che consuma solo 1 byte memoria RAM. La memoria RAM è limitata.

D'altra parte, se si dichiara una variabile di tipo "byte" e si dà un valore superiore a 255, la variabile si sovraccarica e perde le informazioni.

Le cosiddette " **Arrays** " sono un altro tipo di variabile. Un array è un gruppo di variabili e può comprendere una di quelle della tabella precedente. Per accedere al contenuto delle variabili è necessario indicare il proprio numero di indice. Date un'occhiata a questi esempi.

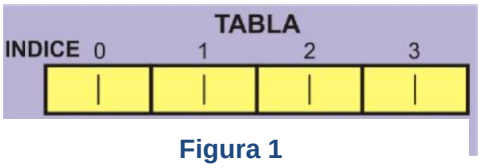


Figura 1

- **int Table[4]**

Crea un array vuoto chiamato "Tabella" e salva spazio per memorizzare quattro numeri. Richiede quindi 8 byte nella memoria RAM del controller.

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

  Crea un array "Table" con abbastanza spazio per memorizzare 4 numeri; qui vengono memorizzati i tre valori indicati.

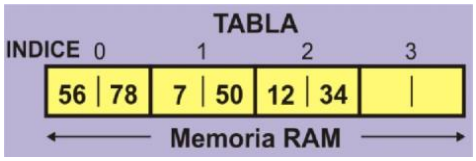


Figura 2

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

Recupera i valori per la variabile 0 (5678) e la variabile 2 (1234) dall'array "Table". Esso li aggiunge e memorizza il risultato (6912) nella variabile "Sum".

- **unsigned int Values[12]**

Crea un array "Valori" che consente di risparmiare spazio sufficiente per 12 intere variabili senza segno. Richiede un totale di 24 byte nella memoria RAM.

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

Il risultato del calcolo (37035) viene inserito nella variabile numero 4 dell'array "Valori".

- **char Text[8] = "ARDUINO"**

Crea un array carattere di tipo "Text" che richiede 8 byte nella memoria RAM e memorizza la catena "ARDUINO \o" (\o è un codice di override che indica la fine della catena).

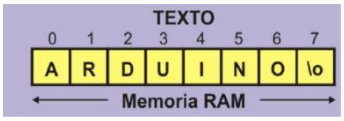


Figura 3

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

Recupera la variabile numero 2 dall'array "Testo" (lettera D) e lo memorizza nella variabile "letter".

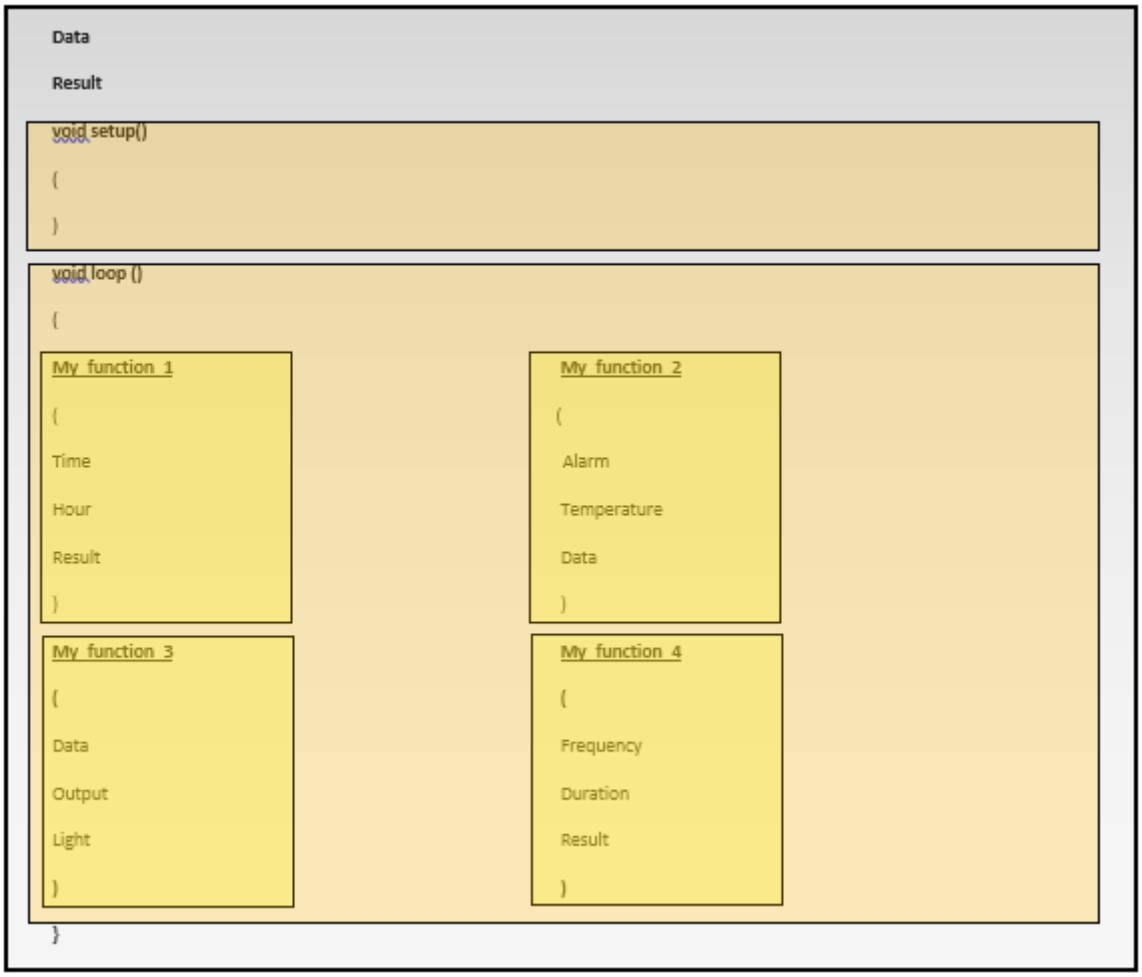
C. AMBITO DELLE VARIABILI

Le variabili possono essere "globali" o "locali" a seconda del luogo in cui si dichiarano nel programma. Ciò non influisce molto al momento, ma può essere importante in seguito quando si iniziano a creare programmi più grandi e complessi.

Globali: Queste sono le variabili che vengono dichiarate o definite al di fuori di tutte le funzioni del programma, tra cui setup() e loop(). Alcuni degli esempi dell'unità precedente utilizzavano questi tipi di variabili. Date un'occhiata alla figura e vedraete che le variabili "Dati" e "Risultato" sono globali. Possono essere utilizzate da tutte le funzioni del programma.

Locali: Queste sono le variabili che vengono dichiarate e create all'interno di una determinata funzione e possono essere utilizzate solo da tale funzione. In questo modo evitiamo una variabile dichiarata in una funzione che venga utilizzata o modificata da un'altra.

Quando l'esecuzione di una funzione finisce tutte le variabili locali scompaiono fino a quando non si utilizza nuovamente la funzione. Secondo la figura, "My_Function_1" ha le seguenti variabili locali: "Ora", "Ora" e "Risultato".



Rispondi alle seguenti domande:

1. Può "My_Function_2" utilizzare la variabile "Light"? _____

2. Può "My_Function_1" utilizzare la variabile "ora"? _____

D OPERAZIONI

Sono sicuro che hai già indovinato che i parametri che si combinano con il linguaggio Arduino e i valori caricati nelle variabili possono essere ottenuti come risultato di ogni tipo di operazioni aritmetiche tra variabili e / o costanti.

Queste sono le operazioni matematiche più comuni:

= Equivalenza
+ Addizione
- Sottrazione
*** Moltiplicazione**
/ Divisione
% Resto (il resto di una divisione tra due numeri interi)

Tutti i calcoli vengono eseguiti da sinistra a destra, proprio come in aritmetica. La moltiplicazione e la divisione hanno la priorità rispetto all'aggiunta e alla sottrazione. È possibile utilizzare parentesi per stabilire la priorità desiderata in un'operazione. Esaminiamo attentamente i seguenti esempi:

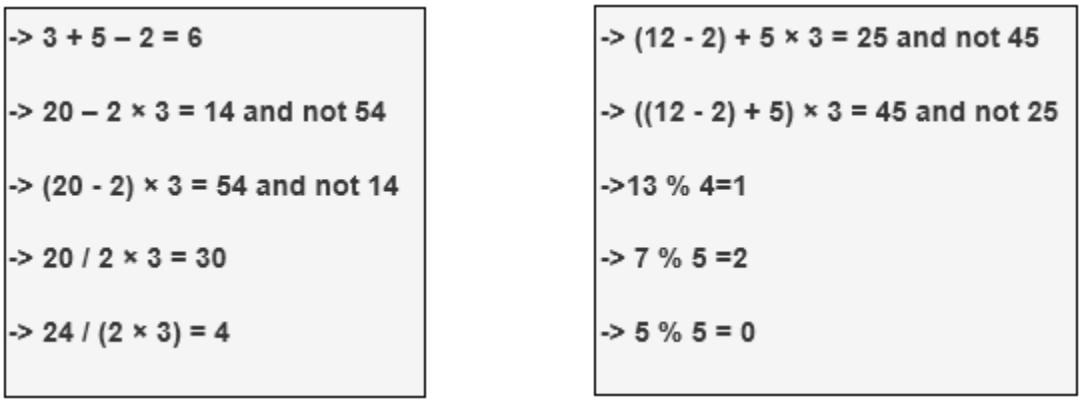


Figura 4

Questa sezione vi darà molta flessibilità e capacità quando esprimi i parametri che utilizzano alcune funzioni di Arduino. Date un'occhiata agli esempi e provate a rispondere alle domande.

```
byte Light=3;

pinMode(Light×2,OUTPUT);

digitalRead(Light+1);

digitalWrite((Light-1)×3),HIGH);
```

Quale pin è configurato come output? _____

Quale pin viene letto? _____

Quale pin è impostato a livello "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 PAUSE

Questo sembra essere una contraddizione, ma è spesso una buona idea per il controllo non fare niente, oppure detto in altre parole, non perdere tempo.

Il linguaggio di programmazione Arduino ha una serie di funzioni che mettono in pausa l'esecuzione di un programma per un determinato lasso di tempo.

A LA FUNZIONE DELAYMICROSECONDS

Sospende il programma per la quantità di tempo (in microsecondi) specificata. Ci sono mille microsecondi in un millisecondo e un milione di microsecondi in un secondo. Ricorda quanto segue: 1 secondo = 1000 millisecondi (mS) e 1 millisecondo = 1000 milioni di secondi dato che 1 secondo = 1.000.000 milioni di secondi di secondo (µS) e 1 µS = 0.000001 secondi.

Sintassi:

delayMicroseconds(n);

n: indica il numero di millisecondi per cui si desidera mettere in pausa il programma. È un intero a 16 bit (intero senza segno). Attualmente, il valore più grande che produrrà un ritardo accurato è 16383 (circa 16 mS o 0.016 secondi). Per i ritardi più lunghi di poche migliaia di microsecondi, devi usare **delay()**; lo osserveremo a breve.

Esempi:

A = 100;

delayMicroseconds(A); // mette in pausa il programma per 100 µS = 0,1 mS = 0,0001 secondi

delayMicroseconds(A×10-100); // mette in pausa il programma per 900 µS = 0,9 mS = 0,0009 secondi

B LA FUNZIONE THE DELAY():

Sospende il programma tanti millisecondi quanti indicati. Tenete presente quanto segue: 1 secondo = 1000 millisecondi, ovvero 1 millisecondo (mS) = 0.001 secondi.

Sintassi:

delay(n);

n: indica il numero di millisecondi (mS) per cui si desidera mettere in pausa il programma. Questi sono indicati come numeri lunghi senza un segno a 32 bit o numeri lunghi non firmati e la loro gamma è da 0 a 4,294,967,295 (2 ^ 32-1).

Esempi:

A=100; delay(A); // Sospende il programma per 100 mS = 0,1 secondi di ritardo (1000)

// Sospende il programma per 1000 mS = 1 ritardo segundo (A * 600);

// Sospende il programma per 60000 mS = 60 secondi = 1 minuto

RRicorda che quando si mette in pausa il programma utilizzando la funzione **delayMicroseconds()** o la funzione **delay()**, il controller non esegue più le istruzioni di programma finché la pausa non è terminata.

C LA FUNZIONE MICROS()

Restituisce il numero di microsecondi trascorsi da quando Arduino ha iniziato a eseguire il programma. Restituisce un valore lungo senza segno e il numero si azzererà (tornare a zero) dopo circa 70 minuti. Ricordati che ci sono 1.000 µS (microsecondi) in un millisecondo (mS) e 1.000.000 µS in un secondo.

Sintassi:

var = micros();

var: Questa è la variabile che memorizza i microsecondi (µS) trascorsi dal riavvio del sistema.

D. LA FUNZIONE MILLIS()

Restituisce il numero di millisecondi (mS) trascorso da quando Arduino ha iniziato a eseguire il programma. Questo numero si azzererà (tornerà a zero) dopo circa 50 giorni.

Restituisce un valore lungo senza segno e il numero si azzarerà (tornerà a zero) dopo circa 50 giorni. Ricorda che ci sono 1.000 millisecondi (mS) in un secondo.

Sintassi:

var = millis();

var: Questa è la variabile che memorizza i millisecondi (mS) trascorsi dal riavvio del sistema.

3 SUONO

Cosa succede quando lanci una pietra in uno stagno? La pietra colpisce l'acqua e cambia la sua pressione; questo emette una serie di onde che si propagano verso l'esterno e poi scompaiono gradualmente.

Il suono è molto simile. Tutto quello che serve è un dispositivo in grado di vibrare e produrre variazioni di pressione o vibrazioni nell'aria. Queste vibrazioni raggiungono i nostri timpani e il nostro cervello li percepisce come un suono. Quando parliamo o cantiamo, le nostre corde vocali producono cambiamenti di pressione che si estendono attraverso l'aria fino a raggiungere i timpani altrui.

Ci sono molti dispositivi noti che provocano cambiamenti nella pressione dell'aria e creano onde che trasportano suoni: altoparlanti, cuffie, buzzer, sirene ecc.

Tutti questi dispositivi si basano sugli stessi tre componenti fondamentali: un diaframma, una bobina e un magnete permanente. Il diaframma può essere fatto di carta, cartone, plastica o qualsiasi altro materiale che sia stato trattato correttamente. Il diaframma è collegato alla bobina che a sua volta è montata su un magnete permanente.

Viene applicato un segnale elettrico alla bobina che crea un campo magnetico; questo a sua volta interagisce con il campo magnetico del magnete permanente. Questi due campi magnetici possono attirarsi o respingersi a vicenda. Questa attrazione o repulsione tra la bobina e il magnete "trascina" o "spinge" il diaframma unito al magnete. I movimenti del diaframma provocano cambiamenti di pressione, vibrazioni o onde. Ed è così che il suono attraversa l'aria. Oltre a ciò, queste vibrazioni o onde sono una vera e propria riproduzione del segnale elettrico che li ha generati in primo luogo.

Il segnale elettrico che causa questo processo deve essere variabile e fluttuare costantemente tra il livello "1" e "0" ad una determinata velocità o frequenza. I segnali fissi non producono lo stesso effetto. Immagina un segnale che è costantemente sul livello "1" o "0"; la bobina entra o esce dal magnete permanente e rimane immobile; il diaframma non vibra e non si producono onde sonore.

Gli esseri umani sono in grado di percepire frequenze sonore tra circa 20 Hz e 20.000 Hz (20 KHz) al secondo. Ciò indica la frequenza o il numero di volte al secondo che un segnale passa da "1" a "0".

Percepiscono frequenze come "altezze". Una bassa frequenza produce un suono basso; maggiore è la frequenza più alto è il suono.

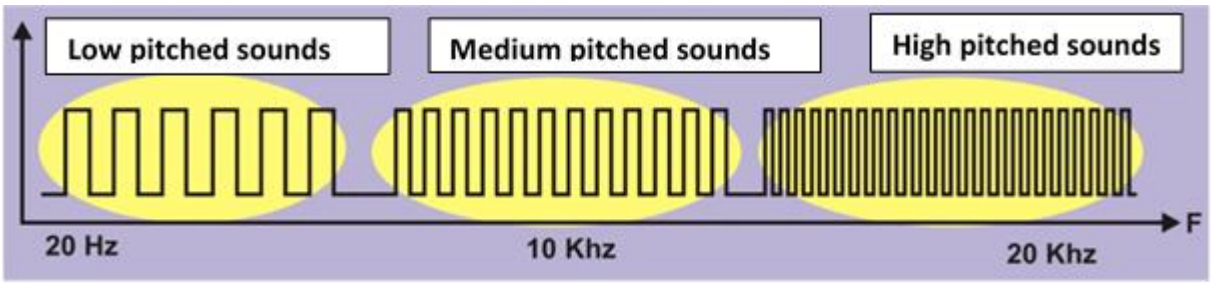


Figura 5

Le vibrazioni con frequenza inferiore a 20 Hz sono chiamate "infrasuoni" e quelli con frequenza a più di 20 KHz "ultrasuoni". Gli esseri umani non sono in grado di ascoltare questi suoni.

Facciamo dei calcoli. Supponiamo che si desideri generare un segnale acustico con una frequenza di 100 Hz:

1. Sulla base di **F**, la frequenza desiderata (100 Hz) calcoliamo il periodo **T**, o il tempo di un ciclo

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

2. Ora calcoliamo il semi periodo S o, in altre parole, il tempo in cui il segnale sarà a livello "1" e al livello "0".

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

3. Con quello che hai appreso potresti creare un programma che generi la frequenza desiderata tramite l'altoparlante collegato al pin 2 sul tuo board I / O BASIC. Diamo un'occhiata alla seguente soluzione:

```
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);              //Mette in pausa per  5000 µS
digitalWrite(2,LOW);                  //Pin 2 at level "0"
delayMicroseconds(5000);              //Mette in pausa per 5000 µS
}
```

Si noti che il pin 2 rimane a livello "1" (HIGH) per 5000 µS e al livello "0" (LOW) per lo stesso periodo. In altre parole, il ciclo "T" a livello "1" e livello "0" viene prodotto sul pin 2 e dura per 10000 µS. Poichè è costantemente ripetuto, un secondo contiene 100 cicli (1000000/10000). Questa è la frequenza F.

Registra il programma e assicurati che funzioni correttamente. Ciò significa un suono costante e fastidioso a una frequenza di 100 Hz.

Modifica il programma per ottenere suoni con diverse frequenze **F**; calcola i loro periodi **T** e semi-periodi **S** utilizzando la seguente tabella:

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 FUNZIONI DI SUONO

Il linguaggio di programmazione Arduino rende molto facile produrre tutti i tipi di suoni. Non dovrai elaborare né il periodo P né il semi periodo S. Tutto quello che devi fare è indicare la frequenza F e Arduino si prenderà cura del resto.

- **La funzione Tone()**

Genera un suono con la frequenza F richiesta sul pin di output e per la durata indicata.

Sintassi:

tone(pin, frequency, duration);

pin: indica su quale pin si stia per generare il suono

frequency: questo è un numero intero senza segno (unsigned int) che rappresenta la frequenza del tono in Hz (hertz). Ricorda che gli esseri umani sono in grado di ascoltare solo le frequenze sonore tra 20 Hz e 20.000 Hz (20 KHz).

duration: questo parametro è facoltativo. È un numero senza segno che rappresenta la durata del tono in millisecondi (se non c'è alcuna indicazione il tono continua indefinitamente o finché non viene eseguita la funzione noTone()).

Esempi:

```
int Pin=11;
int F = 1200;
int D = 3000;
tone(2,F,D);            // Genera un tono di 1200 Hz per 3 secondi sul pin 2
tone(6,F*10,500);       // Genera un tono di 12 KHZ per 0,5 secondi sul pin 2
tone(Pin,200); // Genera un tono di 200 Hz in modo indefinito sul pin 11
```

- **La funzione noTone():**

Arresta un tono su un pin specificato.

Sintassi:

noTone(pin);

pin: rappresenta su quale pin si trovi il suono

Esempi:

```
tone(2,13000); // Genera un tono a tempo indefinito di 13 KHz sul pin 2
….
….                    // Il tono continua a suonare
….
noTone(2);            // Il tono si arresta
```

SEZIONE PRATICA

5. ESEMPIO 1: Luci lampeggianti

Questo è l'esempio di programma più semplice da immaginare: un LED bianco lampeggiante collegato al pin 6 sul pannello di base I / O BASIC. Date un'occhiata alla soluzione in figura:

La variabile "Time" contiene la durata della pausa, 500 mS = 0.5 secondi. Il pin 6, quello cui è collegato il LED bianco, viene configurato come output.

Il corpo principale del programma contenuto nella funzione **loop()** si limita ad accendere e spegnere il pin 6 agli intervalli definiti dalla variabile "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

- **Ora è il tuo turno**

Con questo esempio puoi eseguire diverse attività. Ecco alcuni suggerimenti:

1. È possibile provare a variare l'intervallo di pausa utilizzando la variabile "Time". Vi suggerisco di abbreviarlo fino a raggiungere 10 mS. Cosa notate?

Sembra che il LED bianco sia sempre attivo ma questo non è il caso. Il LED è in corso e disattiva molto rapidamente. Infatti la nostra retina non è in grado di rilevare tali cambiamenti rapidi e crea un'illusione ottica: percepiamo che il LED sia permanente.

Il tempo è veramente un concetto elastico! Quello che sembra costante per te non lo è per il controller. Deve mantenere il LED acceso per 10 mS e mantenerlo fuori per altri 10 mS.

2. Invece di utilizzare la funzione delay() provare ad usare delaymicroseconds(). È possibile creare pause estremamente brevi e precise: al milionesimo di secondo (µS).

3. È anche possibile modificare il programma per simulare un faro. Ad esempio, è possibile che il LED emetta un lampeggio di 0,1 secondi ogni 2 secondi.

6. ESEMPIO 2: Set di luci

Le quattro luci LED sul pannello di base I / O di BASIC si stanno accendendo una dopo l'altra per la stessa quantità di tempo a partire da quella bianca. Ti darà una sensazione di movimento da destra a sinistra.

La programmazione potrebbe essere un po' lunga, ma è davvero semplice. Accende la luce bianca, la interrompe, lo spegne e poi accende la verde. Poi mette in pausa la verde, la spegne e accende la luce l'ambra. Successivamente ferma la luce ambra, la spegne e poi accende la rossa. Infine interrompe la rossa, la spegne e il ciclo ricomincia.

- **Ora è il tuo turno**

  1. Modificare i tempi: l'idea è quella di andare più veloce o più lento quando si sposta da destra a sinistra.
  2. Accendere e spegnere le luci da destra a sinistra e da sinistra a destra.

7. ESEMPIO 3: Segnale di traffico V1

Che ne dici di fare un breve progetto con tutte le cose che hai imparato? Suggerisco di simulare un semplice segnale di traffico.

Il programma in quanto tale non offre davvero nulla di nuovo. Si tratta solo di accendere e spegnere i LED verde, ambra e rosso nell'ordine giusto per il tempo indicato.

- **Conclusione**

Si noti che tutte le funzioni del programma che stabiliscono la sequenza di lavoro del segnale di traffico sono incluse nella funzione **setup()**; la funzione **loop()** è vuota.

Devi includere la funzione **loop()** anche se non ha nulla all'interno. D'altra parte tutte le funzioni incluse nella funzione **setup()** vengono eseguite solo una volta. Quindi, ogni volta che vuoi che il tuo segnale di traffico esegua una nuova sequenza, dovrai premere nuovamente il pulsante RESET.

8. ESEMPIO 4: Campana elettrica

Ecco un altro progetto semplice. Questa volta stai per simulare una campana elettrica. Ogni volta che si preme il pulsante, la campana emette un suono costituito da tre diverse tonalità.

Questa sarà la prima volta che utilizzerai la funzione tone() e vedrai che è veramente molto semplice. Tutto quello che devi fare è stabilire il pin di output, la frequenza e la durata (quest'ultima è facoltativa).

La soluzione al programma è contenuta nella figura. Viene generato un tono iniziale di 1000 Hz per 0,2 secondi. Successivamente, dopo una pausa di 0,4 secondi, viene generato un altro tono di 15000 Hz per 0,3 secondi. Infine, dopo 0,4 secondi, viene generato un terzo tono di 2000 Hz per 0,4 secondi.

Troviamo i toni sul pin 2 del controller. È qui collegato il piccolo altoparlante incluso nella baseboard I/O BASIC.

Prendi nota che tutte le funzioni sono incluse nella funzione **setup()** quindi la sequenza viene eseguita ogni volta che si preme RESET.

- **Ora è il tuo turno**

Stai per migliorare la campana aggiungendo una luce di segnalazione. Oltre alla sequenza precedentemente descritta, suggerisco di accendere il LED bianco ogni volta che premi RESET. Una volta generata la terza tonalità e terminata la sequenza, la luce del LED dovrebbe spegnersi.

9. ESEMPIO 5: Melodie

Questo esempio combina una serie di note musicali di diverse frequenze e durate che generano una melodia ben nota. Per semplificarlo, ogni nota musicale corrisponde ad una certa frequenza. Otteniamo le note acute e piatte e le combinazioni di queste frequenze. Dai un'occhiata e prova combinazioni diverse!

10. ESEMPIO: Segnali di traffico V2

Ecco un altro piccolo progetto. È una versione migliorata del segnale stradale V1: abbiamo aggiunto alcuni segnali acustici di avvertimento per i non vedenti.

La programmazione è molto sequenziale. Si inizia illuminando la luce verde allo stesso tempo in cui generano sei segnali da 1 KHz per 0,5 secondi. Successivamente si illumina la luce ambra e genera sei note di KHZ da 0,4 secondi. Infine, la luce rossa si attiva e genera una nota da 1 KHz per 6 secondi.

Poiché tutte le funzioni del programma sono incluse nella funzione setup(), vengono eseguite solo una volta: ogni volta che si preme il pulsante RESET la sequenza si avvia.

- **Ora è il tuo turno**

Noterete che l'ora in cui ogni luce rimane attiva è in realtà molto più breve della sua lunghezza effettiva.

L'idea è quella di modificare il tuo programma in modo da simulare il modo in cui funzionano i veri segnali del traffico.

RIFERIMENTI

PUBBLICAZIONI

[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

SITI 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