

# 5

---

## Music

---

### Melodies

This section uses Atari Logo to make tunes, to combine them to make bigger ones, and to manipulate melodic elements by playing them backward and transposing them upward and downward. The section concludes with a pitch and rhythm sequencer.

#### *Playing a Tune*

TUNE lets you play single-voice melodies. It takes two inputs, a list of notes to play and a duration, and plays each note in the list with that duration.

```
TO TUNE :LIST :DUR
IF EMPTY? :LIST [STOP]
IF (FIRST :LIST) = "R [TOUT 0 15000 0 :DUR]
      [TOUT 0 PITCH FIRST :LIST 15 :DUR]
TUNE BUTFIRST :LIST :DUR
END
```

The notes are represented as positive or negative integers. The letter R in a list is interpreted as silence (that is, a rest).

Try the following melody. Type:

```
SETENV 0 1
TUNE [1 1 8 8 10 10 8 R 6 6 5 5 3 3 1 R] 30
```

The melody you just played is "Twinkle, Twinkle Little Star." The amplitude (loudness) of each of the notes of the melody cannot change.

#### *Duration Using* TUNE

TUNE gives the same duration to each note in the list. You might try different durations with the same list of notes. For example:

TUNE [1 3 5 1] 40

or

TUNE [1 3 5 1] 20



The difference in these two melodies is that the first is played twice as fast as the second.

### *Numeric Pitch Representation in TUNE*

Here's how to translate between musical notations. Beneath each note is the letter for the note as well as the number that TUNE uses for that note.



The following is a melody using this notation. You see the traditional music notation for "Twinkle, Twinkle Little Star" along with the traditional note names. Underneath is the list of numbers TUNE uses to reproduce that melody.



As we mentioned earlier, R is interpreted as a rest. What TUNE really does when it sees an R is make the frequency of the note so high that you can't hear it! (That's what the high frequency of 15,000 is doing in TUNE.)

### *Putting Melodies Together*

You can give a list of notes a name. For example:

MAKE "TWINKLE1 [1 1 8 8 10 10 8 R 6 6 5 5 3 3 1 R]

Now type:

TUNE :TWINKLE1 30

You can put different melodies together. Here we will stay with "Twinkle, Twinkle" and make another list of notes as a continuation of the melody. Let's give it the name TWINKLE2.

## MUSIC

```
MAKE "TWINKLE2 [8 8 6 6 5 5 3 R]
```



Now type:

```
TUNE :TWINKLE2 30
```

Because this part of the melody is normally repeated, it's exactly half as long as TWINKLE1! You can use the REPEAT command to play it twice. Type:

```
REPEAT 2 [TUNE :TWINKLE2 30]
```

The SONG procedure combines TWINKLE1 and TWINKLE2 to make the entire melody.

```
TO SONG :DURATION
TUNE :TWINKLE1 :DURATION
REPEAT 2 [TUNE :TWINKLE2 :DURATION]
TUNE :TWINKLE1 :DURATION
END
```

To hear it, type:

```
SONG 30
```

The *tempo* of SONG (that is, the rate at which the notes follow each other) is determined by its duration input. To play SONG at a faster tempo, type:

```
SONG 20
```

### How TUNE Works

TUNE goes through its list of notes, one element at a time, and plays each note at the duration you specified. TUNE calls T00T for each note.

PITCH converts each note number in the notes list to its corresponding frequency. PITCH uses PITCH1 to help.

```
TO PITCH :NOTENUMB
OP PITCH1 :NOTENUMB 220
END

TO PITCH1 :NOTENUMB :BASE
IF :NOTENUMB = 0 [OP INT :BASE]
IF :NOTENUMB > 12 [OP PITCH1 :NOTENUMB-12 :BASE*2]
IF :NOTENUMB < 0 [OP PITCH1 :NOTENUMB+12 :BASE/2]
[OP PITCH1 :NOTENUMB - 1 :BASE*1.0595]
END
```

The note in this program is A at frequency 220. The A an octave above it has a frequency of 440. In fact, frequencies of notes an octave apart are

always related to each other in this way: going an octave higher doubles the frequency, going an octave lower halves the frequency. The variable :BASE is doubled or halved to perform this octave-changing function.

There are twelve chromatic steps in an octave. Therefore, the frequency of each note in the scale is the twelfth root of two higher than the next. Multiplying a note by 1.0595 gets the next note in the scale. The chromatic steps in between the octaves are determined by multiplying :BASE by 1.0595  $n$  times, where  $n$  is the number of chromatic steps.

TUNE's note numbers start at a frequency of 220. That A is 1, and all the notes in the scale go up or down from there.

### *Symmetry in Melodies*

One way to listen to the characteristics of a melody is to hear it backward. (This is similar to the kind of analysis sometimes done in a painting class. People will often look at a painting sideways or upside down in order to concentrate on the shapes and colors rather than the figures themselves.)

```
TO REVERSE :LIST
IF EMPTY :LIST [OP []]
OP FPUT LAST :LIST REVERSE BL :LIST
END
```

You use REVERSE to put a list of notes in reverse order. Try it by typing:

```
PRINT REVERSE [1 3 5 6]
```

You should get:

```
6 5 3 1
```

as your result. You can compare the sound of a list of notes forward and backward using TUNE with and without REVERSE. Type and listen to the following:

```
TUNE [1 3 5 6] 30
```

To hear the reverse of it, type:

```
TUNE REVERSE [1 3 5 6] 30
```

Also try:

```
TUNE :TWINKLE1 30
```

```
TUNE REVERSE :TWINKLE1 30
```

You can use REVERSE to build a symmetrical melody from a short one. The procedure MUSIC.MIRROR takes a list of notes and plays it in the given order, then plays it in reverse order. (By the way, this reversing process is usually called taking the *retrograde* of the phrase.)

```
TO MUSIC.MIRROR :LIST :DUR
TUNE :LIST :DUR
TUNE REVERSE :LIST :DUR
END
```

## MUSIC

Listen to the following examples, the first using our four-note phrase and the second using the tune list : TWINKLE1.

```
MUSIC.MIRROR [1 3 5 6] 30
MUSIC.MIRROR :TWINKLE1 30
```

In the next example, we construct a substantial melody with only two four-note phrases. We use the same four notes played before and make up another melody. Type the following:



```
MAKE "TUNE1 [1 3 5 6]
MAKE "TUNE2 [1 5 8 8]
```

Now we put these two melodies together in the procedure SONG1.

```
TO SONG1
REPEAT 2 [MUSIC.MIRROR :TUNE1 35 MUSIC.MIRROR :TUNE2 35]
END
```

Play it by typing:

```
SONG1
```

With eight notes we have been able to construct a twenty-four-note song! Try other melodies of your own design.

### *Transposing a Melody*

A melody has a certain shape or contour that can be preserved regardless of the pitch at which the melody starts. If you add or subtract a musical step (or several steps) from each note in a melody, you don't change its overall shape. This process of raising or lowering all the notes equally is called *transposing*.

### *Transposing Up*

TRANPOSE.UP transposes all the notes of a phrase up. TRANPOSE.UP works by adding its second input to each of the numbers in its input list.

Try:

```
TUNE TRANPOSE.UP [1 3 5 6] 2 30
```

This is equivalent to typing:

```
TUNE [3 5 7 8] 30
```

```
TO TRANPOSE.UP :LIST :INT
IF EMPTY? :LIST [OP []]
OP FPUT (FIRST :LIST) + :INT TRANPOSE.UP BF :LIST :INT
END
```



Listen to the difference between the list, before and after it has been transposed up.

Type:

TUNE [1 3 5 6] 30



and then

TUNE TRANPOSE.UP [1 3 5 6] 2 30



### *An Effect Using* TRANPOSE.UP

CLIMB and CLIMBING use TRANPOSE.UP to create the effect of a tune climbing. CLIMBING takes a list of notes as its first input and transposes it up step by step as many times as you want. The number of steps is CLIMB's second input.

Try:

CLIMB :TUNE1 3

CLIMB :TUNE2 7

CLIMB :TUNE3 5

TO CLIMB :NOTELIST :TIMES

CLIMBING :NOTELIST :TIMES 0

END

TO CLIMBING :NOTELIST :TIMES :UP

IF :TIMES = :UP [STOP]

TUNE TRANPOSE.UP :NOTELIST :UP 35

CLIMBING :NOTELIST :TIMES :UP+1

END

### *Transposing Down*

TRANPOSE.DOWN is similar to TRANPOSE.UP except that the melody is transposed *down*.

TO TRANPOSE.DOWN :LIST :INT

IF EMPTY :LIST [OP []]

OP FPUT (FIRST :LIST) - :INT TRANPOSE.DOWN BF :LIST :INT

END

Type:

MAKE "TUNE3 [5 8 6 5]

Then listen to each of the following:

TUNE :TUNE3 30

TUNE TRANPOSE.DOWN :TUNE3 1 30

TUNE TRANPOSE.DOWN :TUNE3 3 30

### A Single-Voice Music Sequencer

A music sequencer is an instrument that will repeat a sequence of notes for an indefinite period of time. We can make one by modifying TUNE.

```
TO SEQUENCER :LIST :DUR
IF EMPTY :LIST [STOP]
IF (FIRST :LIST) = "R [T00T 0 15000 15 :DUR]
    [T00T 0 PITCH FIRST :LIST 15 :DUR]
SEQUENCER (SE BUTFIRST :LIST FIRST :LIST) :DUR
END
```

Type:

SEQUENCER [6 5 3 1] 30



Press BREAK to stop.

The big difference between TUNE and SEQUENCER is that SEQUENCER repeats your tune over and over. It won't stop until you press BREAK. In TUNE, the first note is played, then removed on the recursive call.

### Short Durations

If the duration of the notes gets very short, you may want to change the "envelope" of the voice—that is, the rate at which the sound goes to silence (or decays) after its duration has been expended. This prevents a note from "spilling" into the next note.

SETENV's first input determines the voice. Since we're using voice 0 in SEQUENCER, the first input to SETENV should be 0. It's the second input to SETENV that determines the decaying time for the note. Try SETENV 0 1, which is a quick decay.

Type:

```
SETENV 0 1
SEQUENCER [6 5 3 1] 30
```

Try other values. To restore SETENV values, type:

```
SETENV 0 0
```

### A Single-Voice Rhythm Sequencer

RHYTHM.SEQ is a single-voice rhythm sequencer that makes bongolike sounds. This procedure expects its list of notes to include only the letters H, M, and L (for *high*, *medium*, and *low*) and R (for *rest*). Type the following two lines and listen to the result, pressing BREAK to stop.

```
SETENV 0 1
RHYTHM.SEQ [H M L L H] 10
```



Press **BREAK** to stop.

The second input is the duration for each of the notes in the list.

```
TO RHYTHM.SEQ :LIST1 :DUR
IF EMPTY :LIST1 [STOP]
IF EQUALP FIRST :LIST1 "R [TOOT 0 15000 0 :DUR]
  [TOOT 0 BONGO FIRST :LIST1 15 :DUR]
RHYTHM.SEQ (SE BUTFIRST :LIST1 FIRST :LIST1) :DUR
END
```

Traditionally speaking, rhythm usually implies periodic or repeating patterns. The list of elements that you have given **RHYTHM.SEQ** becomes such a pattern as it continues to repeat. For example, type:

```
RHYTHM.SEQ [L M H R R] 20
```

This is a five-beat pattern that gets its rhythm from the sequencing action alone.

A second way to produce rhythmic patterns is to use rests in different ways. Since **RHYTHM.SEQ** doesn't have many notes, you can concentrate on how far apart to space them in time. For example, type:

```
RHYTHM.SEQ [L R M H R M L R M H] 20
```

This has an internal feeling of three (waltzlike), yet it is a ten-beat pattern. Both rhythms seem to coexist.

A third way in which to construct patterns is with the low, medium, and high pitches. They can be used to either reinforce the existing patterns or they can serve as counterpoint to them. For example, type:

```
RHYTHM.SEQ [L M H R L M H L R H] 20
```



This rhythm reinforces the patterns of the previous ten-beat pattern by repeating the low-medium-high sequence almost three times (there's a rest in the middle of one of them) and by adding an additional rest between one of these repetitions to get the ten-beat phrase. Try the previous rhythmic sequences at a faster tempo by typing:

```
RHYTHM.SEQ [L R M H R M L R M H] 10
RHYTHM.SEQ [L M H R L M H L R H] 10
```

### **How** **RHYTHM.SEQ Works**

**RHYTHM.SEQ** was designed by modifying **SEQUENCER**. The most conspicuous difference is that **RHYTHM.SEQ** uses a procedure called **BONGO** instead of **PITCH** to produce the **TOOT** frequencies.

```
TO BONGO :NOTE
IF :NOTE = "L [OP (59 + RANDOM 3)]
IF :NOTE = "M [OP (74 + RANDOM 3)]
IF :NOTE = "H [OP (87 + RANDOM 3)] [OP 15000]
END
```



BONGO interprets L, M, and H for RHYTHM.SEQ. It interprets R (and any other character) as a rest by outputting (OP) an inaudible frequency of 15,000.

As in SEQUENCER, if the rhythm is very fast, the second input to SETENV should be very small so that there is separation between notes.

Notice that the frequencies for L, M, and H are small numbers and, thus, relatively low notes. These frequencies will change slightly each time depending on the tiny RANDOM values that are added to them. This has been done to make the sounds more bongolike and less, for example, pianolike—that is, less “pitchy.”

---

#### PROGRAM LISTING

---

```

TO TUNE :LIST :DUR
IF EMPTY :LIST [STOP]
IF (FIRST :LIST)="R [TOOT 0 15000 0 ►
:DUR] [TOOT 0 PITCH FIRST :LIST ►
15 :DUR]
TUNE BUTFIRST :LIST :DUR
END

TO SONG :DURATION
TUNE :TWINKLE1 :DURATION
REPEAT 2 [TUNE :TWINKLE2 :DURATION]
TUNE :TWINKLE1 :DURATION
END

TO PITCH :NOTENUMB
OP PITCH1 :NOTENUMB 220
END

TO PITCH1 :NOTENUMB :BASE
IF :NOTENUMB = 0 [OP INT :BASE]
IF :NOTENUMB > 12 [OP PITCH1 ►
:NOTENUMB-12 :BASE*2]
IF :NOTENUMB < 0 [OP PITCH1 ►
:NOTENUMB+12 :BASE/2] [OP PITCH1 ►
:NOTENUMB - 1 :BASE*1.0595]
END

TO REVERSE :LIST
IF EMPTY :LIST [OP []]
OP FPUT LAST :LIST REVERSE BL :LIST
END

TO MUSIC.MIRROR :LIST :DUR
TUNE :LIST :DUR
TUNE REVERSE :LIST :DUR
END

```

```

TO SONG1
REPEAT 2 [MUSIC.MIRROR :TUNE1 35 ►
MUSIC.MIRROR :TUNE2 35]
END

TO TRANSPOSE.UP :LIST :INT
IF EMPTY :LIST [OP []]
OP FPUT (FIRST :LIST) + :INT ►
TRANSPOSE.UP BF :LIST :INT
END

TO CLIMB :NOTELIST :TIMES
CLIMBING :NOTELIST :TIMES 0
END

TO CLIMBING :NOTELIST :TIMES :UP
IF :TIMES = :UP [STOP]
TUNE TRANSPOSE.UP :NOTELIST :UP 35
CLIMBING :NOTELIST :TIMES :UP+1
END

TO TRANSPOSE.DOWN :LIST :INT
IF EMPTY :LIST [OP []]
OP FPUT (FIRST :LIST) - :INT ►
TRANSPOSE.DOWN BF :LIST :INT
END

TO SEQUENCER :LIST :DUR
IF EMPTY :LIST [STOP]
IF (FIRST :LIST) = "R [TOOT 0 15000 15 ►
:DUR] [TOOT 0 PITCH FIRST :LIST ►
15 :DUR]
SEQUENCER (SE BUTFIRST :LIST FIRST ►
:LIST) :DUR
END

```

```

TO RHYTHM.SEQ :LIST1 :DUR
IF EMPTY :LIST1 [STOP]
IF EQUALP FIRST :LIST1 "R [TOOT 0 ►
15000 0 :DUR] [TOOT 0 BONGO FIRST ►
:LIST1 15 :DUR]
RHYTHM.SEQ (SE BUTFIRST :LIST1 FIRST ►
:LIST1) :DUR
END

TO BONGO :NOTE
IF :NOTE = "L [OP (59 + RANDOM 3)]
IF :NOTE = "M [OP (74 + RANDOM 3)]
IF :NOTE = "H [OP (87 + RANDOM 3)] [OP ►
15000]
END

```

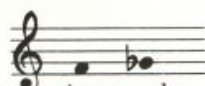
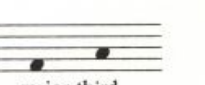
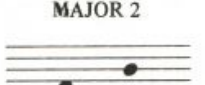
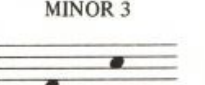
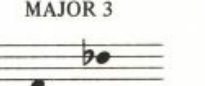
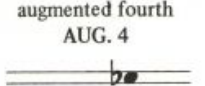
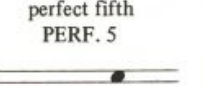
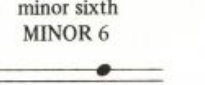
## Ear Training

This project is an interactive music tutorial in ear training. It gives you the opportunity to listen to musical intervals and learn to recognize them.

### *How to Use the Ear Training Tutorial*

The program picks two notes at random to construct a musical interval. Your task is to hear the interval and to select what you think it is. The program will tell you if you are right or give you the correct answer.

The program selects intervals within roughly one octave. They are shown here in traditional musical notation, with a written description, and with the abbreviated notation this program uses.

 minor second MINOR 2	 major second MAJOR 2	 minor third MINOR 3	 major third MAJOR 3
 perfect fourth PERF. 4	 augmented fourth AUG. 4	 perfect fifth PERF. 5	 minor sixth MINOR 6
 major sixth MAJOR 6	 minor seventh MINOR 7	 major seventh MAJOR 7	 octave OCTAVE

## MUSIC

*Running the Program*

Those of you who feel bold can run the tutorial without reading this section.

Type:

EAR.TRAINING

The following is a step-by-step description of how to use the program.

The program prints out the following instructions.

```
THIS PROGRAM PLAYS A NOTE
THEN ANOTHER...
...AND ASKS YOU FOR THE PITCH-
INTERVAL BETWEEN THEM.
```

FOR EXAMPLE, IF THE FIRST NOTE IS:

(a sound here)

AND THE SECOND IS:

(a sound here)

THE INTERVAL BETWEEN THEM IS A PERF.4

THE POSSIBLE INTERVALS ARE:

```
MINOR.2 MAJOR.2 MINOR.3
MAJOR.3 PERF.4 AUG.4
PERF.5 MINOR.6 MAJOR.6
MINOR.7 MAJOR.7 OCTAVE
MINOR.9 MAJOR.9 MINOR.10
MAJOR.10
```

HERE WE GO

Then the program gives you two notes and states:

```
TYPE ? AND RETURN FOR THE INTERVAL,
GIVE YOUR OWN ANSWER OR
PRESS RETURN FOR ANOTHER HEARING
```

Let's say you type:

MAJOR.6

There are two possibilities: either you are right, in which case it responds:

TRUE

or you are wrong, in which case it responds:

FALSE. THE RIGHT ANSWER IS: *whatever*

If you had typed ? and RETURN the program would have informed you of the interval you had just heard.

The program then gives you instructions.

PRESS RETURN TO CONTINUE, OR  
ANY CHARACTER AND RETURN TO STOP.

Pressing RETURN gives you a new interval. If, when you hear an interval, you are not sure of the answer, you may listen a second time. When the program states:

TYPE ? AND RETURN FOR THE INTERVAL,  
GIVE YOUR OWN ANSWER OR  
PRESS RETURN FOR ANOTHER HEARING

pressing RETURN causes the same interval to be repeated, this time with the notes played together as well as one after the other.

---

#### PROGRAM LISTING

---

TO EAR.TRAINING	PRINT []
CT	WAIT 180
PRINT [THIS PROGRAM PLAYS A NOTE,]	EAR.TRAINING2
WAIT 60	END
PRINT [THEN ANOTHER...]	
WAIT 120	TO EAR.TRAINING2
PRINT [...AND ASKS YOU FOR THE PITCH-]	CT
PRINT [INTERVAL BETWEEN THEM.]	PRINT [IF THE FIRST NOTE IS:]
WAIT 180	MAKE "BASE (RANDOM 6) + 1
PRINT []	WAIT 30
PRINT []	T00T 0 PITCH :BASE 15 60
PRINT [FOR EXAMPLE, IF THE FIRST NOTE ▶	WAIT 120
IS:]	PRINT [AND THE SECOND IS:]
T00T 0 PITCH 2 15 60	MAKE "INTERVAL (RANDOM 16) + 1
WAIT 120	WAIT 30
PRINT [AND THE SECOND IS:]	T00T 0 PITCH :BASE + :INTERVAL 15 60
WAIT 60	WAIT 60
T00T 0 PITCH 7 15 60	PRINT []
WAIT 180	PRINT [TYPE ? AND RETURN FOR THE ▶
PRINT [THE INTERVAL BETWEEN THEM IS A ▶	INTERVAL,]
PERF.4]	PRINT [GIVE YOUR OWN ANSWER OR]
WAIT 100	PRINT [PRESS RETURN FOR ANOTHER ▶
PRINT []	HEARING.]
PRINT []	CHECK.ANSWER :BASE :INTERVAL RL
PRINT [THE POSSIBLE INTERVALS ARE:]	TRY.AGAIN
PRINT []	END
PRINT [MINOR.2 MAJOR.2 MINOR.3]	
PRINT [MAJOR.3 PERF.4 AUG.4]	TO CHECK.ANSWER :BASE :INTERVAL :ANS
PRINT [PERF.5 MINOR.6 MAJOR.6]	IF EMPTY :ANS [ANOTHER.HEARING :BASE ▶
PRINT [MINOR.7 MAJOR.7 OCTAVE]	:INTERVAL STOP]
PRINT [MINOR.9 MAJOR.9 MINOR.10]	IF (FIRST :ANS) = "?" [PRINT (SE [THIS ▶
PRINT [MAJOR.10]	INTERVAL IS A] INTERVALS ▶
WAIT 240	:INTERVAL) WAIT 60] [CHECKLIST ▶
PRINT []	:INTERVAL FIRST :ANS]
PRINT [HERE WE GO]	END



```

TO CHECKLIST :INTERVAL :ANS
IF (INTERVALS :INTERVAL) = :ANS [PRINT ►
  TRUE] [PRINT (SE [FALSE. THE ►
    RIGHT ANSWER IS:] INTERVALS ►
    :INTERVAL]
WAIT 60
END

TO INTERVALS :NUMBER
OP ITEM :NUMBER [MINOR.2 MAJOR.2 ►
  MINOR.3 MAJOR.3 PERF.4 AUG.4 ►
  PERF.5 MINOR.6 MAJOR.6 MINOR.7 ►
  MAJOR.7 OCTAVE MINOR.9 MAJOR.9 ►
  MINOR.10 MAJOR.10]
END

TO ITEM :NUM :LIST
IF :NUM=1 [OP FIRST :LIST]
OP ITEM :NUM-1 BF :LIST
END

TO ANOTHER.HEARING :BASE :INTERVAL
PRINT []
PRINT [TOGETHER, THE NOTES ARE:]
WAIT 60
TOOT 0 PITCH :BASE 15 120
TOOT 1 PITCH :BASE + :INTERVAL 15 90
WAIT 120
PRINT [AGAIN, THE FIRST NOTE IS:]
TOOT 0 PITCH :BASE 15 120
PRINT [AND THE SECOND:]

TOOT 1 PITCH :BASE + :INTERVAL 15 90
PRINT []
PRINT [TYPE ? AND RETURN FOR THE ►
  INTERVAL,]
PRINT [GIVE YOUR OWN ANSWER OR]
PRINT [PRESS RETURN FOR ANOTHER ►
  HEARING.]
CHECK.ANSWER :BASE :INTERVAL RL
END

TO TRY.AGAIN
PRINT []
PRINT [PRESS RETURN TO CONTINUE, OR]
PRINT [ANY CHARACTER AND RETURN TO ►
  STOP.]
IF EMPTY RL [EAR.TRAINING2]
END

TO PITCH :NOTENUMB
OP PITCH1 :NOTENUMB 220
END

TO PITCH1 :NOTENUMB :BASE
IF :NOTENUMB = 0 [OP INT :BASE]
IF :NOTENUMB > 12 [OP PITCH1 ►
  :NOTENUMB-12 :BASE*2]
IF :NOTENUMB < 0 [OP PITCH1 ►
  :NOTENUMB+12 :BASE/2] [OP PITCH1 ►
  :NOTENUMB - 1 :BASE*1.0595]
END

```

## Sound Effects

This write-up presents a palette of sound effects to give you ideas for using sound in your own projects. We've kept our discussion brief. Instead, we ask you to use your ears. Some of these sound effects are new, others are taken from projects found elsewhere in this book. You might want to try them out and use those effects you like in your own projects, either as they are or in modified form. The procedures in this collection use the Atari Logo music primitives TOOT and SETENV.

### *A European Ambulance Siren*

Typing the following lines results in a European ambulance sound.

---

By Greg Gargarian and Margaret Minsky; with contributions by Max Behensky.



```
SETENV 0 10
REPEAT 10 [T00T 0 267 15 40 T00T 0 200 15 30])
```

### *Advancing and Retreating Sounds*

The procedures ADVANCE and RETREAT make sounds like something is rushing toward you or retreating from you. When sound sources advance toward you, you hear their pitch rising slightly and their volume increasing. When they retreat, you hear a falling pitch and decreasing volume. That is what these procedures try to do.

ADVANCE and RETREAT take two inputs. The first is a starting pitch for their sound and the second is FAST or SLOW for the speed of the advance or retreat. These procedures make sounds in both voice 0 and voice 1.

Here are the procedures for ADVANCE.

```
TO ADVANCE :PCH :DURATION
IF :DURATION = "FAST [ADVANCE1 :PCH 1]
IF :DURATION = "SLOW [ADVANCE2 :PCH 5]
END
```

```
TO ADVANCE1 :PCH :AMP
IF :AMP > 15 [STOP]
T00T 0 :PCH :AMP 5
T00T 1 :PCH*1.01 :AMP 5
ADVANCE1 :PCH*1.01 :AMP + 3
END
```

```
TO ADVANCE2 :PCH :AMP
IF :AMP > 15 [STOP]
T00T 0 :PCH :AMP 5
T00T 1 :PCH*1.01 :AMP 5
ADVANCE2 :PCH*1.01 :AMP + 1
END
```

Try:

```
SETENV 0 1
ADVANCE 440 "SLOW
ADVANCE 440 "FAST
```

Here are the procedures for RETREAT.

```
TO RETREAT :PCH :DURATION
IF :DURATION = "FAST [RETREAT1 :PCH 15]
IF :DURATION = "SLOW [RETREAT2 :PCH 15]
END
```

```
TO RETREAT1 :PCH :AMP
IF :AMP < 0 [STOP]
T00T 0 :PCH :AMP 5
T00T 1 :PCH*.99 :AMP 5
RETREAT1 :PCH*.99 :AMP - 3
END
```

## MUSIC

```

TO RETREAT2 :PCH :AMP
IF :AMP < 0 [STOP]
TOOT 0 :PCH :AMP 5
TOOT 1 :PCH*.99 :AMP 5
RETREAT2 :PCH*.99 :AMP - 1
END

```

Try:

```

RETREAT 440 "FAST
RETREAT 440 "SLOW

```

You might want to try some other inputs.

```

ADVANCE 1000 "FAST
ADVANCE 100 "SLOW
RETREAT 200 "SLOW

```

You can try putting them together.

```

REPEAT 10 [ADVANCE 200 "SLOW RETREAT 210 "FAST WAIT 10]

```

This one sounds like a monster snoring.

```

REPEAT 5 [ADVANCE 90 "FAST RETREAT 95 "SLOW WAIT 10]

```

***Making Sliding Sounds: Glissandi***

The RAMP procedure makes a sound that slides "smoothly" from a starting pitch to an ending pitch.

```

TO RAMP :START :FINISH :RATE
IF :FINISH < :START [REPEAT (:START-:FINISH)/ :RATE
  [TOOT 1 :START 15 2 MAKE "START :START - :RATE]]
IF :START < :FINISH [REPEAT (:FINISH-:START)/ :RATE
  [TOOT 1 :START 15 2 MAKE "START :START + :RATE]]
END

```

Try:

```

RAMP 400 1000 20
RAMP 1000 400 20
RAMP 300 500 70
RAMP 100 800 40

```

You get the idea. The first input is the starting frequency, the second is the ending frequency, and the third determines the rate of the slide. The bigger the third input, the faster the slide.

Try:

```

RAMP 500 700 4
RAMP 500 700 40

```

This one makes a "whooping" sound:

```
REPEAT 5 [RAMP 400 800 40]
```

### *A Motorcycle Sound*

MOTORCYCLE uses RAMP to make the motorcycle warm up and ADVANCE and RETREAT to make it drive away.

```
TO MOTORCYCLE
REPEAT 10 [RAMP 25 120 (RANDOM 18)+1]
ADVANCE 50 "SLOW
RETREAT 50 "SLOW
END
```

### *A More Continuous Sliding for an Ambulance Siren*

RAMP2 is a more complicated procedure that makes a more continuous slide. It can be used to make the sound of an ambulance siren.

```
TO RAMP2 :START :FINISH :RATE
IF :FINISH < :START
  [DOWN :START :FINISH :RATE TOOT 0 :FINISH 12 6]
IF :FINISH > :START
  [UP :START :FINISH :RATE TOOT 0 :FINISH 12 6]
END

TO UP :S :F :R
REPEAT (:F - :S) / :R
  [TOOT 1 :S 15 4 WAIT 1 TOOT 0 :S 12 4 MAKE "S :S + :R]
END

TO DOWN :S :F :R
REPEAT (:S - :F) / :R
  [TOOT 1 :S 15 4 WAIT 1 TOOT 0 :S 12 4 MAKE "S :S - :R]
END
```

Try:

```
REPEAT 10 [RAMP2 1200 1600 25 RAMP 1600 1200 25]
```

### *A Spaceship Sound*

DEPARTURE uses RAMP in voice 1 and holds the starting pitch of the RAMP in voice 0.

```
TO DEPARTURE :FIRST :LAST :RATE
IF 255 < (ABS (:FIRST-:LAST)/:RATE)*2 [TOOT 0 :FIRST 12 255]
  [TOOT 0 :FIRST 12 (ABS (:FIRST-:LAST)/:RATE)*2]
RAMP :FIRST :LAST :RATE
END
```

## MUSIC

```

TO ABS :NUM
IF :NUM < 0 [OP -:NUM] [OP :NUM]
END

```

Try DEPARTURE:

```

SETENV 0 8
SETENV 1 8
DEPARTURE 200 50 5
DEPARTURE 500 50 10

```

For a zigzag sound, try the following:

```
REPEAT 2 [DEPARTURE 1000 500 20 DEPARTURE 500 1000 20]
```

For a spaceshiplike sound, try:

```

TO SPACE.SHIP :NUM
REPEAT :NUM [DEPARTURE 1500 2000 20]
END

```

Try:

```
SPACE.SHIP 5
```

or

```
SPACE.SHIP 2
```

*A Boing-ng-ng Sound*

BOING works best in the low register . . . boings usually do!

BOING's first input is the frequency of the boing and the second input is the duration (in sixtieths of a second).

```

TO BOING :FR :DUR
SETENV 0 0
SETENV 1 0
IF :DUR > 100 [BOING1 15 1.5 :FR :FR/20 (:DUR/10) STOP]
BOING1 15 3 :FR :FR/20 (:DUR/5)
END

```

```

TO BOING1 :AMP :INC :FR :FR1 :DUR
IF :AMP < 1 [STOP]
TOOT 0 :FR :AMP :DUR
TOOT 1 :FR - ( :FR / 40 ) :AMP :DUR
BOING1 :AMP - :INC :INC ( :FR + :FR1 ) :FR1 :DUR
END

```

Try:

```

BOING 200 40
REPEAT 5 [BOING 50 30]

```

*Trills and Thrills*

The TRILL procedure plays a "trill" (a sound made up of alternating sounds). TRILL's first input is a frequency, the second input the interval, and the third is the number of times to alternate.

```
TO TRILL :FREQ :STEPSIZE :TIMES
REPEAT :TIMES [TOOT 0 :FREQ 15 5
                TOOT 0 :FREQ*STEPVALUE :STEPVALUE 15 5]
END
```

```
TO STEPVALUE :STEP
IF :STEP<2 [OP 1.0595]
IF :STEP=2 [OP 1.1225]
IF :STEP=3 [OP 1.1893]
IF :STEP=4 [OP 1.26]
IF :STEP=5 [OP 1.335]
IF :STEP=6 [OP 1.4145]
IF :STEP=7 [OP 1.4987]
IF :STEP=8 [OP 1.5878]
IF :STEP=9 [OP 1.6823]
IF :STEP=10 [OP 1.7824]
IF :STEP=11 [OP 1.888]
IF :STEP=12 [OP 2]
OP 2*STEPVALUE :STEP-12
END
```

Try:

```
TRILL 400 3 4
TRILL 400 3 8
TRILL 200 3 8
TRILL 200 2 8
TRILL 200 1 8
```

*More with Trill*

AGITATION is a procedure that uses TRILL to make trills of decreasing stepsize.

```
TO AGITATION :FREQ :NUMB
IF :NUMB < 1 [TOOT 0 :FREQ 5 30 STOP]
TRILL :FREQ :NUMB 4
AGITATION :FREQ :NUMB-1
END
```

Try:

```
AGITATION 200 4
AGITATION 1000 8
AGITATION 660 48
REPEAT 3 [AGITATION 50 4]
REPEAT 2 [AGITATION 2000 3]
```



## MUSIC

*Bird Sounds*

BIRDSONG and BIRDSONG1 are two different kinds of bird sounds. BIRDSONG uses RAMP to make a short and upward-gliding sound at a high frequency.

```
TO BIRDSONG
RAMP 1010 1070 10
WAIT (RANDOM 10)+1
END
```

```
TO BIRDSONG1
TRILL 1200 2 1
TRILL 1010 2 1
WAIT (RANDOM 10)+1
TRILL 1133 1 1
TRILL 801 2 1
WAIT (RANDOM 10)+1
END
```

Try:

**BIRDSONG**

BIRDSONG1 uses several short calls of TRILL to make a nervous-jumping bird sound.

Try:

**BIRDSONG1***Bird Music*

BIRDS makes a birdlike song using BIRDSONG and BIRDSONG1. To create variety BIRDSONG and BIRDSONG1 are played alternately, each a random number of times.

```
TO BIRDS
REPEAT (RANDOM 3)+1 [BIRDSONG]
WAIT (RANDOM 10)+1
REPEAT (RANDOM 3)+1 [BIRDSONG1]
WAIT (RANDOM 10)+1
BIRDS
END
```

Listen to it by typing:

**BIRDS**

Press the BREAK key to stop!

*Sound for Jack and Jill*

FANFARE is the music finale to the Jack and Jill project found in this book.

```

TO FANFARE
FANFARE1 55 110 7040
FANFARE1 35 70 7040
FANFARE1 30 60 7040
FANFARE1 25 50 7040
FANFARE1 30 50 7680
FANFARE1 120 200 7680
SETENV 0 15 SETENV 1 15
TOOT 0 240 15 240 TOOT 1 400 15 240
END

```

```

TO FANFARE1 :FR0 :FR1 :HIGHFR
IF :FR0 > :HIGHFR [STOP]
TOOT 0 :FR0 15 10 TOOT 1 :FR1 15 7
FANFARE1 :FR0*2 :FR1*2 :HIGHFR
END

```

Try:

**FANFARE**

Since FANFARE does SETENVs, you might want to restore by saying:

```

SETENV 0 0
SETENV 1 0

```

### *Playing with SETENV and Amplitudes*

BOUNCE makes a ping-ponglike bouncing sound.

BOUNCE's input is for the frequency. As the BOUNCE note gets faster, its amplitude gets quieter and, near the end, its envelope gets shorter.

```

TO BOUNCE :FREQ
SETENV 0 1
BOUNCE1 :FREQ 15 40
END

```

Try:

```

BOUNCE 440
BOUNCE 100
REPEAT 5 [BOUNCE (RANDOM 400)+100]

```

You can shorten the longer durations of the bounce by lowering the duration input to BOUNCE1 (that is, by changing 40 to a smaller number).

```

TO BOUNCE1 :FREQ :AMP :DUR
IF :AMP < 1 [MAKE "AMP ABS :AMP]
IF :DUR < 1 [FASTBOUNCE :FREQ STOP]
TOOT 0 :FREQ :AMP 10
WAIT :DUR
BOUNCE1 :FREQ :AMP-1 :DUR-(15-:AMP)
END

```

## MUSIC

You can shorten the tail of the bounce by lowering the input to REPEAT in FASTBOUNCE.

```
TO FASTBOUNCE :FREQ
SETENV 0 0
REPEAT 10 [TOOT 0 :FREQ 2 5 WAIT 5]
END
```

*An Echo Effect*

ECHO is similar to BOUNCE, but ECHO doesn't get faster as it gets quieter. It uses SETENV to gradually change the decay of the repeating notes and RANDOM to produce the frequency, starting envelope, and pulsing rate of the echoed note.

```
TO ECHO
ENDING (RANDOM 800)+50 RANDOM 7 (RANDOM 15)+8
ECHO
END
```

```
TO ENDING :FR :DECAY :RATE
IF :DECAY=0 [ENDING1 :FR 15 :RATE STOP]
SETENV 0 :DECAY
TOOT 0 :FR 15 :RATE
ENDING :FR :DECAY-1 :RATE
END
```

```
TO ENDING1 :FR :AMP :RATE
IF :AMP=0 [STOP]
TOOT 0 :FR :AMP :RATE
ENDING1 :FR :AMP-1 :RATE
END
```

Try:

```
ENDING 400 2 25
ENDING 100 1 10
```

Now type:

```
ECHO
```

Press BREAK to stop!

## Naming Notes

In the Melodies project, notes are represented by numbers that are later converted to appropriate frequencies. It takes time to do this for each note. If you want to play notes very rapidly one after the other, you can use another technique described in this project. The idea is to *precompute* the frequencies that correspond to the notes. In this project, we do this by giving names to these frequencies. The names we chose in this project come from traditional music notation, for example A#4, which represents the A-sharp in the fourth octave of piano pitches. We might make this a name by doing

```
MAKE "A#4 466
```

Thereafter (as in the Argue program) you can use `THING` to refer quickly to the frequency of a note. For example, if `:NOTE` is A#4, then `THING :NOTE` is 466. Using this scheme, your music procedures would contain lines like

```
TOOT 0 THING :NOTE 15 20
```

You might have to name a lot of notes. This means there will be a lot of variables, and they will use up quite a bit of Logo's workspace. Sometimes it is worth it.

This project shows a program that automatically creates note names like A#4 and figures out the right frequencies. It does this for several octaves.\*

### *Naming Notes*

This program uses the naming technique just described to allow fast symbolic access to musical notes. The names follow a convention similar to standard music notation where, for example, G3 would be the name for G in the third piano octave.

Some examples:

```
MAKE "G3 392
MAKE "G#3 415
MAKE "A4 440
MAKE "A#4 466
MAKE "B4 493
```

Since we want to have names like these for many notes, we create procedures that automatically calculate and name frequencies.

\*It is nice to use names like A#4, but the same technique can work with numbers (or anything else) as the names of variables for the notes. The main advantage of this project's technique is the use of variables for fast access to precomputed values. You could, for example, do `MAKE 38 466` to give the name 38 to the frequency 466.

### Procedures for Naming Notes

NAMENOTES calls NAMEOCTAVES, which calls NAMEOCTAVE, to name the notes in each octave. NAMENOTES also names the special "note" R so that it represents a rest.

```
TO NAMENOTES
MAKE "R 15000
NAMEOCTAVES [A A# B C C# D D# E F F# G G#] 1 55
END
```

NAMEOCTAVES takes three inputs: a list of prefixes for the names of notes ([A A# B C C# D D# E F F# G G#]), the starting suffix that is the number of the lowest octave for which names are to be created, and the frequency of the lowest pitch in that octave (the A of that octave).

```
TO NAMEOCTAVES :NAMES :OCTAVE :STARTFREQ
IF :OCTAVE > 8 [STOP]
NAMEOCTAVE :NAMES :OCTAVE :STARTFREQ
NAMEOCTAVES :NAMES :OCTAVE + 1 :STARTFREQ * 2
END
```

NAMEOCTAVES calls NAMEOCTAVE to make the variables for each octave. Each note name is created (in NAMEOCTAVE) by making a new word out of the appropriate prefix and the octave number. Then NAMEOCTAVES updates the octave number and the lowest frequency for the next octave. This continues until eight octaves of pitches have been named.

In NAMEOCTAVE the twelfth root of two is used in the following formula to compute the frequency of a note one half-step above another.

$$(\text{frequency of a note}) \times (\text{twelfth root of } 2) = (\text{frequency of next note})$$

```
TO NAMEOCTAVE :NAMES :OCTAVE :FREQ
IF EMPTY :NAMES [STOP]
MAKE ( WORD FIRST :NAMES :OCTAVE ) :FREQ
NAMEOCTAVE BF :NAMES :OCTAVE :FREQ * 1.0595631
END
```

### Playing Melodies with Named Notes

Run NAMENOTES to create the note variables. Now you can use a procedure such as PLAYTUNE to play a melody.

```
TO PLAYTUNE :LIST
IF EMPTY :LIST [STOP]
TOOT 0 THING FIRST :LIST 15 15
PLAYTUNE BF :LIST
END
```

You can use PLAYTUNE like this:

```
PLAYTUNE [A2 A#2 B2 C2 C#2 D2 D#2 E2 F2 F#2 G2 G#2 A3]
```

You could name lists that represent phrases of songs and play them with PLAYTUNE. Here are some examples.



```

MAKE "SCALE [C2 D2 E2 F2 G2 A3 B3 C3]
MAKE "SCALE2 [C3 D3 E3 F3 G3 A4 B4 C4]
MAKE "TWINKLE [C2 C2 G2 G2 A3 A3 G2 R
F2 F2 E2 E2 D2 D2 C2 R]
MAKE "FOLK [D4 C#4 D4 D4 D3 D3 F#3 F#3
A4 A4 B4 A4 B4 C#4 D4 D4]

```

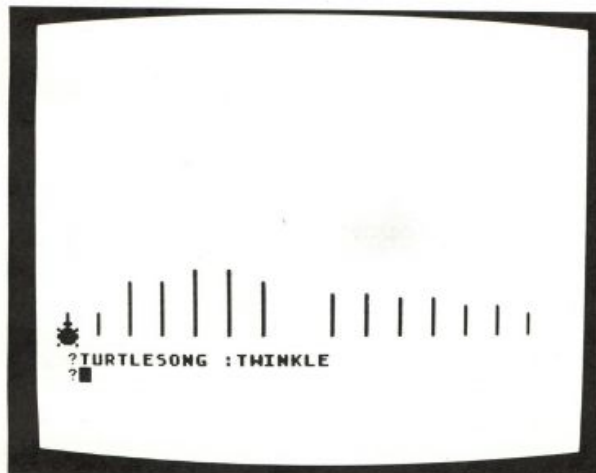
PLAYTUNE : TWINKLE

For more ideas about what you can do with melodies and rhythms, see the Melodies project.

### *Making Turtles Move to Your Song*

Here's a program that makes a turtle show the "ups and downs" of your song. You can use it by trying the procedure TURTLESONG with a list of notes as its input. For example, try

TURTLESONG : TWINKLE



Here are the procedures.

```

TO TURTLESONG :LIST
  SETUP.TURTLE
  SONG :LIST
  END

```

```

TO SETUP.TURTLE
  CS
  TELL 0
  PU LT 90 FD 150 RT 90 BK 60 PD
  ST
  SETPN 0
  SETPC 0 40
  END

```

## MUSIC

```

TO SONG :LIST
IF EMPTY :LIST [STOP]
MAKE "NOTE FIRST :LIST
IF :NOTE = "R [TOOT 0 THING :NOTE 15 40] [JUMP THING :NOTE]
PU RT 90 FD 20 LT 90 PD
SONG BF :LIST
END

```

```

TO JUMP :FREQ
MAKE "INT (:FREQ - 100) / 3
FD :INT
TOOT 0 :FREQ 15 40
WAIT 15
RT 180
FD :INT
RT 180
END

```

These procedures draw lines on the screen whose lengths represent pitches. They fit best for notes in the range from :A2(110) to :D4(640).

*Using the Atari Keyboard as a Music Keyboard*

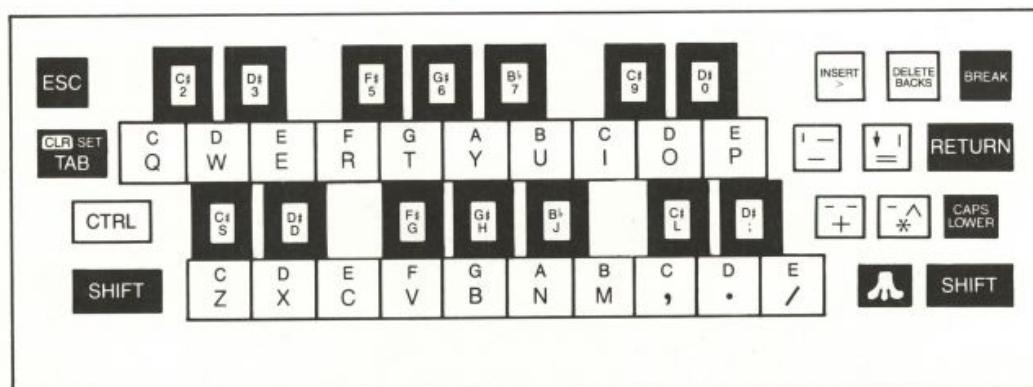
MUSIC is another example of using the precomputed notes; it makes the Atari keyboard act like a music keyboard. Start it up by typing

MUSIC

The program takes a while to set itself up, then it types

READY

Now you can play music by pressing keys. Notes are assigned to the keys according to a layout that is like that of a piano keyboard:



In order to make the layout similar to that of a piano, some of the keys do not play any note. Reminder: Take care not to press the Atari (/ \) key.

### The Music Keyboard Procedures

MUSIC also creates variables for fast reference. It creates variables whose names are names of Atari keyboard keys (for example Z, X, C) and whose values are names of notes. This is done by ASSIGNKEYS. For example, :S becomes C#3.

MUSIC then calls KEYBOARD. KEYBOARD waits for you to press keys. It converts the name of the key you pressed to the name of a note and calls NOTE to play it. NOTE converts the name of the note to a frequency and calls the Logo primitive T00T to make the sound.

There are a couple of fine points in this program. The length of time each note sounds is controlled by the global variable :DURATION, which is set up in the MUSIC procedure and by the SETENV commands in the procedure TEMPO. Also, the VOICE input to KEYBOARD makes it possible for you to press two keys in quick succession and hear both notes. The KEYBOARD procedure calls itself alternating between 0 and 1 as values for :VOICE. This allows the program to alternate the playing of notes between the Atari sound hardware voices 0 and 1. Thus one note keeps sounding in one voice while the program starts a second note sounding in the other voice.

Reminder: After using the MUSIC program, you may want to restore the music envelope decay to its initial state by saying SETENV 0 0 and SETENV 1 0.

```
TO MUSIC
  ASSIGNKEYS [[Z C3] [X D3] [C E3] [V F3] [B G3] [N A4] [M B4]
    [, C4] [ . D4] [/ E4]]
  ASSIGNKEYS [[S C#3] [D D#3] [G F#3] [H G#3] [J A#4]
    [L C#4] [ ; D#4]]
  ASSIGNKEYS [[Q C4] [W D4] [E E4] [R F4] [T G4] [Y A5] [U B5]
    [I C5] [O D5] [P E5]]
  ASSIGNKEYS [[2 C#4] [3 D#4] [5 F#4] [6 G#4] [7 A#5]
    [9 C#5] [0 D#5]]
  MAKE "DURATION 20
  TEMPO :DURATION
  PR [READY]
  KEYBOARD 0
END
```

```
TO ASSIGNKEYS :KEYNOTEPAIRS
  IF EMPTY? :KEYNOTEPAIRS [STOP]
  MAKE FIRST FIRST :KEYNOTEPAIRS LAST FIRST :KEYNOTEPAIRS
  ASSIGNKEYS BF :KEYNOTEPAIRS
END
```

```
TO TEMPO :N
  SETENV 0 :N / 10
  SETENV 1 :N / 10
END
```

```

TO KEYBOARD :VOICE
MAKE "TEMP RC
IF NAMEP :TEMP [NOTE :VOICE THING :TEMP]
KEYBOARD 1 - :VOICE
END

```

```

TO NOTE :VOICE :NOTE
TOOT :VOICE THING :NOTE 15 :DURATION
END

```

---

PROGRAM LISTING

---

```

TO NAMENOTES
MAKE "R 15000
NAMEOCTAVES [A A# B C C# D D# E F F# G ▶
  G#] 1 55
END

```

```

TO NAMEOCTAVES :NAMES :OCTAVE ▶
  :STARTFREQ
IF :OCTAVE > 8 [STOP]
NAMEOCTAVE :NAMES :OCTAVE :STARTFREQ
NAMEOCTAVES :NAMES :OCTAVE + 1 ▶
  :STARTFREQ * 2
END

```

```

TO NAMEOCTAVE :NAMES :OCTAVE :FREQ
IF EMPTY :NAMES [STOP]
MAKE ( WORD FIRST :NAMES :OCTAVE ) ▶
  :FREQ
NAMEOCTAVE BF :NAMES :OCTAVE :FREQ * ▶
  1.0595631
END

```

```

TO PLAYTUNE :LIST
IF EMPTY :LIST [STOP]
TOOT 0 THING FIRST :LIST 15 15
PLAYTUNE BF :LIST
END

```

```

TO TURTLESONG :LIST
SETUP.TURTLE
SONG :LIST
END

```

```

TO SETUP.TURTLE
CS
TELL 0
PU LT 90 FD 150 RT 90 BK 60 PD
ST
SETPN 0
SETPC 0 40
END

```

```

TO SONG :LIST
IF EMPTY :LIST [STOP]
MAKE "NOTE FIRST :LIST
IF :NOTE = "R [TOOT 0 THING :NOTE 15 ▶
  40] [JUMP THING :NOTE]
PU RT 90 FD 20 LT 90 PD
SONG BF :LIST
END

```

```

TO JUMP :FREQ
MAKE "INT (:FREQ - 100) / 3
FD :INT
TOOT 0 :FREQ 15 40
WAIT 15
RT 180
FD :INT
RT 180
END

```

```

TO MUSIC
ASSIGNKEYS [[Z C3] [X D3] [C E3] [V ▶
  F3] [B G3] [N A4] [M B4] [, C4] ▶
  [, D4] [/ E4]]
ASSIGNKEYS [[S C#3] [D D#3] [G F#3] [H ▶
  G#3] [J A#4] [L C#4] [; D#4]]
ASSIGNKEYS [[Q C4] [W D4] [E E4] [R ▶
  F4] [T G4] [Y A5] [U B5] [I C5] ▶
  [O D5] [P E5]]
ASSIGNKEYS [[2 C#4] [3 D#4] [5 F#4] [6 ▶
  G#4] [7 A#5] [9 C#5] [0 D#5]]
MAKE "DURATION 20
TEMPO :DURATION
PR [READY]
KEYBOARD 0
END

```

```

TO ASSIGNKEYS :KEYNOTEPAIRS
IF EMPTY :KEYNOTEPAIRS [STOP]
MAKE FIRST FIRST :KEYNOTEPAIRS LAST ▶
  FIRST :KEYNOTEPAIRS
ASSIGNKEYS BF :KEYNOTEPAIRS
END

```

# NAMING NOTES

257

```

TO TEMPO :N
SETENV 0 :N / 10
SETENV 1 :N / 10
END

```

```

TO KEYBOARD :VOICE
MAKE "TEMP RC
IF NAMEP :TEMP [NOTE :VOICE THING ►
:TEMP]
KEYBOARD 1 - :VOICE
END

```

```

TO NOTE :VOICE :NOTE
TOOT :VOICE THING :NOTE 15 :DURATION
END

```

```

MAKE "SCALE [C2 D2 E2 F2 G2 A3 B3 C3]
MAKE "SCALE2 [C3 D3 E3 F3 G3 A4 B4 C4]
MAKE "TWINKLE [C2 C2 G2 G2 A3 A3 G2 R ►
F2 F2 E2 E2 D2 D2 C2 R]
MAKE "FOLK [D4 C#4 D4 D4 D3 D3 F#3 F#3 ►
A4 A4 B4 A4 B4 C#4 D4 D4]

```