Chapter 3: Sound and Music.

Now that we have color and graphics, let's add sound and make some music. Basic concepts of the physics of sound, numeric variables, and musical notation will be introduced. You will be able to translate a tune into frequencies and durations to have the computer synthesize a voice.

Sound Basics – Things you need to know about sound:

Sound is created by vibrating air striking your ear-drum. These vibrations are known as sound waves. When the air is vibrating quickly you will hear a high note and when the air is vibrating slowly you will hear a low note. The rate of the vibration is called frequency.

Illustration 9: Sound Waves

Frequency is measured in a unit called hertz (Hz). It represents how many cycles (ups and downs) a wave vibrates through in a second. A normal
person can hear very low sounds at 20 Hz and very high sounds at 20,000 Hz. BASIC-256 can produce tones in the range of 50Hz to 7000Hz.

Another property of a sound is its length. Computers are very fast and can measure times accurately to a millisecond (ms). A millisecond (ms) is 1/1000 (one thousandths) of a second.

Let's make some sounds.

```
1 # c3_sounds.kbs
2 sound 233, 1000
3 sound 466, 500
4 sound 233, 1000
```

*Program 17: Play Three Individual Notes*

You may have heard a clicking noise in your speakers between the notes played in the last example. This is caused by the computer creating the sound and needing to stop and think a millisecond or so. The `sound` statement also can be written using a list of frequencies and durations to smooth out the transition from one note to another.

```
1 # c3_soundslist.kbs
2 sound {233, 1000, 466, 500, 233, 1000}
```

*Program 18: List of Sounds*

This second sound program plays the same three tones for the same duration but the computer creates and plays all of the sounds at once, making them smoother.
sound frequency, duration
sound {frequency1, duration1, frequency2, duration2 ...}
sound numeric_array

The basic sound statement takes two arguments; (1) the frequency of the sound in Hz (cycles per second) and (2) the length of the tone in milliseconds (ms). The second form of the sound statement uses curly braces and can specify several tones and durations in a list. The third form of the sound statement uses an array containing frequencies and durations. Arrays are covered in Chapter 11.

How do we get BASIC-256 to play a tune? The first thing we need to do is to convert the notes on a music staff to frequencies. Illustration 9 shows two octaves of music notes, their names, and the approximate frequency the note makes. In music you will also find a special mark called the rest. The rest means not to play anything for a certain duration. If you are using a list of sounds you can insert a rest by specifying a frequency of zero (0) and the needed duration for the silence.
Take a little piece of music and then look up the frequency values for each of the notes. Why don't we have the computer play "Charge!". The music is in Illustration 11. You might notice that the high G in the music is not on the musical notes; if a note is not on the chart you can double (to make higher) or half (to make lower) the same note from one octave away.

Illustration 10: Musical Notes

Illustration 11: Charge!
Now that we have the frequencies we need the duration for each of the notes. Table 2 shows most of the common note and rest symbols, how long they are when compared to each other, and a few typical durations.

Duration in milliseconds (ms) can be calculated if you know the speed if the music in beats per minute (BPM) using Formula 1.

\[ \text{Note Duration} = \frac{1000 \times 60}{\text{Beats Per Minute}} \times \text{Relative Length} \]

*Formula 1: Calculating Note Duration*

<table>
<thead>
<tr>
<th>Note Name</th>
<th>Symbols for Note - Rest</th>
<th>Length (ms)</th>
<th>At 100 BPM</th>
<th>At 120 BPM</th>
<th>At 140 BPM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dotted Whole</td>
<td></td>
<td>6.000</td>
<td>3600 ms</td>
<td>3000 ms</td>
<td>2571 ms</td>
</tr>
<tr>
<td>Whole</td>
<td></td>
<td>4.000</td>
<td>2400 ms</td>
<td>2000 ms</td>
<td>1714 ms</td>
</tr>
<tr>
<td>Dotted Half</td>
<td></td>
<td>3.000</td>
<td>1800 ms</td>
<td>1500 ms</td>
<td>1285 ms</td>
</tr>
<tr>
<td>Half</td>
<td></td>
<td>2.000</td>
<td>1200 ms</td>
<td>1000 ms</td>
<td>857 ms</td>
</tr>
<tr>
<td>Dotted Quarter</td>
<td></td>
<td>1.500</td>
<td>900 ms</td>
<td>750 ms</td>
<td>642 ms</td>
</tr>
<tr>
<td>Quarter</td>
<td></td>
<td>1.000</td>
<td>600 ms</td>
<td>500 ms</td>
<td>428 ms</td>
</tr>
<tr>
<td>Dotted Eighth</td>
<td></td>
<td>0.750</td>
<td>450 ms</td>
<td>375 ms</td>
<td>321 ms</td>
</tr>
<tr>
<td>Eighth</td>
<td></td>
<td>0.500</td>
<td>300 ms</td>
<td>250 ms</td>
<td>214 ms</td>
</tr>
<tr>
<td>Dotted Sixteenth</td>
<td></td>
<td>0.375</td>
<td>225 ms</td>
<td>187 ms</td>
<td>160 ms</td>
</tr>
<tr>
<td>Sixteenth</td>
<td></td>
<td>0.250</td>
<td>150 ms</td>
<td>125 ms</td>
<td>107 ms</td>
</tr>
</tbody>
</table>

*Table 2: Musical Notes and Typical Durations*

© 2014 James M. Reneau (CC BY-NC-SA 3.0 US)
Now with the formula and table to calculate note durations, we can write the program to play "Charge!".

```
1 # c3_charge.kbs
2 # play charge
3
4 sound {392, 375, 523, 375, 659, 375, 784, 250, 659, 250, 784, 250}
5 say "Charge!"
```

Program 19: Charge!

**Numeric Variables:**

Computers are really good at remembering things, where we humans sometimes have trouble. The BASIC language allows us to give names to places in the computer's memory and then store information in them. These places are called variables.

There are four types of variables: numeric variables, string variables, numeric array variables, and string array variables. You will learn how to use numeric variables in this chapter and the others in later chapters.
**Numeric variable**

A numeric variable allows you to assign a name to a block of storage in the computer's short-term memory. You may store and retrieve numeric (whole or decimal) values from these variables in your program.

A numeric variable name must begin with a letter; may contain letters and numbers; and are case sensitive. You may not use words reserved by the BASIC-256 language when naming your variables (see Appendix I).

Examples of valid variable names include: a, b6, reader, x, and zoo.

<table>
<thead>
<tr>
<th>Variable names are case sensitive. This means that an upper case variable and a lowercase variable with the same letters do not represent the same location in the computer's memory.</th>
</tr>
</thead>
</table>

Program 20 is an example of a program using numeric variables.

```kbs
# c3_numericvariables.kbs
# use numeric variables

let numerator = 30
let denominator = 5
let result = numerator / denominator
print result
```

*Program 20: Simple Numeric Variables*
The program above uses three variables. On line two it stores the value 30 into the location named "numerator". Line three stores the value 5 in the variable "denominator". Line four takes the value from "numerator" divides it by the value in the "denominator" variable and stores the value in the variable named "result".

```plaintext
let variable = expression
variable = expression
```

The `let` statement will calculate an expression (if necessary) and saves the value into a variable. We call this process assignment or assigning a variable.

The variable on the left hand side of the equal sign will take on the value of the variable, number, function, or mathematical expression on the right hand side of the equal sign.

The actual `let` statement is optional. You can just assign a variable using the equal sign.

```plaintext
let a = 7
let b = a / 2 + .7
print a + b
```

*Program 21: Simple Variable Assignment*
The statements Program 21 will create two storage locations in memory and store the value or the result of the calculation in them. Line three of the program will add the values together and print the value 11.2. You may use a numeric variable anywhere you need a number and the value in the variable will be pulled from memory.

Variables are called variables because they can be changed as a program runs. Look at the example in Program 22 (below) In line 1 the variable z is assigned the value 99. In line 2 the expression \( z - 1 \) is calculated and the result is stored back in \( z \). In the last line the value of \( z \) is printed, Can you guess what that will be?

1. \( \text{z = 99} \)
2. \( \text{z = z - 1} \)
3. \( \text{print z} \)

*Program 22: Variable Re-assignment*

Variables and their associated values persist, once they are created, for the remainder of the time a program is running. Once a program stops (either...
completes or errors) the variables values are emptied and the memory is
returned to the computer's operating system to be assigned for future tasks.

Now that we have learned a bit more about variables we could re-write the
"Charge!" program using variables and the formula to calculate note
durations (Formula 1).

```
# c3_charge2.kbs
# play charge - use variables

beats = 120
dottedeighth = 1000 * 60 / beats * .75
eighth = 1000 * 60 / beats * .5

sound {392, dottedeighth, 523, dottedeighth, 659, 
      dottedeighth, 784, eighth, 659, eighth, 784, eighth}

say "Charge!"
```

**Program 23: Charge! with Variables**

**Variable Assignment Shortcuts:**

Another thing you will learn about computer programming is that there are
often more than one way do do a task. BASIC-256 and most computer
programming languages allow for a shortcut form of addition and subtraction
when working with a variable. In the programs of future chapters you will see these shortcuts.

© 2014 James M. Reneau (CC BY-NC-SA 3.0 US)
Table 3: Shortcut Variable Assignments

<table>
<thead>
<tr>
<th>Shortcut Assignment</th>
<th>Longhand Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable+=expression</td>
<td>variable = variable + expression</td>
</tr>
<tr>
<td>a+=9</td>
<td>a = a + 9</td>
</tr>
<tr>
<td>variable -= expression</td>
<td>variable = variable - expression</td>
</tr>
<tr>
<td>b -= a+2</td>
<td>b = b - (a + 2)</td>
</tr>
<tr>
<td>variable++</td>
<td>variable = variable + 1</td>
</tr>
<tr>
<td>foo++</td>
<td>foo = foo + 1</td>
</tr>
<tr>
<td>Variable--</td>
<td>Variable = variable - 1</td>
</tr>
<tr>
<td>bar--</td>
<td>bar = bar - 1</td>
</tr>
</tbody>
</table>

For this chapter's big program let's take a piece of music by J.S. Bach and write a program to play it.

The musical score is a part of J.S. Bach's Little Fuge in G.

Illustration 12: First Four Measures of J.S. Bach's Little Fuge in G

```
# c3_littlefuge.kbs
# Music by J.S.Bach - XVIII Fuge in G moll.
```
Chapter 3: Sound and Music.

```
4  tempo = 100  # beats per minute
5  milimin = 1000 * 60  # miliseconds in a minute
6  q = milimin / tempo  # quarter note is a beat
7  h = q * 2  # half note (2 quarters)
8  e = q / 2  # eight note (1/2 quarter)
9  s = q / 4  # sixteenth note (1/4 quarter)
10 de = e + s  # dotted eight - eight + 16th
11 dq = q + e  # doted quarter - quarter + eight
13 sound{392, q, 587, q, 466, dq, 440, e, 392, e, 466, e, 440, e, 392, e, 370, e, 440, e, 294, q, 392, e, 294, e, 440, e, 294, e, 466, e, 440, s, 392, s, 440, e, 294, e, 392, e, 294, s, 392, s, 440, e, 294, s, 440, s, 466, e, 440, s, 392, s, 440, s, 294, s}
```

Program 24: Big Program - Little Fuge in G

© 2014 James M. Reneau (CC BY-NC-SA 3.0 US)
Exercises:

3.1. Write a program using a single sound statement to play “Shave and a Hair Cut”. Remember you must include the quarter rests in the second measure in your sound with a frequency of zero and the duration of a quarter note.
3.2. Type the sound statement below and insert the variable assignments before it to play “Row Row Row your Boat”. The variables c, d, e, f, g, and cc should contain the frequency of the notes of the tune. The variable n4 should contain the length in milliseconds of a quarter note; n2 twice n4, and n8 one half of n4.

```
sound {c,n4+n8, c,n4+n8, c,n4, d,n8, e,n4+n8, e,n4, d,n8, e,n4, f,n8, g,n2+n4, cc,n8, cc,n8, cc,n8, g,n8, g,n8, g,n8, e,n8, e,n8, e,n8, e,n8, c,n8, c,n8, c,n8, g,n4, f,n8, d,n4, e,n8, c,n2+n4}
```

3.3. Create a program with two variables 'a' and 'b' that you will assign to two numbers. Print the sum of a and b, the difference of a and b, the difference of b and a, the product of a and b, the quotient of a divided by b, and the quotient of b divided by a. Run the program with several different values of a and b. What happens when a or b are set to the value of zero?