Sound Part 2: Exploring Programming

Can you really learn to program by investigation? If you are given lots of examples and good references, the answer is a qualified "yes." Computer Science is a lab science, and the good news is that you don't have to kill your object of study---experiments with micro-processors are non-violent.

Blink (again)

Think about the program *Blink* from our last exercise. Did it ever stop running? If you are new to programming, you may not find this curious, but a computer (or micro-processor) typically executes each command that you give it, in the order that you give them, and then waits until you give it more commands. Why did the LED continue to blink indefinitely? If we don't worry about the implementational details, the reason is simple: Commands inside the procedure *setup()* run once at the start of the program. After setup() has completed, the commands inside the procedure *loop()* run repeatedly.

```
void setup()        // run once, when the program starts
{
   pinMode(ledPin, OUTPUT);
}

void loop()         // run over and over again
{
   digitalWrite(ledPin, HIGH);
   delay(1000);
   digitalWrite(ledPin, LOW);
   delay(1000);
}
```

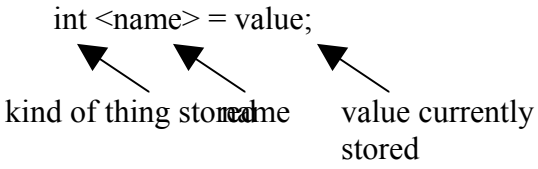
Statements between these 2 brackets belong to setup()

Statements between these 2 brackets belong to loop()

There are a few other concepts that we need before we can play a song on our micro-processor: variables, arrays, if statements, loops, and procedures. If you are already familiar with these concepts, you can proceed quickly through the sections that follow until you get to **Making Music**. Be sure to answer the questions associated with each section.

Variables

Remember the first statement in *Blink*: `int ledPin = 13;`? In that statement, `ledPin` is *declared* as a *variable*. Variables store things like numbers, or letters, and variables have names like *ledPin*. The statement `int ledPin = 13;` means that the variable named ledPin can store integers (i.e., int) and it now stores the number 13.



Later on, when we write: `pinMode(ledPin, OUTPUT);` we think of ledPin as having the value that we assigned earlier, 13. So that `pinMode(ledPin, OUTPUT);` means exactly the same as `pinMode(13, OUTPUT);`. For more information on variables, checkout this link.

As a test of your knowledge of variables, answer the following question; ask your instructor if you need help. In the Arduino programming language, `int` stands for interger, which is a whole---a number without a fractional or decimal constituent. The word `double` stands for a real number---a number that can have a fractional component. How would you declare a variable that can store a real number and assign the value 3.2 to that variable?

Arrays

Arrays are just groups of variables. Grouping variables makes them easier to declare. For example, we could declare 6 variables to store numbers without using arrays as follows:

```
int number1;
int number2;
int number3;
int number4;
int number5;
int number6;
```

Or we could declare them as an array with one statement as follows:

```
int numbers[6];
```

The entire collection, i.e., the array, is called *numbers* and its size is 6 (the number in square brackets following the name).

To store the number 10 in the first variable, without arrays, I would write:

```
number1 = 10;
```

With arrays, I would write:

```
numbers[0] = 10;
```

The name of each variable in an array is the array name followed by its index position in square brackets; we count index positions starting from 0. What would be the name of the 2nd variable in the array?

While it is possible to assign a value to an individual variable (and change that value at any time), it can be convenient to assign values to all variables in the array at the time the array is declared. This can only be done when the array is declared, and the statement looks something like this:

int numbers = {1, 2, 3, 4, 5, 6};

The values enclosed in {}s are assigned to the variables in the array sequentially from left to right, such that numbers[0]=1, numbers[1]=2, and so on. The number of values specified determines the number of variables in the array.

For more information on arrays, follow this link.

Again, as a test of your knowledge, answer the following question and ask your instructor if you need help. How would you declare an array of 10 variables that can store real numbers and assign the value 1.1 to each variable in the array?

If Statements

The *if statement* allows you to make choices while the program is running. You can make something happen or not depending on whether a given condition is true at the moment the program is executing. It looks like this:

```
if (someCondition) {
   // do stuff if the condition is true
}
```

There is a common variation called if-else that looks like this:

```
if (someCondition) {
   // do stuff if the condition is true
} else {
   // do stuff if the condition is false
}
```

There's also the else-if, where you can check a second condition if the first is false:

```
if (someCondition) {
   // do stuff if the condition is true
} else if (anotherCondition) {
   // do stuff only if the first condition is false
   // and the second condition is true
}
```

To see the *if statement* in action, we can run another sample program. Checkout **File > Examples > Digital > Button**. Before you can run the program, you need to connect a switch (we'll use a *pushbutton* switch) to pin 2 (or any digital pin) of the Arduino. Create the circuit as shown and explained on the following link. You may add an external LED at pin 13 as you did in the previous exercise or just used the built-in LED. Once your circuit is complete, run **Button** and be prepared to explain what it does and exactly how it does it.

Loops

Remember that the statements (i.e., commands) in loop() run repeatedly---you saw that again in the example above. The name *loop* is indicative of its behavior. You can create loops in your program whenever you need them. We will learn to do this using a ***for loop***.

A *for* statement (i.e., *for* loop) repeats the statements enclosed in curly braces following the header of the loop; it looks like this:

```
for (initialization; condition; increment)
{
//statement(s) to be repeated;
}
```

As indicated above, there are three parts to the **for** loop header. Here's how it works:

1. The **initialization** is executed once at the start of the loop.
2. The **condition** is tested. If it's true, the statements enclosed in {}s are executed.
3. The **increment** is executed and step 2 above is repeated.
4. When the **condition** becomes false, the loop ends.

Typically for loops are used to repeat commands a specific number of times, and the **`initialization`**, **`condition`**, and **`increment`** are used to count the number of times that the commands repeat. Run the following examples and make sure that you understand them. These examples make use of the Serial.print() and Serial.println() commands to print data to the serial port. You can see what's printed to the serial port by selecting the ***Serial Monitor*** on the Arduino IDE tool bar. Be sure to open (i.e., select) the ***Serial Monitor*** when running each of the examples below.

Example 1

```
// Prints numbers to the serial port
void setup()
{
  Serial.begin(9600);               // open serial port at 9600 bps:
}

void loop()
{
  for (int count = 1; count < 10; count++)
   {
       Serial.println(count, DEC);  // print number to the serial port
```

```
    }
}
```

What does the statement count++ do? Modify the program above to print the numbers 0 through 10---what did you change?

Example 2

```
// Prints numbers in reverse order to the serial port
void setup()
{
  Serial.begin(9600);               // open serial port at 9600 bps:
}

void loop()
{
  for (int count = 20; count > 0; count--)
   {
       Serial.println(count, DEC);  // print number to the serial port
   }
}
```

Modify the program above to print the numbers 30 through 0---what did you change?

Example 3

```
// Prints even numbers to the serial port
void setup()
{
  Serial.begin(9600);               // open serial port at 9600 bps:
}

void loop()
{
  for (int count = 0; count < 21; count=count+2)
   {
       Serial.println(count, DEC);  // print number to the serial port
   }
}
```

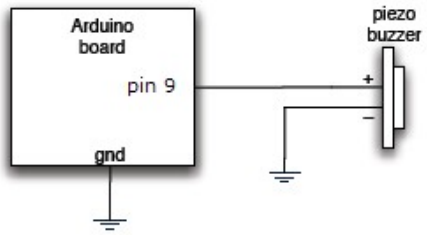
Modify the program above to print the odd numbers between 0 and 20---what did you change?

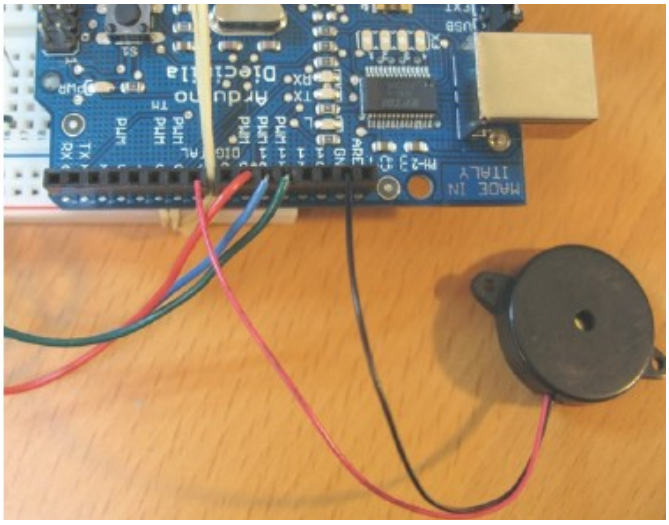
Procedures

This is the last programming "concept" we need to understand before we can "make music" using our new microprocessor brain. Actually, you have already used *procedures* in the programs above; both setup() and loop() are procedures. And, fortunately for us, there is an excellent tutorial on procedures provided by Limor Fried and from http://www.ladyada.net/learn/arduino." Thanks! Read and work Lesson 2 from ladyada to learn about procedures. The tutorial covers a few concepts that we have already discussed, but a little repetition never hurts, hurts, hurts…

Making Music

Ok, let's make some music using our Arduino. We will again use an example program from the Arduino IDE. Begin by attaching a piezo speaker to the Arduino per the schematic below. You will probably have to use your breadboard to make the connection to the speaker because most of our piezo speakers are configured differently than the one pictured below.





Once you have connected your speaker, select **File > Examples > Digital > Melody** to load the program. Run the program and then study it. Try to understand how it works through analysis and experimentation. As part of your experimentation and to verify your understanding of the program, modify Melody to play a new song.

We will end this exercise with a demonstration of your new song and a discussion of the Melody program.