Sketching in Code

In the last chapter, we managed to get our first sketch up and running with the famous Blink sketch, otherwise known as the “Hello world” example. This is often a good place to start because it has a long-standing tradition as the first program to write when learning a new language. In the case of programming hardware, we generally use the LED to blink hello. While that is a good beginning, we want to quickly get to some of the more interesting aspects of Arduino C, beginning with the focus of this chapter: the fundamental mechanics and control structures of sketching Arduino code. Our first project, RGB Blink, will explore the basic structure, syntax, and functions necessary for controlling the output of a three-color LED, taking our blinking LED to the next level.

We’ll start with a primer on what sketching means and how we can put the philosophy of sketching in code to use. As we will do throughout this book, we will then jump right into our project example and once you have this project wired up and the sketch uploaded to your Arduino board, we’ll back up and further explain how the sketch works and discuss most of the elements used in the project. In this way, when we look at each of the individual concepts in the chapter, you’ll already know how they were put to use. In this chapter, we’ll have a look at the basic anatomy of our project sketch and discuss how and why Arduino C does what it does. Don’t worry if it doesn’t make a tremendous amount of sense in the beginning, just work through it and it should get clearer as you go along.

What’s needed for this chapter:

- Arduino Uno
- 5mm RGB LED with common cathode
- 1x 330 ohm and 2x 220 ohm ¼ watt resistors or similar
- Hookup wires
- Solderless breadboard

What is Sketching in Code?

The idea of sketching in code is a way of thinking about writing code as a simple intuitive process, just like drawing in a sketchbook. In this way, an Arduino program is called a sketch and is saved in a folder called a sketchbook. Sketching means we can get our hands dirty and quickly try out a new idea. It is a skill available to all of us, not just artists and designers, and neither is it limited to pens, paper, pencils, or napkins.
So often, an idea in one of my classes begins with the simple words, “Wouldn’t it be cool if …?” Quickly sketching out these ideas serves as a way to conceptualize this moment of inspiration. Writing code can be just like this; it is after all a creative process used to solve specific problems. Sketching implies a sense of directness in the application of materials, like a pencil to a piece of paper. The Arduino development environment takes this same approach to making code as simple and direct as is possible. It is also why we will begin with simple sketches that quickly get you making something right away rather than bogging down in page after page of complex theory or algorithms.

When sketching in code, it’s okay to write bad code. You won’t hurt anything. As long as you get in there and start somewhere, as doing anything is always better than doing nothing. Making mistakes and learning what does work and what doesn’t, is an important part of learning anything new. In our sketchbook, we will start with simple source code, the basic instructions that tell the Arduino what it should do, along with basic hardware, starting small and working our way through to more involved examples.

You should freely experiment with every example, changing values, and piecing things together in unexpected ways to see what happens. Don’t be discouraged if it doesn’t work the first time; stick with it and it will get clearer with time. Testing and iterations are important parts of successfully writing code. So, change existing code or write some new code to establish a basic framework and then verify that your changes compile correctly. Work in incremental steps one addition at a time as you add to this basic framework for your code so that when you finally have something substantial, you can load it on to the interface board to see what happens. If something didn’t quite work as expected, all you need to do is undo that last incremental step. In this way it’s a good idea when you make changes, to only change one thing at a time before verifying that it still works before moving on to the next change.

**Project 1: RGB Blink**

To really get started, we are going to use a nifty little component called a red-green-blue light emitting diode, or more simply an RGB LED. The RGB LED works off a similar principle as televisions and computer monitors. By using the same three colors of light that we are receptive to in our vision—specifically red, green, and blue—we can reproduce a vast array of colors through an additive color-mixing process. This form of color mixing should sound familiar from secondary school science, and is shown in the RGB color wheel in Figure 2-1. While it’s a little harder to make out in a black-and-white book like this one, if you use your imagination you’ll see that by combining two primary colors we end up with a secondary color. For example, add the color red to blue and we get magenta or if we add red and green we will get yellow. If we add all three primary colors together we will end up with white light.