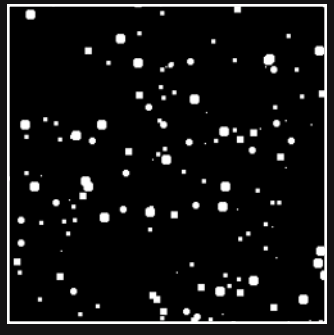
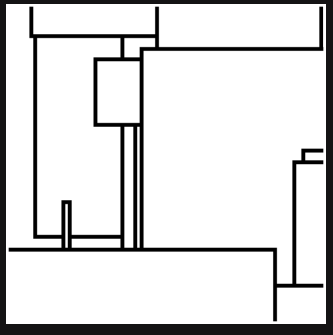
3 CODE GRAMMAR 101





This is where the techie stuff really begins! Hopefully, in the last two chapters you've gotten some context and maybe even inspiration and are ready to dive right in. This chapter looks at the nuts and bolts of programming in general. I make the assumption that you have no experience with coding at all, so for those of you coming to Processing from another language, you may want to skim this chapter. In Chapter 4, I cover the basics of graphics coding, and in Chapter 5, I discuss the details of the Processing environment.

For new coders, my recommendation is to read this chapter at least once, but to not worry about retaining all this info before continuing on. The material will sink in as you begin programming, and you can refer back to this chapter from time to time as a reference. Most of what I cover in this chapter is general programming theory and basic syntax and semantics using Processing and to some degree Java. By syntax, I mean the way language is put together to form actual statements, functions, classes, and so on. Semantics refers to the actual meaning of the code we write. Most of what I cover here, especially the theoretical stuff, is applicable to other languages besides Processing, but of course each language has its own syntax, so the actual code will look different. Without further ado, I present: Coding . . . really, really simplified.

Structure and abstraction

There are a couple of ways to think about structure in coding. On the simplest level, you can think of structure in terms of the syntax you use to write a single line of code. For example, `ball.x = width-ball.width;`. On the other end of the spectrum, structure can involve applying complex rules and protocols for integrating large, convoluted software systems. Fortunately for your needs, most of the structural issues will be pretty straightforward. You will use basic syntactic structure to order a program, not unlike how you would structure any written document, using punctuation, sentence structures, paragraphs, and so on. In coding, there are some other more abstract notions of structure that I will discuss as well.

Our brains like some sense of order—although the range of chaotic tolerance among different people's brains seems pretty wide (as my office usually attests to). When a program gets overly complicated, it becomes hard to keep track of what's going on. This often becomes an especially vexing problem when you take a break from a project and then pick it up some time later. There are a number of structures or abstractions commonly used in coding to help order the process. The two major programming approaches I will cover are **procedural programming** and **object-oriented programming (OOP)**. I introduced and defined both briefly in Chapter 2. Procedural programming relies on reusable blocks of code that work like processing machines, which you call when you need them. These reusable blocks of code are referred to as **functions**. The second approach, OOP, is a far more ambitious and abstract approach that models a programming problem using concepts from the real world. For example, if you were writing an object-oriented program to generate a virtual garden, you would create code structures, called **classes**, to organize your program. The classes might each describe a garden concept from the real world: leaf, vine, flower, sunlight, water, and so forth.