Development 1: Sketching, Techniques

This unit discusses the idea of sketching code and the iterative development process.

There are similarities between learning a programming language and learning a new spoken language. First, one learns basic elements of a spoken language—such as simple words and grammar rules—and mimics short phrases. Learning to communicate ideas and express emotions within the language takes more time. Similarly, the first step in computer programming is to understand the basic elements such as comments, variables, and functions. The next step is to learn to read and modify simple example programs. Later, one begins to write programs from scratch. The most interesting and difficult stage of learning to program comes later, as one gains the ability to put the language elements together to express ideas about form, motion, and behavior. Like learning a foreign language, becoming fluent in a programming language can take years.

Sketching software

Sketching ranges from informal exploration to focused refinement. It is used to create many variations within a short period of time, or to develop a specific idea. Sketching forces the definition of vague ideas by making them physical. Sketches are powerful communication tools—they can get ideas out of one's head and into a format that can be better understood by others.

It is important to work out ideas on paper before investing time in writing code. Paper and pencil allow for fast iteration in the early stages of a project. The most important aspect of programming is figuring out what will be created and how it will function, so working out these ideas away from a computer keeps the focus on an idea, rather than its implementation.

A good paper sketch for software will include a series of images that demonstrate how the narrative structure of the piece works, much like an animator's storyboard. In addition to images that will appear on screen, sketches often contain diagrams of the program's flow, data elements, and notation for showing how forms will move and interact. Programs can also be planned using combinations of image mock-ups, formal schematics, and text descriptions.

After refined ideas reach a point where working on paper is no longer useful, code can continue the development. The first step in creating code is a continuation of the sketching process. Write short pieces of code independently before worrying about the structure of the larger program. Writing small, focused programs makes a developer better at writing code when it matters most: when working on a more refined implementation.

Processing programs are called sketches to emphasize this method of working. The Processing sketchbook is a way of storing and organizing programs. Code sketches can be reviewed and developed incrementally like drawings in a paper sketchbook. Ideas that flash by while walking or just after waking up can be quickly made into code and stored for future use. The Processing environment encourages this type of writing because one need only press the New button in the toolbar to start a new sketch.

Some programming languages encourage a sketching approach, and others make it difficult. Scripting languages, such as Perl or Python, are designed to encourage rapid development at the expense of running speed and control. Processing is not a scripting language, but is designed to "feel" like a scripting language while providing the same capabilities as a more complete language like Java. This topic is discussed in more depth in Appendix F (p. 679).

Programming techniques

There are as many ways to write programs as there are people who write software. Some common strategies for creating programs include modification, augmentation, collage, and writing code from scratch. People learning to code often expect most programs to be written from scratch, but that's rarely the case, particularly for the style of work built with Processing. Learning to read and modify code helps programmers increase their skills. Even advanced programmers work from others' examples when learning new techniques.

Modification
Changing the values of variables in existing programs is a good way to explore code. Programs can be modified by trial and error or more deliberately. One way to start understanding a program is to change slightly the value of one variable and then run the code to see the result. If there is no obvious difference, change the value again. Making the correlation between a variable and a change in the way a program runs is a good first step to understanding how it works. Disabling lines of code by placing them inside a `/*` and `*/` comment block (called "commenting out") is another way to decrypt a program. A little understanding of the way a program is structured can facilitate logical guesses about what different lines of code are doing. These modification techniques aid in learning new skills or parsing an example. Making small changes to an existing program encourages exploration and getting a feel for the code.

Augmentation
Augmentation uses existing code as a base for further exploration. It is similar to but more ambitious than modification. Generic program examples can serve as a foundation for longer, more specific programs. An example that draws a Bézier curve can be used as a base for drawing a series of curves (as shown in Shape 2, p. 69). A program that displays a photograph can form the basis of a photo montage application. Sample programs

provide a concise reminder of syntax. The spartan programs presented in this book provide a broad base for making enhancements.

Collage
The collage technique involves cutting and pasting elements of different programs together to create a new program. It's analogous to creating music by sampling or making a visual collage from newspaper and magazine clippings. In order to avoid errors, combine code carefully by copying a few lines of code at a time and running the program to make sure it's always working. Copying large portions of code can introduce a number of simultaneous errors. Mindlessly copying and pasting code can create "Frankenstein" code that's difficult to debug. As an individual's knowledge of programming increases, using this technique becomes easier, and common problems can be avoided, such as adding multiple copies of the same method, such as `draw()` or `setup()`, to the code.

Coding from scratch
Rarely do programmers write a complete program entirely from scratch. At the minimum, most people start with a template. A template is an outline with code infrastructure common to many programs. Sometimes it's not possible to find a related example or appropriate template and it's necessary to start with a blank page. In this case, comments are a great way to start building a program. Comments can be used to build an outline of the program's intention, logic, and flow. After this structure has been defined, lines of code can be slowly added and run in an attempt to realize those decisions.

Regardless of the technique used for programming, writing a few lines of code or making only a few changes at a time is a good tactic. Entering many lines of untested code before running the program increases the potential for multiple errors. The more errors in a program, the more difficult they become to find. Running a growing program piece by piece reduces the chance for multiple errors. As your comfort with code and your skills increase, it becomes possible to make more modifications between tests.