Data 1: Variables

This unit introduces different types of data and explains how to create variables and assign them values.

Syntax introduced:
`int, float, boolean, true, false, = (assign), width, height`

What is data? Data often consists of measurements of physical characteristics. For example, Casey's California driver's license states his sex is M, his hair is BRN, and his eyes are HZL. The values M, BRN, and HZL are items of data associated with Casey. Data can be the population of a country, the average annual rainfall in Los Angeles, or your current heart rate. In software, data is stored as numbers and characters. Examples of digital data include a photograph of a friend stored on your hard drive, a song downloaded from the Internet, and a news article loaded through a web browser. Less obvious is the data continually created and exchanged between computers and other devices. For example, computers are continually receiving data from the mouse and keyboard. When writing a program, you might create a data element to save the location of a shape, to store a color for later use, or to continuously measure changes in cursor position.

Data types

Processing can store and modify many different kinds of data, including numbers, letters, words, colors, images, fonts, and boolean values (`true`, `false`). The computer stores each in a different way, so it has to know which type of data is being used to know how to manage it. For example, storing a word takes more room than storing one letter; therefore, storing the word *Cincinnati* requires more space than storing the letter *C*. If space has been allocated for only one letter, trying to store a word in the same space will cause an error. Every data element is represented as sequences of bits (0s and 1s) in the computer's memory (more information about bits is found in Appendix D, p. 669). For example, 01000001 can be interpreted as the letter *A*, and it can also be interpreted as the number 65. It's necessary to specify the type of data so the computer knows how to correctly interpret the bits.

Numeric data is the first type of data encountered in the following sections of this book. There are two types of numeric data used in Processing: integer and floating-point. Integers are whole numbers such as 12, –120, 8, and 934. Processing represents integer data with the `int` data type. Floating-point numbers have a decimal point for creating fractions of whole numbers such as 12.8, –120.75, 8.125, and 934.82736. Processing represents floating-point data with the `float` data type. Floating-point numbers are often used to approximate analog or continuous values because they have decimal

resolution. For example, using integer values, there is only one number between 3 and 5, but floating-point numbers allow us to express myriad numbers between such as 4.0, 4.5, 4.75, 4.825, etc. Both int and float values may be positive, negative, or zero.

The simplest data element in Processing is a boolean variable. Variables of this type can have only one of two values—`true` or `false`. The name boolean refers to the mathematician George Boole (b. 1815), the inventor of Boolean algebra—the foundation for how digital computers work. A boolean variable is often used to make decisions about which lines of code are run and which are ignored.

The following table compares the capacities of the data types mentioned above with other common data types:

Name	Size	Value range
boolean	1 bit	true or false
byte	8 bits	-128 to 127
char	16 bits	0 to 65535
int	32 bits	-2,147,483,648 to 2,147,483,647
float	32 bits	3.40282347E+38 to -3.40282347E+38
color	32 bits	16,777,216 colors

Additional types of data are introduced and explained in Data 2 (p. 101), Data 3 (p. 105), Image 1 (p. 95), Typography 1 (p. 111), and Structure 4 (p. 395).

Variables

A variable is a container for storing data. Variables allow a data element to be reused many times within a program. Every variable has two parts, a name and a value. If the number 21 is stored in the variable called *age*, every time the word *age* appears in the program, it will be replaced with the value 21 when the code is run. In addition to its name and value, every variable has a data type that defines the category of data it can hold.

A variable must be declared before it is used. A variable declaration states the data type and variable name. The following lines declare variables and then assign values to the variables:

3-01

```
int x;      // Declare the variable x of type int
float y;    // Declare the variable y of type float
boolean b;  // Declare the variable b of type boolean
x = 50;     // Assign the value 50 to x
y = 12.6;   // Assign the value 12.6 to y
b = true;   // Assign the value true to b
```

As a shortcut, a variable can be declared and assigned on the same line:

```
int x = 50;                                                    3-02
float y = 12.6;
boolean b = true;
```

More than one variable can be declared in one line, and the variables can then be assigned separately:

```
float x, y, z;                                                 3-03
x = -3.9;
y = 10.1;
z = 124.23;
```

When a variable is declared, it is necessary to state the data type before its name; but after it's declared, the data type cannot be changed or restated. If the data type is included again for the same variable, the computer will interpret this as an attempt to make a new variable with the same name, and this will cause an error (an exception to this rule is made when each variable has a different scope, p. 178):

```
int x = 69;  // Assign 69 to x                                 3-04
x = 70;      // Assign 70 to x
int x = 71;  // ERROR! The data type for x is duplicated
```

The = symbol is called the assignment operator. It assigns the value from the right side of the = to the variable on its left. Values can be assigned only to variables. Trying to assign a constant to another constant produces an error:

```
// Error! The left side of an assignment must be a variable    3-05
5 = 12;
```

When working with variables of different types in the same program, be careful not to mix types in a way that causes an error. For example, it's not possible to fit a floating-point number into an integer variable:

```
// Error! It's not possible to fit a floating-point number into an int  3-06
int x = 24.8;
```

```
float f = 12.5;                                                3-07
// Error! It's not possible to fit a floating-point number into an int
int y = f;
```

Variables should have names that describe their content. This makes programs easier to read and can reduce the need for verbose commenting. It's up to the programmer to decide how she will name variables. For example, a variable storing the room temperature could logically have the following names:

```
t
temp
temperature
roomTemp
roomTemperature
```

Variables like `t` should be used minimally or not at all because they are cryptic—there's no hint as to what they contain. However, long names such as *roomTemperature* can also make code tedious to read. If we were writing a program with this variable, our preference might be to use the name *roomTemp* because it is both concise and descriptive. The name *temp* could also work, but because it's used commonly as an abbreviation for "temporary," it wouldn't be the best choice.

There are a few conventions that make it easier for other people to read your programs. Variables' names should start with a lowercase letter, and if there are multiple words in the name, the first letter of each additional word should be capitalized. There are a few absolute rules in naming variables. Variable names cannot start with numbers, and they must not be a reserved word. Examples of reserved words include `int`, `if`, `true`, and `null`. A complete list is found in Appendix B (p. 663). To avoid confusion, variables should not have the same names as elements of the Processing language such as `line` and `ellipse`. The complete Processing language is listed in the reference included with the software.

Another important consideration related to variables is the scope (p. 178). The scope of a variable defines where it can be used relative to where it's created.

Processing variables

The Processing language has built-in variables for storing commonly used data. The width and height of the display window are stored in variables called `width` and `height`. If a program doesn't include `size()`, the `width` and `height` variables are both set to 100. Test by running the following programs

```
println(width + ", " + height);  // Prints "100, 100" to the console
```
3-08

```
size(300, 400);
println(width + ", " + height);  // Prints "300, 400" to the console
```
3-09

```
size(1280, 1024);
println(width + ", " + height);  // Prints "1280, 1024" to the console
```
3-10

Using the `width` and `height` variables is useful when writing a program to scale to different sizes. This technique allows a simple change to the parameters of `size()` to alter the dimensions and proportions of a program, rather than changing values throughout the code. Run the following code with different values in the `size()` function to see it scale to every window size.

3-11

```
size(100, 100);
ellipse(width*0.5, height*0.5, width*0.66, height*0.66);
line(width*0.5, 0, width*0.5, height);
line(0, height*0.5, width, height*0.5);
```

You should always use actual numbers in `size()` instead of variables. When a sketch is exported, these numbers are used to determine the dimension of the sketch on its Web page. More information about this can be seen in the reference for `size()`.

Processing variables that store the cursor position and the most recent key pressed are discussed in Input 1 (p. 205) and Input 2 (p. 223).

Exercises

1. *Think about different types of numbers you use daily. Are they integer or floating-point numbers?*
2. *Make a few `int` and `float` variables. Try assigning them in different ways. Write the values to the console with `println()`.*
3. *Create a composition that scales proportionally with different window sizes. Put different values into `size()` to test.*