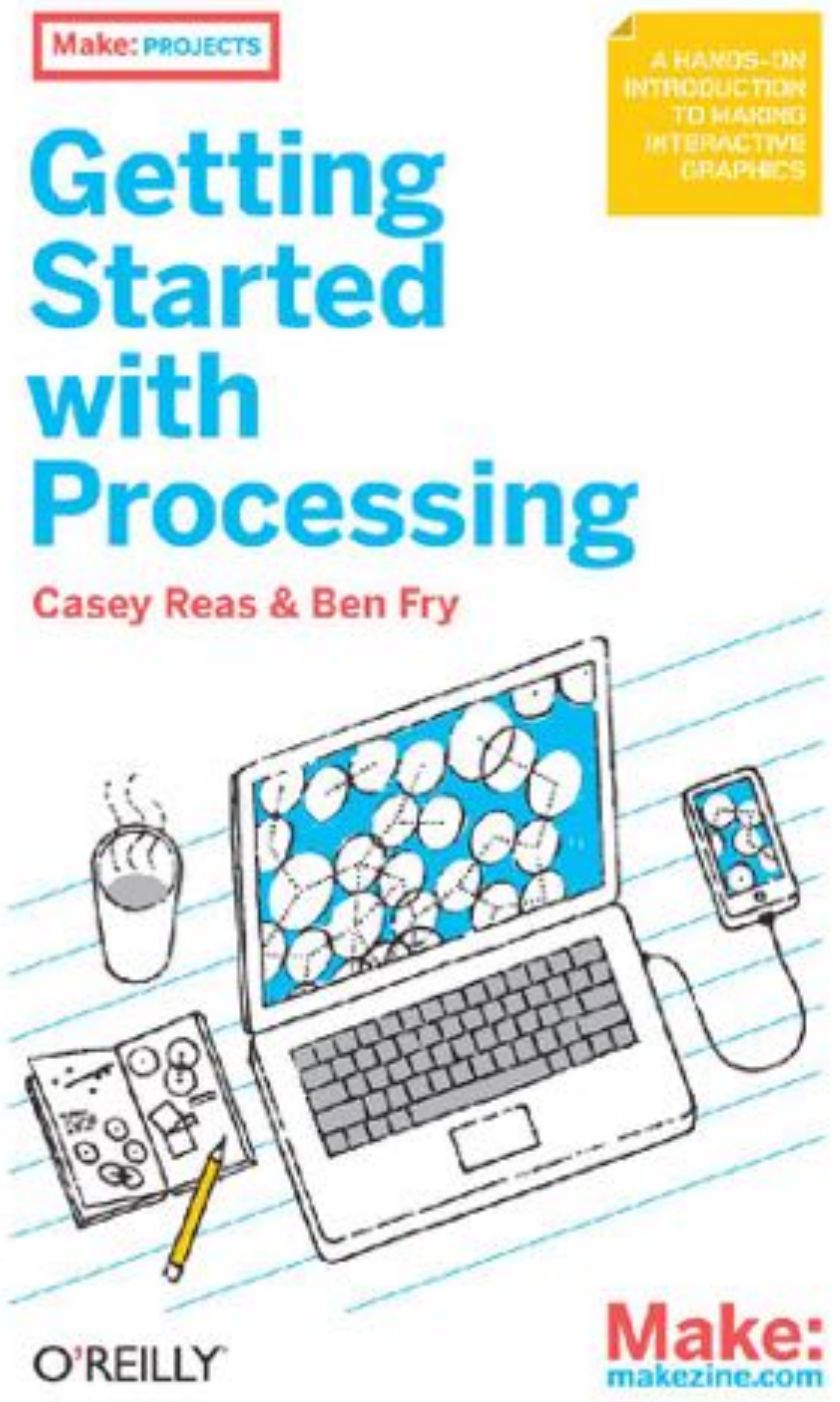
USC IML 288 __ 2018.09.10

4/Variables

all code illustrations taken from the processing.org website +


Make: PROJECTS

A HANDS-ON INTRODUCTION TO MAKING INTERACTIVE GRAPHICS

Getting Started with Processing

Casey Reas & Ben Fry

O'REILLY

Make: makezine.com

Open Source/Graphics Programming

Make: PROJECTS

Getting Started with Processing

Learn computer programming the easy way with Processing, a simple language that lets you use code to create drawings, animation, and interactive graphics. Programming courses usually start with theory, but this book lets you jump right into creative and fun projects. It's ideal for anyone who wants to learn programming, and serves as a simple introduction to graphics for people who already have some programming skills.

Written by the founders of Processing, this book takes you through the learning process one step at a time to help you grasp core programming concepts. Join the thousands of hobbyists, students, and professionals who have discovered this free and educational community platform.

» Quickly learn programming basics, from variables to objects
» Understand the fundamentals of computer graphics
» Get acquainted with the Processing software environment
» Create interactive graphics with easy-to-follow projects
» Use the Arduino open source prototyping platform to control your Processing graphics

Casey Reas is a professor in the Department of Design Media Arts at UCLA and a graduate of the MIT Media Laboratory. Reas's software has been featured in numerous solo and group exhibitions in the U.S., Europe, and Asia.

Ben Fry, a designer, programmer, and author based in Cambridge, Massachusetts, received his doctoral degree from the MIT Media Laboratory. He worked with Casey Reas to develop Processing, which won a Golden Nica from the Prix Ars Electronica in 2005.

Make: makezine.com

O'REILLY

US $19.99 CAN $24.99
ISBN: 978-1-449-37980-3

4/Variables

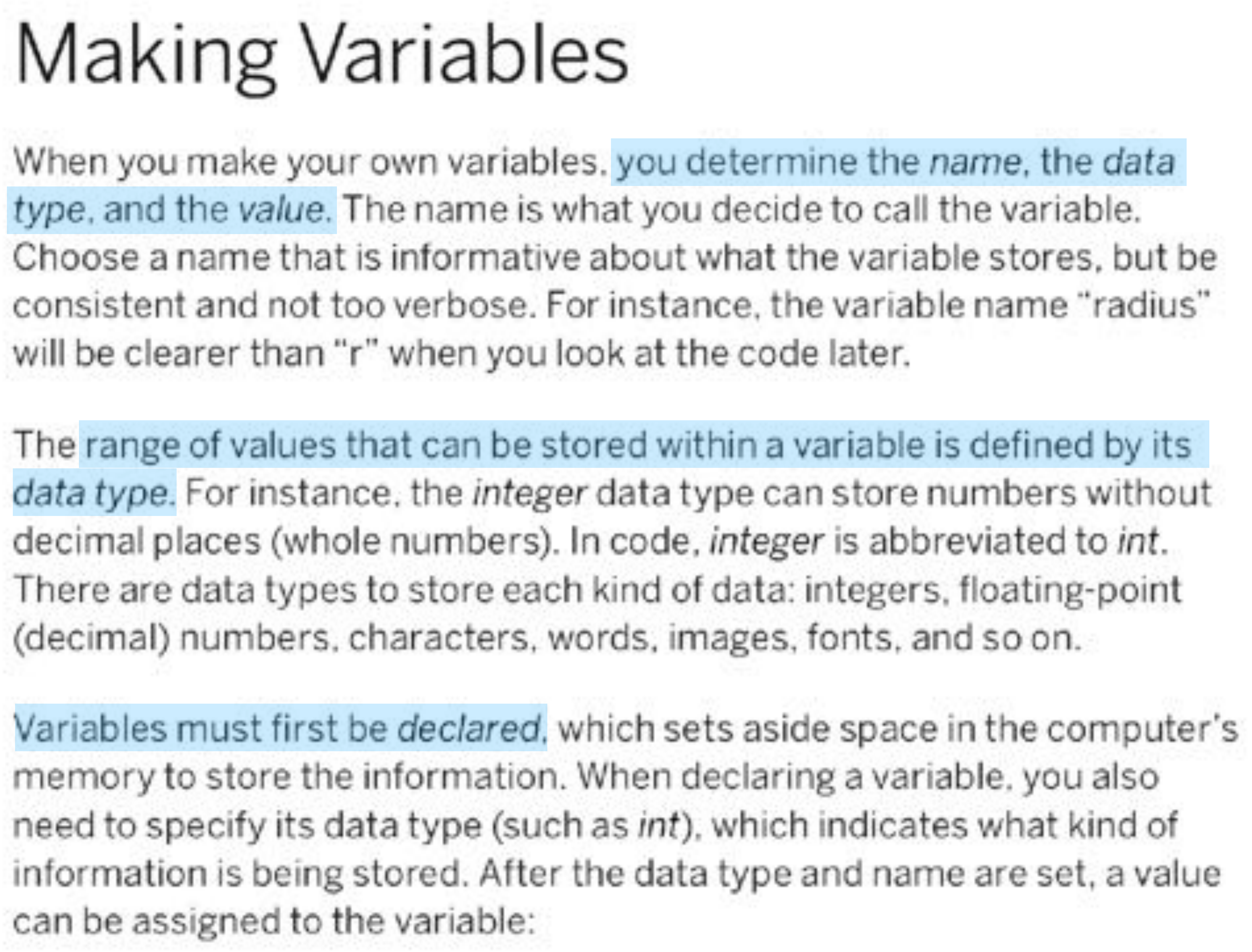
A variable stores a value in memory so that it can be used later in a program. The variable can be used many times within a single program, and the value is easily changed while the program is running.

The primary reason we use variables is to avoid repeating ourselves in the code. If you are typing the same number more than once, consider making it into a variable to make your code more general and easier to update.

Draw 3 circles. Each circle should have the same y position and diameter. The circles shouldn’t overlap (will have different x positions)

Draw 3 circles. Each circle should have the same y position and diameter. The circles shouldn't overlap (will have different x positions)

Change one variable so that the circles overlap.



Making Variables

When you make your own variables, you determine the *name*, the *data type*, and the *value*. The name is what you decide to call the variable. Choose a name that is informative about what the variable stores, but be consistent and not too verbose. For instance, the variable name "radius" will be clearer than "r" when you look at the code later.

The range of values that can be stored within a variable is defined by its *data type*. For instance, the *integer* data type can store numbers without decimal places (whole numbers). In code, *integer* is abbreviated to *int*. There are data types to store each kind of data: integers, floating-point (decimal) numbers, characters, words, images, fonts, and so on.

Variables must first be *declared*, which sets aside space in the computer's memory to store the information. When declaring a variable, you also need to specify its data type (such as *int*), which indicates what kind of information is being stored. After the data type and name are set, a value can be assigned to the variable:

```
int x;   // Declare x as an int variable
x = 12;  // Assign a value to x
```

```
int x = 12; // Declare x as an int variable and assign a value
```

Draw 4 lines. Two lines should connect the corners of your sketch. The other two lines should bisect the width and height of the canvas.

A Little Math —> Math

People often assume that math and programming are the same thing. Although knowledge of math can be useful for certain types of coding, basic arithmetic covers the most important parts.

In code, symbols like +, −, and * are called *operators*. When placed between two values, they create an *expression*. For instance, 5 + 9 and 1024 – 512 are both expressions. The operators for the basic math operations are:

+	Addition
−	Subtraction
*	Multiplication
/	Division
=	Assignment

Processing has a set of rules to define which operators take precedence over others, meaning which calculations are made first, second, third, and so on. These rules define the order in which the code is run. A little knowledge about this goes a long way toward understanding how a short line of code like this works:

```
int x = 4 + 4 * 5;  // Assign 24 to x
```

Draw 4 wide rectangles. Each rectangle should have the same width and height. The rectangles should be stacked one on top of another, starting on the "ground" (y = height). The rectangles should not be horizontally aligned.

Inside the parentheses are three statements, separated by semicolons, that work together to control how many times the code inside the block is run. From left to right, these statements are referred to as the *initialization* (*init*), the *test*, and the *update*:

The *init* typically declares a new variable to use within the *for* loop and assigns a value. The variable name *i* is frequently used, but there's really nothing special about it. The *test* evaluates the value of this variable, and the *update* changes the variable's value. Figure 4-1 shows the order in which they run and how they control the code statements inside the block.

The *test* statement requires more explanation. It's always a *relational expression* that compares two values with a *relational operator*. In this example, the expression is "i < 400" and the operator is the < (less than) symbol. The most common relational operators are:

>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
==	Equal to
!=	Not equal to

The update generally increments the variable generated by the *init*, and *test*ed for in the relational expression

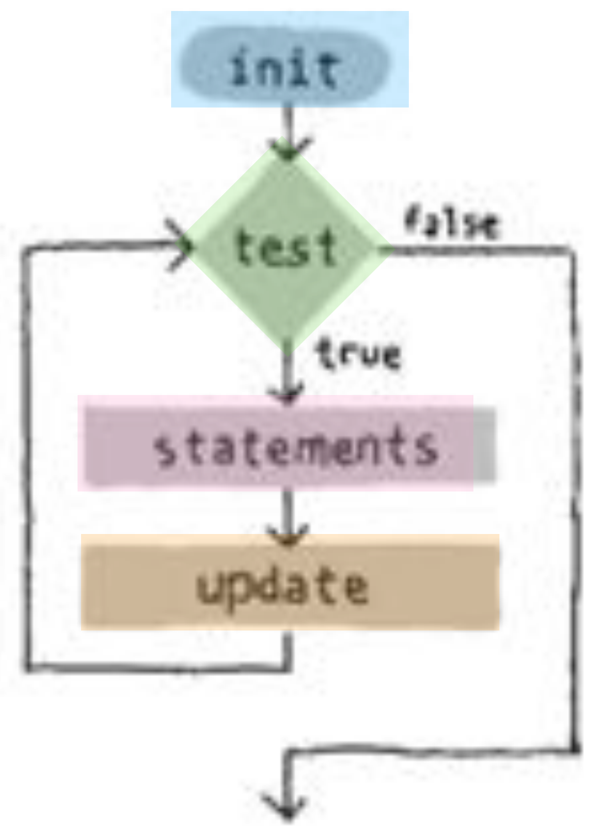


Figure 4-1. Flow diagram of a for loop.

```
for (init; test; update) {
  statements
}
```

The *for* loop is different in many ways from the code we've written so far. Notice the braces, the { and } characters. The code between the braces is called a *block*. This is the code that will be repeated on each iteration of the *for* loop.

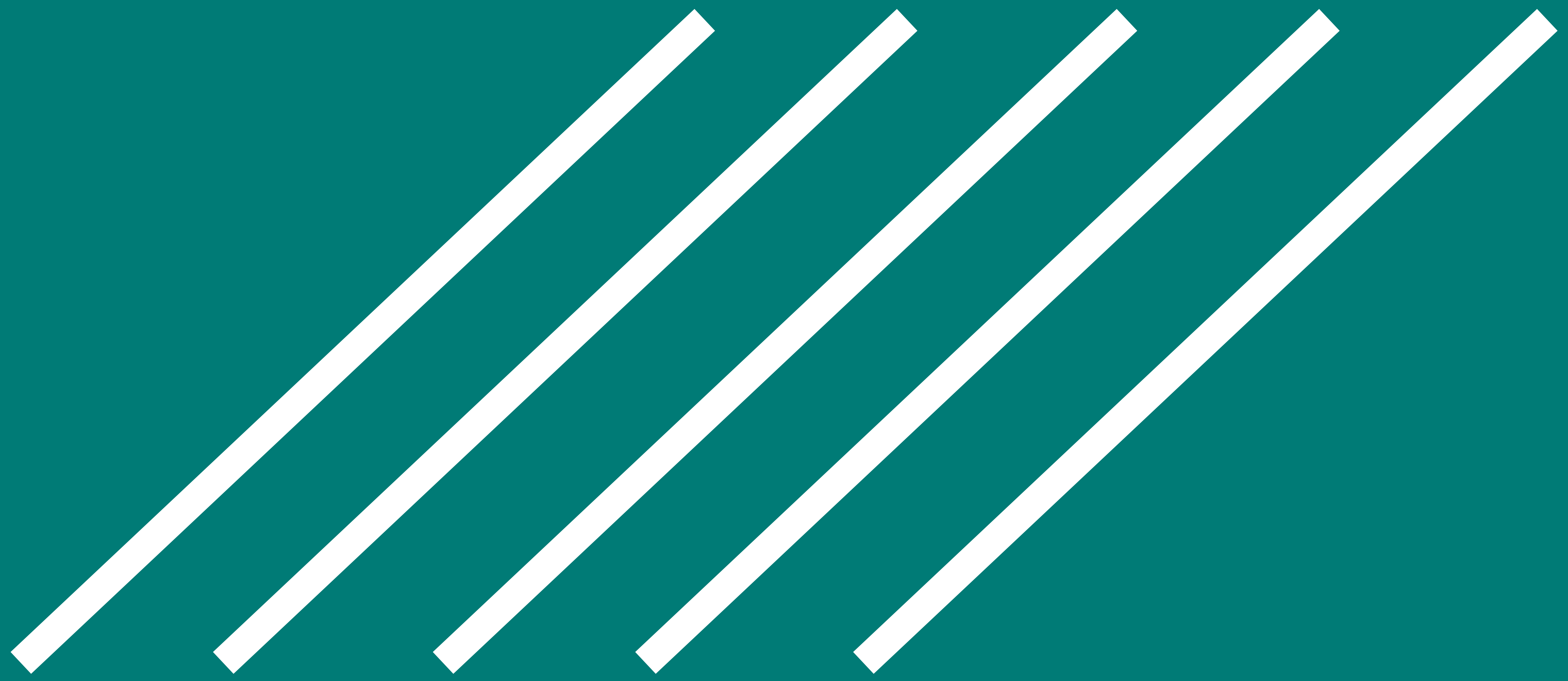
Some calculations are used so frequently in programming that shortcuts have been developed; it's always nice to save a few keystrokes. For instance, you can add to a variable, or subtract from it, with a single operator:

```
x += 10; // This is the same as x = x + 10
y -= 15; // This is the same as y = y - 15
```

It's also common to add or subtract 1 from a variable, so shortcuts exist for this as well. The ++ and −− operators do this:

```
x++; // This is the same as x = x + 1
y--; // This is the same as y = y - 1
```

Draw 5 non vertical, equidistant parallel lines that trend upward from left the right.



Draw 5 non vertical, equidistant parallel lines that trend upward from left the right.

Use a *for loop* to draw the same system described above.

Draw 5 non vertical, equidistant parallel lines that trend upward from left the right.

Use a *for loop* to draw the same system described above.

Use the *for loop* above to draw 100 lines instead of 5.

Use a *for loop* to draw a horizontal line of circles (20px diameter).

Use a *for loop* to draw a horizontal line of circles (20px diameter).

Use a new *for loop* to draw a vertical line of circles (20px diameter).

Use a *for loop* to draw a horizontal line of circles (20px diameter).

Use a new *for loop* to draw a vertical line of circles (20px diameter).

Use an *embedded for loop* to draw a grid of circles (20px diameter).

TAKASHI MURAKAMI

https://fashionista.com/2018/04/takashi-murakami-uniqlo-collaboration-interview

https://www.thebroad.org/art/takashi-murakami/hustlenpunch-by-kaikai-and-kiki

https://www.wright20.com/auctions/2014/09/art-design/130

Homework 02 // Due 2018.09.17

Part I. Generate a Graphic Inspired by Takashi Murakami
and think about how to use variables in your design, and the topics described in the previous lecture (form + color).

Size: appropriate to the work and fits within 1280x720 pixels
(e.g. a square work would be 720x720 pixels)

Part II. Use a *for loop* with Your Design to Generate a Texture

Size: appropriate to the work and fits within 1280x720 pixels