The Imaginary Institute
www.imaginary-institute.com
2Dcourse@imaginary-institute.com

2D ANIMATION & INTERACTION COURSE
WEEK 2: ANIMATION

WEEK 2: ANIMATION

This week we dig deeper into making images. We'll find that we can create our own named objects to hold numbers, and how that this gives us a huge amount of expressive power. In particular, we'll see how to use these objects, called *variables*, to produce animation. We'll see how to put all these pieces together in a program that we can run to create an animated piece.

The basic idea behind animation is simple. The computer automatically calls our program about 60 times a second, and we respond by giving the computer a series of instructions that describe the picture we want it to draw. The computer draws that picture and displays it, until it calls our program again, we create a new picture, and then that one is displayed until the process repeats yet again, over and over, 60 times every second until we stop the program.

If the individual frames don't vary too much from one another, then our brains fuse this sequence of rapidly-changing still images into the sensation of a moving image. If this sounds unlikely, it may seem a bit more credible once you know that this is exactly how both movies and television work.

So to create animation, we merely need to produce a series of still frames that vary just a little bit from one to the next over time. This week we cover the basic tools that you'll use in every program that produces animation. As we add more graphics power and interactive techniques over the next few weeks, these tools will remain our bedrock for producing animation.

As usual, many of the videos have Processing sketches (or programs) associated with them. These sketches fall into two categories:

Example sketches are meant for you to look at now, because they illustrate the points we're talking about. These sketches appear in a green section, like this.

Support sketches are programs that I wrote for use in the videos. They frequently use advanced techniques that we haven't covered yet. These sketches appear in a gray section, like this.

GROUP 1: OVERVIEW

We start by looking at this week's context, so you have an overview of where we're going and how the pieces all fit together.

W2 G1 V1 **Week 2 Overview** *(6m11s)*

We take a tour through the naming of objects, the idea of a variable, creating animation, and writing programs.

©The Imaginary Institute, 2013. All rights reserved.

GROUP 2: NAMES

Our programs will contain many objects, and we'll give each one a name.

W2 G2 V1a Names Part 1 *(7m1s)*
W2 G2 V1b Names Part 2 *(8m19s)*

We can name our objects just about anything we want, but there are a few rules that the system enforces. I also have a few conventions to recommend.

GROUP 3: VARIABLES

A *variable* is a key element of every computer program: it's an object that can hold a value, such as a number. Once we *assign* a value to a variable, it holds that value until we assign it something new. Different types of variables are designed to hold different types of information. We'll meet the two most important numerical types, and see how to use them.

W2 G3 V1a **Integers Part 1** *(9m38s)*
W2 G3 V1b **Integers Part 2** *(6m43s)*

An *integer* (or *int*) is a whole number with no fractional part.

drawings.pde

int_cookie.pde

A program to draw a stack of cookies on a table. There can only be a whole number of cookies – that is, the number of cookies is an integer.

W2 G3 V2 **Using Integers** *(12m55s)*

Combining integers with plus, minus, times, and divide, and the importance of parentheses when doing more than one operation on a line.

W2 G3 V3 **floats** *(9m48s)*

A *float* is a numbers that may have a fractional part.

W2 G3 V4 **Ints And Floats** *(10m33s)*

Although both integers and floats are numbers, they don't behave in exactly the same ways. We look at some very important differences.

fsum1.pde

Add a value to a float and print the result.

fsum7.pde

Add a value to a float 7 times and print the result. It's probably not what you'd expect.

fsum7test.pde

Test the result of adding a float to a float several times.

isum1.pde

Add a value to an int and print the result.

isum7.pde

Add a value to an int 7 times and print the result.

isum7test.pde

Test the result of adding an int to an int several times.

W2 G3 V5a **Dividing Integers Part 1** *(7m16s)*

W2 G3 V5b **Dividing Integers Part 2** *(6m44s)*

When you divide one integer by another, any fractional part in the result is immediately thrown away. This can be a very tricky problem to find and fix, so it's important to be aware of it, and know how to retain that fractional part when it's important to you.

integerDivision.pde

Demonstrate what happens when we divide integers.

GROUP 4: ANIMATION

To create animation, we produce a series of still images one after the other. If each new frame is just a little bit different than the previous one, and new images are displayed quickly enough, the human visual system will interpret the sequence of still images as a moving picture.

W2 G4 V1a **Animation Part 1** *(7m9s)*

W2 G4 V1a **Animation Part 2** *(8m47s)*

How animation works and the importance of frame rate.

AnimationDemo.pde

Create animations of different lengths and see how the length and frame rate affect the perceived smoothness of animation.

W2 G4 V2 **frameCount** *(9m38s)*

How to use `frameCount` to create animation.

frameCountDemo.pde

Using `frameCount` to animate a moving box.

GROUP 5: WRITING PROGRAMS

When our programs grow to more than a few lines long, it's useful to add some documentation, called a *comment*, right in the program to help us understand what's going on. If something goes wrong, we say that our program has a "bug." There are an endless variety of bugs, but there are a few standard techniques for tracking down the source of a bug so we can correct it.

W2 G5 V1 **Comments** *(7m57s)*

The two flavors of comments that you can add to your code.

W2 G5 V2 **Color Coding** *(9m40s)*

How Processing automatically color-codes your program to help you see what's going on at a glance.

W2 G5 V3 **Errors** *(9m48s)*

When our program isn't properly written, we say it has a *syntax error*. If something goes wrong while it's running, we say it has a *run-time error*. We see these errors and how they get reported.

W2 G5 V4 **Printing** *(8m29s)*

The two types of print statements and how to use them effectively.

W2 G5 V5 **Debugging** *(7m8s)*

Some general guidelines for tracking down and fixing bugs.

FourCircles.pde

A little sketch to draw four circles. It's easy to deliberately create and then fix bugs in sketches this short.

GROUP 6: RECAP AND HOMEWORK

We survey what we've seen this week, and then you get to put it into action in your homework.

W2 G6 V1a **Recap Week 2 Part 1** *(8m25s)*
W2 G6 V1b **Recap Week 2 Part 2** *(7m43s)*

A recap of everything we've seen this week. Remember that there's a PDF that summarizes this information.

Recap.pde

A simple animation. You can use this as a starting point for your homework if you like.

W2 G6 V2 **Homework Week 2** *(8m4s)*

Homework! Also summarized in the PDF handout.

GROUP 7: SUPPLEMENTS

The videos in this section are optional. They're here to illuminate interesting ideas, answer some questions you might have wondered about, and generally flesh out some of the topics we've seen.

W2 G7 V1 **longs** *(10m52s)*

The *long* data type holds whole numbers with more range than an int.

hairsInt.pde

A little sketch that tries to estimate the number of hairs on the heads of people in the US. It uses ints and doesn't get the right answer.

hairsLong.pde

A version of the hair-estimating program that uses longs, and gets the correct result.

W2 G7 V2 **Scientific Notation** *(10m39s)*

Really enormous numbers often have a lot of zeros at the end, and very small numbers (near 0) often have a lot of zeros at the start. Scientific notation is a shorthand for writing those kinds of numbers, and it's how Processing prints those kinds of numbers.

W2 G7 V3 **Doubles** *(9m45s)*

The *double* data type is like the float, but it has more range and more precision.

floatsize1.pde

What happens if we try to put a too-big number into a float.

floatsize2.pde

What happens if we calculate a too-big number and save it in a float.

precision.pde

Demonstrate that although a float stores a number very well, a double is even more precise.

precision_adding.pde

Demonstrate adding a small number to a float and to a double.

W2 G7 V4 **Preferences** *(5m51s)*

How to set Processing's preferences to your liking.

W2 G7 V5 **External Editor** *(10m53s)*

How to use your favorite text editor to write programs, bypassing Processing's built-in editor window.

EditorDemo.pde

A little four-line program with a built-in typo.

W2 G7 V6 **Tabs** *(7m59s)*

You can manage a large program by breaking up the text into separate files, all saved in the same directory. Processing will show these as tabs in the text editor.

Landscape.pde

Sky.pde

Snow.pde

Tree.pde

Utilities.pde

A big program for drawing picture postcards from another planet. The program is distributed over five source files.