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

2D ANIMATION & INTERACTION COURSE
WEEK 4: CONTROL

WEEK 4: CONTROL

This week is all about control: gaining more control over the graphics we draw, controlling how our programs run, and controlling our programs with the mouse. We'll also look at a concept that lets us think about how to structure animation, and look at some of Processing's built-in functions that can make our life more convenient.

We're moving fast now, and there's a lot of information this week. Give yourself the time to think about it all, and play with it. Try writing lots of tiny little programs, each one just big enough to play with one idea, until it locks into place for you. Then try expanding that idea and dreaming up other ways to use it.

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.

W4 G1 V1 **Week 4 Overview** *(8m38s)*

An overview of the many topics you'll learn about in this week's videos: new graphics shapes, variations on if statements, two varieties of loops, using the mouse, the idea of *state*, and some of Processing's built-in functions.

GROUP 2: GRAPHICS PRIMITIVES

We've come a long way with lines, boxes, and circles, but there are many more kinds of shapes available to draw with. We'll also look at the idea of an "arrow", which isn't a built-in shape, but a powerful conceptual tool for creating shapes.

©The Imaginary Institute, 2013. All rights reserved.

W4 G2 V1 **Points** *(3m20s)*

You can't get much simpler than a single colored point. A single dot of color is usually hard to spot, but you can accumulate lots and lots of points to make interesting and subtle visual effects.

pointCloud.pde

Make a "starburst" pattern out of many points.

pointCloud2.pde

Draw lots of curves, each made up of lots of points.

pointExplorer.pde

Drag around a point with the mouse, and see the corresponding command.

W4 G2 V2 **Triangles** *(3m50s)*

Triangles are easy to draw: just provide the three points.

triangleExplorer.pde

Drag around the three points of a triangle, and see the command that draws that triangle.

W4 G2 V3 **Quads** *(4m56s)*

Four-sided figures, also called *quadrilaterals* or *quads*, can be drawn just by naming four points. Quads can make shapes from a square to a skinny box to a bowtie.

quadExplorer.pde

Drag around the four points of a quadrilateral (or quad), and see the command that draws that quad.

W4 G2 V4 **Arcs** *(12m40s)*

An *arc* is a piece of a circle. We can use arcs to draw wedges, like those seen in pie charts.

arcExplorer.pde

Interactively explore arcs and the command that makes them.

arcRunner.pde

Free-running program that draws random wedges of an ellipse.

W4 G2 V5a **Angles Part 1** *(10m5s)*
W4 G2 V5b **Angles Part 2** *(6m42s)*

We see the connection between degrees (0-360 around a circle) and radians (0-2π, or about 0-6.28, around a circle). Processing thinks of angles starting at 3 o'clock and increasing as we go clockwise.

arcExplorer.pde
Interactively explore arcs and the command that makes them.

base60Explorer.pde
Divide the number 360 by integers, and see how many of them divide 360 into a whole number of equal pieces.

polyRoll.pde
Unroll a polygon of radius 1 to see the length of its perimeter, which approaches 2π as the number of sides increases.

W4 G2 V6a **Degrees and Radians Part 1** *(6m31s)*
W4 G2 V6b **Degrees and Radians Part 2** *(7m14s)*

Why 360 degrees and 2π radians are both such good choices for measuring angles.

arcExplorer.pde
Interactively explore arcs and the command that makes them

base60Explorer.pde
Divide the number 360 by integers and see how many of them divide 360 into a whole number of equal pieces.

polyRoll.pde
Unroll a polygon of radius 1 to see its perimeter, which approaches 2π as the number of sides increases.

W4 G2 V7 **Shapes** *(9m27s)*

We can draw a shape by supplying as many points as we like. Processing connects them up with straight lines, and will fill them if we want.

shapeExplorer.pde
Interactively manipulate the points of a shape made of straight lines that join those points.

W4 G2 V8 **Arrows** *(10m53s)*

Arrows are a very important concept for thinking about the relationships of shapes to each other. By manipulating arrows, we can manipulate those relationships easily.

Arrow.pde
ArrowDemo1.pde
Interactive program to demonstrate how to use an arrow to offset a point. This is a useful way to control its location with the mouse.

Arrow.pde
ArrowExplorer.pde
Interactive program to manipulate a random chain of connected arrows. You can drag the arrows to re-direct and re-size them.

GROUP 3: IF STATEMENTS REVISITED

The if statement is so handy, there are shortcuts available for two special types of situations that crop up frequently.

W4 G3 V1 **If Statements Revisited** *(13m1s)*

We look at two variations on the if statement: the *conditional* and *switch* statements.

GROUP 4: LOOPS

A *loop* allows us to repeat a bunch of lines of our program over and over, while changing one or more variables each time. This is something like calling a function over and over with different arguments, but it's far more convenient when we want to repeat it many times. We'll look at two different kinds of loops: the *while* loop and the *for* loop. Each type of loop can do everything the other type can do; the two varieties offer us two different ways to think about how we get the same work done.

W4 G4 V1a **While Loops Part 1** *(9m1s)*
W4 G4 V1b **While Loops Part 2** *(10m7s)*

We look at the basic principles of the while loop, and how to write and control one.

ballWithTrail.pde
Draw a "trail" behind a moving object.

simpleSphere.pde

Draw a fake sphere with a fake highlight.

whileDemo1.pde

Interactively step through a short program, one line at a time, showing the graphical result after each line is executed.

W4 G4 V2 **While Loops Revisited** *(12m56s)*

We look at a while loop in detail, watching it every step of the way.

whileDemoRunning.pde

Using a while loop, draw a stack of rectangles that get longer with time.

whileDemo3.pde

Interactively step through a short program, one line at a time, showing the graphical result after each line is executed.

W4 G4 V3 **While Loops Example** *(10m4s)*

How to draw a little step pyramid using a while loop.

StepPyramid.pde

Use a loop to draw a step pyramid. You can change the number of steps.

W4 G4 V4 **For Loops** *(10m40s)*

We look at the basic principles of the for loop, and how to write and control one.

W4 G4 V5 **For Loops Revisited** *(10m1s)*

We see how to create the stretching-rectangles example we made before with a while loop, but this time we use a for loop.

ForLoopRunning.pde

Using a for loop, draw a stack of rectangles that get longer with time.

forDemo3.pde

Interactively step through a short program, one line at a time, showing the graphical result after each line is executed

W4 G4 V6 **Break and Continue** *(2m18s)*

We can use *break* and *continue* to fine-tune the execution of a loop.

break_continue.pde

A little demonstration of break and continue on rows of circles.

W4 G4 V7 **Break** *(10m27s)*

A closer look at the break statement.

BreakDemo.pde

An interactive demonstration of the break statement on a row of circles.

W4 G4 V8 **Continue** *(11m20s)*

A closer look at the continue statement.

ContinueDemo.pde

An interactive demonstration of the continue statement on a row of circles.

GROUP 5: THE MOUSE

Touch screens are becoming more popular every day, but the mouse is still an important input device. And many touch-screen systems report touch information in a way very similar to the way mouse information is reported. Here we look at how to respond when the user moves the mouse, or pushes or releases a mouse button.

W4 G5 V1 **The Mouse** *(12m17s)*

The functions that we use to respond to the user's manipulation of the mouse.

BezBox.pde
mouseExplorer.pde

Show the different functions that get called as the mouse is moved, clicked, and dragged.

W4 G5 V2a **Using The Mouse Part 1** *(10m3s)*
W4 G5 V2b **Using The Mouse Part 2** *(7m37s)*

Why mouse-handling functions need to be short and fast.

game04.pde

A very simple game of skill that uses the mouse.

simpleMouseReporter.pde

Print mouse information to the output window.

MouseProcGraphics.pde

Draw a timing diagram showing the effects of short versus slow mouse-handling functions.

W4 G5 V3 **Dragging** *(5m8s)*

How to avoid a sudden jump when selecting and moving an on-screen object.

simpleDrag.pde

The simple way to select and drag, which results in a little jump at the start.

betterDrag.pde

A better way to select and drag that doesn't start with a jump.

GROUP 6: STATE

When we create animation, we're only drawing one frame at a time. In order to draw the correct image for each frame, we need to remember where we are in every aspect of the animation. We use the term *state* to refer collectively to all of this remembered information.

W4 G6 V1 **Animating With State** *(10m37s)*

The idea of *state*, and how we can use it to create animation.

CircleOverBoxes.pde

An interactive sketch that uses the mouse and state to create a changing image.

wheels.pde

Using state to create an animation that looks much more complicated than it really is!

arcRotation.pde

Interactive program to move two arc endpoints around a circle.

W4 G6 V2 **Using State** *(11m11s)*

How to use state to create a more advanced project involving a fox and hen.

foxAndHenDist.pde

You control the hen with your mouse, and the fox chases after you, creating a nicely smoothed trail.

arrowScaling.pde

Interactive demonstration of how to scale an arrow by scaling its x and y components.

GROUP 7: USEFUL FUNCTIONS

Processing offers us a variety of built-in functions that can save us a lot of time and effort. We survey them here and see some applications of them.

W4 G7 V1 **Map** *(10m50s)*

The `map()` function lets us take a number from one range, and find its corresponding value in another range. We use this a lot when working with the mouse, converting its X and Y into values that control an object.

W4 G7 V2 **Using map** *(9m30s)*

Some examples of using `map()`.

mouseMap2Lines.pde

An interactive demo for visualizing the map operation.

ScreenToBoxMapping.pde

An interactive demo showing how to use map twice to convert the mouse X and Y to a point in a different rectangle.

sunriseMap.pde

An interactive demo that uses map to convert the time on a slider to an angle that we use for drawing the sun.

temperatureMapper.pde

An interactive demo to show how to use map to convert temperatures from Fahrenheit to Celsius, or vice-versa.

W4 G7 V3 **Lerp** *(12m16s)*

The `lerp()` function lets us create a smooth blend between two numbers.

circlesLerpDemo.pde

An interactive program that lets you drag an ellipse around on the screen. We use lerp to blend the shape in color, shape, and size.

W4 G7 V4 **lerpColor** *(7m52s)*

Using `lerpColor()`, we can easily create a smooth blend between two colors. The colors that we see along the blend depend on whether our current color mode is RGB or HSB.

colorMixer.pde

A little interactive color-mixing program that uses lerpColor.

colorSpaceLerping3D.pde

Looking at the path taken by lerpColor as it blends colors in 3D, in both the RGB and HSB color spaces.

lerpColor.pde

An interactive demo to blend two colors using lerpColor. We show the blend simultaneously in RGB and HSB spaces.

lerpColorWithWheel.pde

An interactive demo to blend two colors using lerpColor. We show the blend simultaneously in RGB and HSB spaces, and show where the two endpoint colors lie on a color wheel.

W4 G7 V5 **Dist** *(11m30s)*

The `dist()` function is convenient way to find the distance between any two points.

diskAndMouse.pde

An interactive demo that uses dist to change the color of circles that lie inside another circle that's under control of the mouse.

foxAndHen.pde

Our fox and hen program, which uses dist to find the distance between the fox and the hen to controlcontrolling the fox's speed.

W4 G7 V6 **Useful Functions** *(10m59s)*

In addition to the functions above, there are a bunch of small functions that are very useful when we're making patterns with numbers. We cover the functions `abs()` (absolute value, which simply forces its argument to be positive), `constrain()` (which clamps the input so it's not less than one value or greater than another), `int()`, `floor()`, `ceil()`, and `round()` (which convert floats to ints in slightly different ways), `sqrt()` (square root), and `sq()` (which just multiplies a number with itself).

usefulFunctions.pde

An interactive demo that lets us see the result of applying the functions above to a number on the number line.

GROUP 8: RECAP AND HOMEWORK

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

W4 G8 V1 **Week 4 Recap and Homework** *(7m3s)*

A recap of everything we've seen this week, followed by the homework assignment. Remember that there are PDF files, one each for summarizing the recap and the homework.

GROUP 9: 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.

W4 G9 V1 **Stroke Joins** *(5m53s)*

When we draw connected series of strokes, we have a choice of the graphical style used to connect them.

CapAndJoinDemo.pde

Interactive demo for exploring the different pairings of cap and join styles.

strokeJoin.pde

Show the three different stroke join styles at once.

W4 G9 V2 **Modes** *(7m59s)*

We can tell Processing to interpret the four arguments to `rect()` and `ellipse()` in different ways. Our choice of *mode* determines what gets drawn. I recommend avoiding these modes, but they're good to know about because you may see them used in other people's code.

ModeExplorer.pde

Explore the different ways in which the mode settings change our interpretation of the arguments to rect and ellipse.

W4 G9 V3a **Writing Fox And Hen Part 1** *(9m34s)*

W4 G9 V3b **Writing Fox And Hen Part 2** *(9m33s)*

W4 G9 V3c **Writing Fox And Hen Part 3** *(5m1s)*

We build the fox and hen program together from the very start to the final version.

foxAndHen01.pde
foxAndHen02.pde
foxAndHen03.pde
foxAndHen04.pde
foxAndHen05.pde
foxAndHen06.pde
foxAndHen07.pde
foxAndHen08.pde
foxAndHen09.pde
foxAndHen10.pde

A series of step-by-step saves of the fox and hen program that we write in the above videos.