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

2D Animation & Interaction Course
Week 1: A Strong Foundation

WEEK 1: A STRONG FOUNDATION

Week 1 is all about building a good foundation. I'm sure you want to be writing exciting programs right away, but it's important to have the basic ideas solidly in place, or you can find yourself getting stuck over and over again in places where you should be sailing easily. So we'll start out slowly, but within a few weeks we'll be zooming along!

This week we get Processing downloaded and installed, and we learn the basics of graphics: how to talk about locations on the screen, how to specify colors, and how to draw lines, boxes, and ellipses in a variety of styles.

Each video is prefixed by numbers identifying its week, group, and position in that group. So this week all sketch names begin with W1 for Week 1, followed by G and a number for the group, and then V and a number for the video. Some videos are in multiple parts to make them easier to watch; these parts all have the same video number but have a suffix of a, b, and so on. In general, it's best to watch the videos in order, from the first video of the first group to the last video of the last group.

Many of the videos have Processing sketches (or programs) associated with them. These files end with the extension .pde, which stands for Processing Development Environment. The sketches listed here fall into two categories:

Example sketches are meant for you to look at now, because they illustrate the points we're talking about. You can see how I do things, and you use these programs as starting points for your own work, simply by copying and pasting the code. These sketches appear in a green section, like this.

Support sketches are programs that I wrote to create demonstrations and other visuals for the course videos. They frequently use advanced techniques that we haven't covered yet. They're not written for you to understand or make use of at this point. I've included them in case you'd like to take a peek behind the scenes, or you're curious for any reason. 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.

W1 G1 V1 Week 1 Overview *(7m46s)*

A brief survey of what we'll be discussing this week: installing Processing, how to name and use colors, how to draw some important shapes, and how to put it all together to make a program and draw a picture.

GROUP 2: INSTALLING AND USING PROCESSING

We start out by preparing for the journey, making sure we have the right attitude and expectations. Then we install Processing, take a tour of the interface, and make our first picture.

W1 G2 V1 **Processing Tour** *(11m23s)*

A tour of the sorts of animations and interactive program that you'll be able to write with the tools we'll cover in this course.

W1 G2 V2 **Programming As A Medium** *(4m18s)*

Writing a program on the computer is a creative act in a particular medium, just like painting with watercolors or sculpting with clay. Success in any medium depends on having a good understanding of its strengths and weaknesses, so you're using your time and energy wisely.

W1 G2 V3a **Installing Processing on Windows** *(7m28s)*

How to download and install Processing on a Windows-based computer.

W1 G2 V3b **Installing Processing on Mac** *(7m0s)*

How to download and install Processing on a Mac.

W1 G2 V4a **The Processing Window Part 1** *(7m43s)*
W1 G2 V4b **The Processing Window Part 2** *(9m28s)*

The Processing Window is the on-screen interface where you write, debug, and run your programs. These videos survey the window so you can use it comfortably.

W1 G2 V5 **Creating A Picture** *(12m35s)*

We jump right in and write a program to create a simple picture.

FirstPicture.pde
Create a little picture of two boxes.

GROUP 3: COLOR

Color is essential to everything we draw. Processing offers a couple of different ways to specify colors.

W1 G3 V1a **Color Systems Part 1** *(8m44s)*
W1 G3 V1b **Color Systems Part 2** *(5m50s)*

There are many different ways to organize and name colors. We look at the color systems that will be important to us when using Processing.

W1 G3 V2 **RGB** *(8m21s)*

Unlike pigment, the three primaries of light are red, green, and blue. We can identify any color of light as a point in a cube with sides labeled red, green, and blue (or RGB).

RGB01.pde

An interactive visualization of the RGB color cube. Use the keyboard to turn on and off various drawing options.

W1 G3 V3 **HSB** *(8m53s)*

Another useful color space describes colors using the qualities hue, saturation, and brightness. We can identify any color as a point in a cone defined by hue, saturation, and brightness (or simply HSB).

HSB01.pde

An interactive visualization of the HSB color cone. Use the keyboard to turn on and off various drawing options.

HSBCube01.pde

An interactive visualization of the HSB color space drawn as a cube (for comparison with the RGB cube). Use the keyboard to turn on and off various drawing options.

W1 G3 V4 **Color Pickers** *(11m21s)*

A survey of some interactive tools for choosing colors.

W1 G3 V5 **Defining HSB Colors** *(9m42s)*

How to specify a color in HSB.

W1 G3 V6 **Using Colors** *(4m36s)*

Graphics objects are drawn using two colors, one each for its stroke (a line around its outer edge) and its fill (the color that makes up its insides).

W1 G3 V7 **Transparency** *(9m51s)*

By making colors transparent, objects drawn with those colors are transparent as well. You can draw objects that are fully opaque (and hide anything behind them), fully transparent (and thus invisible), or anywhere in between.

BoxTransparency.pde

Demonstrate transparency of stroke and fill using the mouse to control the transparency of boxes.

HSBCube01.pde

Demonstrate transparency using the mouse to control the transparency of two ellipses.

W1 G3 V8 **Grays** *(3m45s)*

We can use shortcuts for naming colors that are shades of gray.

GROUP 4: BASIC GRAPHICS

To draw something on the screen, we need to tell Processing where it's located, and what shape it has. In this group we see how to draw three important basic shapes: lines, rectangles (including squares), and ellipses (including circles).

W1 G4 V1 **Coordinates** *(6m0s)*

To draw objects on the screen, we need to tell Processing where to put them. The *coordinate system* is the tool that lets us do that.

coordsDemo.pde

Demonstrate coordinates by letting the user move a point around in a coordinate system, and printing out that point's coordinates.

W1 G4 V2 **fill and stroke** *(11m55s)*

Every object can have a stroke (a colored line around its outer edge) and a fill (a color that floods its interior).

W1 G4 V3 **Style** *(10m21s)*

By changing the stroke color, the stroke thickness (or weight), and the fill color, we can create many different visual styles.

stylesDemo.pde

Show different drawing styles side by side, altering the fill color, stroke color, and stroke weight. The screen starts out blank. Press the space bar to add in the next one in the sequence.

W1 G4 V4 **Lines** *(2m38s)*

To draw a line, we give Processing the x and y values for each of the two endpoints.

lineExplorer.pde

Demonstrate how lines are drawn. Click and drag on the big circles to move the endpoints around. Press space to toggle the circles on and off.

W1 G4 V5 **Rectangles** *(8m31s)*

Rectangles are described (by default) with the x and y values of one corner, followed by the width and height measured from that corner.

rectExplorer.pde

Demonstrate how rectangles are drawn. Click and drag on the upper-left point to move it around. Drag the upper-right point to change the width, and the lower-left point to change the height. Press space to toggle the circles on and off.

W1 G4 V6 **Ellipses** *(7m39s)*

Ellipses are described (by default) by the x and y of the center, and the width and height of the box, centered at that point, that just encloses the ellipse.

ellipseExplorer.pde

Demonstrate how ellipses are drawn. Click and drag on the center point to move it around. Drag the handles at the far right or bottom to change the shape of the ellipse. Press space to toggle the circles on and off.

GROUP 5: PUTTING IT TOGETHER

We take everything from above and use it to write a *sketch*, or program, to draw a picture.

W1 G5 V1 **Putting It Together Part 1** *(8m17s)*
W1 G5 V1 **Putting It Together Part 2** *(7m45s)*

We gather together everything we've seen and write a program to produce a picture made up of several graphics objects.

drawing01.pde

A minimal Processing program. All it does is create a graphics window and tell it to draw things smoothly, but it doesn't actually draw anything. You can use this as a starting point for your own projects.

drawing04.pde

A simple Processing program. Open a graphics window and tell it do draw things smoothly. Fill it with a background color. Tell Processing to draw black, 1-pixel thick strokes around objects (these are the defaults, and therefore we don't have to set them, but they're here to show how it's done so you can change these qualities). Then we draw a bunch of ellipses and boxes, each of a different color.

GROUP 6: RECAP AND HOMEWORK

We summarize this week's material so it's all in one place, and then it's time for you to put it into practice in your first homework assignment.

W1 G6 V1 **Week 1 Recap** *(4m3s)*

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

W1 G6 V2 **Week 1 Homework** *(8m55s)*

Homework for week 1. Also summarized in the PDF handout.

onerect.pde

An example of a very simple sketch to draw a box in a graphics window.

simple_composition.pde

A very simple composition in Processing. We change the fill and stroke colors along the way.

GROUP 7: SUPPLEMENTS

The videos in this section are optional supplements. 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.

W1 G7 V1 **Processing On Windows** *(3m58s)*

A survey of the minor differences between where things appear in the Processing environment on Macintosh computers (where the videos were made) and on Windows.

W1 G7 V2 **Why 255** *(12m51s)*

The number 255 (and the number 256) pop up all the time when we use colors. Where did this strange number come from?

Why255.pde

Interactive demo showing 256 shades of color.

W1 G7 V3 **Color Displays** *(12m19s)*

A survey of the different types of color displays that are popular today, and how they work to produce seemingly smooth fields of color.

W1 G7 V4 **Color Blending** *(8m52s)*

When you blend colors, the results depend on which of Processing's two colors spaces you're using (RGB and HSB). We look at where these differences come from, and when you might want one or the other.

blendHSB.pde

Interactive demo to show blending in HSB space.

blendRGB.pde

Interactive demo to show blending in RGB space.

lerpColor.pde

Demonstrate color blending in both RGB and HSB at once.

lerpColorWithWheel.pde

Blend in RGB and HSB at once, with a color wheel in the corner.

W1 G7 V5 **Stroke Caps** *(8m38s)*

When you draw thick lines, you can control how the ends are drawn. Processing offers you several different styles for these end "caps."

strokeCapDemo.pde

Interactive demo to show different stroke caps on different lines.

strokeCapTrio.pde

Show the three different types of stroke caps at once.

strokeJoin.pde

Show the three different types of stroke joins at once.

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.

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.

2D Animation & Interaction Course: Week 3: Interaction

WEEK 3: INTERACTION

Week 3 is about building up our programming skills and getting started on writing interactive animations. Now that you have solid control of the fundamentals, we'll start moving much faster to expand our expressive power and control.

This week is where we start to pick up steam. We'll cover a lot of vital material, so make sure you give yourself the time to understand and internalize it. Try getting in the habit of writing lots of tiny little toy programs to try out a new technique or idea. Then mess around with it, exploring what you can do with it and what ideas it stimulates.

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. Then we'll write a very little game. We'll do it pretty quickly and a lot of the steps will be new, but by the end of this week you'll understand everything we've done.

W3 G1 V1 Week 3 Overview *(6m31s)*

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. We'll see a bunch of new ideas: variables, functions, if statements, and responding to the keyboard.

W3 G1 V2a **A MiniGame Part 1** *(9m10s)*
W3 G1 V2b **A MiniGame Part 2** *(7m9s)*
W3 G1 V2c **A MiniGame Part 3** *(7m42s)*

To show what you can do with this week's material, we'll write a little game together. I'll go through it all rather quickly. By the end of this week everything here will be familiar to you.

MiniGame.pde

A ball moves from the left of the screen to the right. Press the spacebar to instantly stop its horizontal motion and cause it to drop down. If it lands inside the gap at the bottom, you score big points!

GROUP 2: VARIABLES

We'll expand our control of variables in two ways. First, we'll see how to create and use color variables so we don't have to type in numbers every time we change colors. Then we'll look at a number of universally-used shorthands for doing basic numerical operations with variables.

W3 G2 V1a **Color Variables Part 1** *(9m11s)*
W3 G2 V1b **Color Variables Part 2** *(9m50s)*

How to create color variables, and how to get the color components back out of them.

W3 G2 V2a **Using Color Variables Part 1** *(7m18s)*
W3 G2 V2b **Using Color Variables Part 2** *(7m49s)*

The proper technique for changing and combining color variables.

AddColors.pde

How to add two color variables.

DoubleColor.pde

How to make a color variables twice as bright.

NestedEllipsesStart.pde

The hard way to make an ellipse appear to have a hole in it.

NestedEllipsesFinal.pde

The easier way to make an ellipse appear to have a hole in it. This approach also makes it much easier to change the colors in the drawing.

W3 G2 V3a **Shorthand Arithmetic Part 1** *(12m36s)*
W3 G2 V3b **Shorthand Arithmetic Part 2** *(4m54s)*

We look at some universally-used shortcuts for doing common arithmetic with numerical variables. Once you've got these under your fingertips you'll use them all the time.

GROUP 3: FUNCTIONS

We often want to package up bits of code so that we can repeat them in several places without having to retype everything. The mechanism of a *function* lets us do that. Even better, we can tell the function that some of its variables (called *arguments* or *parameters*) should take on particular values each time the function runs. This gives us an enormous amount of expressive power, since the same steps repeated with different values can give us dramatically different results.

W3 G3 V1a **Functions Part 1** *(9m45s)*
W3 G3 V1b **Functions Part 2** *(11m55s)*

How to create our own functions.

Stacks.pde

We use a function to draw a stack of three ellipses. By calling the function multiple times, we can easily draw lots of stacks of different colors.

textAnimator.pde

Animated text to show how to go from a line of English that describes a function to the Processing version.

W3 G3 V2a **Using Functions Part 1** *(7m29s)*
W3 G3 V2b **Using Functions Part 2** *(9m53s)*

How to use the functions we create.

scoring.pde

Using a function to find the score for a very simple game. In this sketch, there's no interaction yet.

Stacks.pde

We use a function to draw a stack of three ellipses. By calling the function multiple times, we can easily draw lots of stacks of different colors.

W3 G3 V3a **Locals and Globals Part 1** *(10m11s)*
W3 G3 V3b **Locals and Globals Part 2** *(10m2s)*

Variables can be *local* to a function, meaning they only have a value to lines of code inside that function, or they can be *global* to the whole program, and accessible everywhere.

FruitCount.pde

We deliberately create a local variable with the same name as a global variable. This almost always leads to confusion and misbehaving programs.

MovingEllipse.pde

We see how to use global variables to remember the position of a circle from one frame to the next. This is a key technique for writing interactive programs.

W3 G3 V4 **Useful Globals** *(7m39s)*

Processing maintains a variety of global variables for us. Here we look at six of the most important ones.

GROUP 4: IF STATEMENTS

Every time you run an interactive program, you can get a different result. If it's a game, the user might score a goal immediately, or never. If it's a drawing program, the user might draw a snowman, a cupcake, or an abstract field of red blobs. Each time through it's different. A central mechanism for giving your program this kind of flexibility is the *if statement*.

W3 G4 V1 **If Statement Overview** *(3m12s)*

A look at if statements and what they can do for us.

W3 G4 V2 **Booleans** *(6m30s)*

A *boolean* is a data type that can hold only two values: the pre-defined key words `true` or `false`. These booleans are used in tests of all kinds, including if statements.

ShowBools.pde

Print out boolean variables, see how to pass them to functions, and how to return them from functions.

W3 G4 V3a **If Statements Part 1** *(10m54s)*
W3 G4 V3b **If Statements Part 2** *(9m52s)*

Using logical tests, we can build if statements that choose whether to execute either one set of statements, or another.

backgroundColor.pde

Increase the redness of the background over time.

textAnimator.pde

Animate a line of text from an English statement into an if statement in Processing.

W3 G4 V4a **Logical Tests Part 1** *(7m41s)*
W3 G4 V4b **Logical Tests Part 2** *(6m6s)*

We look at two ways to combine two Booleans: whether both are true (called the AND test) or either is true (called the OR test).

MouseInBoxesAnd.pde

We use the mouse to explore logical tests with if statements.

testDemoWithMouse.pde

Another example of using if statements to control interactive behavior.

W3 G4 V5a **Using Logical Tests Part 1** *(12m11s)*
W3 G4 V5b **Using Logical Tests Part 2** *(10m54s)*

Using AND and OR tests to determine if a point is inside a given box, if it's inside a given circle, or if it's inside a more complicated shape.

W3 G4 V6a **Combining If Statements Part 1** *(8m26s)*
W3 G4 V6b **Combining If Statements Part 2** *(11m34s)*

Using the *else* clause in an if statement lets us nest together multiple statements to make a compact and easily-understood chain of tests for handling complicated situations.

GROUP 5: KEYBOARD

Here we see how to write code that responds when the user has pressed a key on the keyboard, and take different actions depending on which key it was. We also see how to respond to when a key is released, and how to handle a few special but common situations.

W3 G5 V1 **char** *(4m10s)*

The *char* data type holds a single character.

W3 G5 V2a **Keyboard Part 1** *(10m37s)*
W3 G5 V2b **Keyboard Part 2** *(11m33s)*

How to respond to typical keyboard events, like a key going down or being released.

KeyboardDemo.pde

Demonstrates the code that lets us use the keyboard in the most common and important way: changing the value of a global variable.

keyExplorer.pde

Show when different keyboard functions are called in response to the user's actions, and what arguments they receive.

W3 G5 V3 **Keyboard AutoRepeat** *(9m4s)*

Many computers automatically repeat a key if it's held down. Here's how to keep that from becoming a problem.

autorepeat.pde

Demonstration of how to avoid problems due to auto-repeat.

W3 G5 V4 **Keyboard Extras** *(7m14s)*

Responding to non-printing keyboard keys, like the shift, control, or arrow keys, requires some special treatment.

KeyboardDemo.pde

Demonstrates the code that lets us use the keyboard in the most common and important way: changing the value of a global variable.

keyExplorer.pde

Show when different keyboard functions are called in response to the user's input, and what arguments they receive.

GROUP 6: EXAMPLES

This week we've covered a lot of material. Here we put the pieces together in two examples.

W3 G6 V1a **A Bouncing Ball Part 1** *(9m46s)*
W3 G6 V1b **A Bouncing Ball Part 2** *(10m39s)*
W3 G6 V1c **A Bouncing Ball Part 3** *(4m23s)*

We write a program that draws a little circle that moves in a straight line across the window until it hits one of the four edges, where it "bounces" like a billiard ball. This technique is a common building-block for many different kinds of animation.

BouncingBall.pde
The basic technique for a bouncing ball.

W3 G6 V2a **A Little Game Part 1** *(9m52s)*
W3 G6 V2b **A Little Game Part 2** *(10m41s)*

We write a little game of (not much) skill. Pressing the space bar the right moment will cause a "dart" to land in a goal zone. You can also speed up or slow down the game to control the difficulty.

PmanGame.pde
The little dart-shooting game.

GROUP 7: RECAP AND HOMEWORK

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

W3 G7 V1 **Week 3 Recap** *(4m48s)*

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

W3 G7 V2 **Week 3 Homework** *(7m8s)*

Homework! Also summarized in the PDF handout.

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.

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.

2D Animation & Interaction Course
Week 5: Lists and Transforms

WEEK 5: LISTS AND TRANSFORMS

We'll start out this week by looking at *random numbers*. By asking the computer to give us an unpredictable number from within a given range, we can make each run of our program different from every other. With random numbers under our belts, we'll then turn to our two main topics: lists and transforms.

The programmer's name for a list is an *array*. With arrays, we can conveniently keep around information on lots and lots of objects without creating individual variables for each one. This lets us easily create complicated, rich assemblies of objects that are all of the same family, but where each object is unique in its appearance, motion, and even behavior.

We'll then look at *transforms*, or *transformations*. Transforms are a simple but very powerful idea: they let us manipulate the coordinate system that we use for all our drawing operations. Everything that we draw after such a manipulation will be affected. For example, if we make the coordinate system bigger, everything we draw from then on will be bigger. If we rotate the coordinate system, then everything we draw from then on will be rotated as well. The idea is simple, but mastering transforms can be tricky and surprising, so we'll look at them carefully.

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.

W5 G1 V1 **Week 5 Overview** *(7m45s)*

We'll get an overview of lists and transforms, and see why they will be so useful to us.

GROUP 2: RANDOMNESS

Random numbers are a great way to add some surprise and unpredictability to our programs. We can ask the computer to provide us with a number from within a given range, and then we can use that number in any way we want, from setting the location or shape of an object to controlling how it moves or behaves.

W5 G2 V1 **RandomNumbers** *(10m5s)*

How to ask for a random number in a given range, and some important gotchas to look out for to make sure you get what you intended.

BouncingBoxes.pde

Many thin vertical strips moving up and down randomly.

JitterBalls.pde

Random balls jiggling around randomly.

Kaleidoscope4.pde

Random circles move outward from the center, jiggling a little bit randomly as they go. We draw each circle 8 times to make a kaleidoscope.

W5 G2 V2 **Random Seeds** *(10m53s)*

Using a *seed*, we can force Processing to produce the same random numbers each time we run the program. This is a great help when tracking down a bug that can come and go depending on which random numbers are used.

circlesWithSeed.pde

Using the seed value to get the same picture each time.

W5 G2 V3 **Flower Garden** *(12m10s)*

We use lots of random numbers to draw a little stylized garden of flowers. There's no knowing how many flowers we'll see, where they'll be located, what colors they are, or even how many petals each flower will have.

FlowerGarden.pde

The flower garden program. It produces an endless series of gardens, each one unique.

GROUP 3: ARRAYS

The programmer's name for a list is an *array*. In Processing, you can create a list, or an array, of any type of data: ints, floats, colors, and so on, where all the elements of an array are of the same type. Arrays are a flexible and powerful way to retain information about a lot of similar objects at one time, and manipulate them all in similar ways.

W5 G3 V1 **Arrays** *(5m33s)*

We look at the basic ideas behind arrays and their use.

W5 G3 V2a **Creating Arrays Part 1** *(11m39s)*
W5 G3 V2b **Creating Arrays Part 2** *(10m45s)*
W5 G3 V2c **Creating Arrays Part 3** *(5m59s)*

How to create an array, put values into it, and use it in a program.

circlesArray.pde

Draw 4 circles using arrays.

circlesBounce1.pde

Add motion to the previous sketch, so the balls move horizontally and bounce off the sides of the graphics window.

W5 G3 V3a **Using Multiple Arrays Part 1** *(9m10s)*
W5 G3 V3b **Using Multiple Arrays Part 2** *(9m45s)*

How multiple arrays let us keep track of several pieces of information for every object.

circlesArrayGrow.pde

Use arrays to draw several growing circles.

circlesArrayGrowMove.pde

Use arrays to make circles move as they grow.

circleArrayWithRandomMotion.pde

Use arrays to move circles unpredictably over time.

circlesArrayGrowWithColor.pde

Using arrays to make lots of circles move, grow, and change color.

W5 G3 V4 **Copying Arrays** *(11m13s)*

The technique for copying an array (never just assign one array to another!).

DroppersOnly.pde

A bunch of circles fall down the screen.

DroppersWithShadows.pde

Copy the array of circles at every frame to make a new array. We use the new array to draw drop shadows behind the circles.

W5 G3 V5a **Array Operations Part 1** *(9m3s)*

W5 G3 V5b **Array Operations Part 2** *(6m4s)*

There are a number of built-in routines that help us manipulate arrays of any type. We can add one element to an array with `append()`, add lots of elements with `expand()`, remove the last element with shorten(), or combine two arrays with `concat()` and `splice()`. We can copy one array to another with `arrayCopy()`, and switch the order of every element of the array using `reverse()`. If the array is made up of numbers or letters we can use `sort()`. And we can extract any chunk of an array with `subset()`.

ArrayOperations2.pde

Interactive program for looking at the different array operations on numbers, letters, and colors.

W5 G3 V6a **Modulo Part 1** *(9m54s)*

W5 G3 V6b **Modulo Part 2** *(5m39s)*

The *modulo* operator for two numbers `a` and `b` is written `a%b`. When `a` and `b` are both positive, this tells us what's left when we remove `b` from `a` as many times as we can without making a negative. The modulo operator is used frequently with arrays, because it lets us easily re-use the elements of an array, in order, as many times as we like.

ArrayModulo.pde

See the effect of using modulo on an array of circles.

ArrayModuloVariations.pde

Some variations of using modulo on an array of circles.

PlotMod.pde

Interactive program to plot the modulo operators on the number line.

WheelOfModulo.pde

Interactive demo to illustrate the metaphor of modulo as a wheel of fortune.

W5 G3 V7 **Arrays and Modulo** *(8m28s)*

Using modulo to select items from multiple arrays of different lengths.

ChaserLights.pde

Interactive program to draw a movie marquee with lots of lights changing color around its outside.

BoxOfQuackers.pde
MyShapes.pde

Use arrays of different lengths to hold random colors, and then use them to draw a huge number of different rubber ducks.

W5 G3 V8 **Arrays and Random Numbers** *(9m2s)*

Arrays let us create big collections of objects by using random numbers, and then remember their information over time.

JitterBalls.pde

Random balls jiggling around randomly.

Kaleidoscope4.pde

Random circles move outward from the center, jiggling a little bit randomly as they go. We draw each circle 8 times to make a kaleidoscope.

RainDrops.pde

Create a bunch of random falling raindrops. When one disappears off the bottom of the screen, restart it up top as a new random drop.

W5 G3 V9 **Selecting With Arrays** *(10m52s)*

How to use arrays and the mouse to select and drag around an object.

circleSelector.pde

An interactive demonstration of selecting and dragging an object that is an element of an array.

GROUP 4: TRANSFORMS

We can manipulate the coordinate system by using *transforms*, or *transformations*. When we move, scale, or rotate the coordinate system, then everything we draw from then on will be affected by that transformation (everything that's already been drawn is unaffected). This is a simple but subtle idea that gives us a tremendous amount of power to draw rich imagery, but like any powerful tool, we have to use it carefully.

W5 G4 V1 **Transforms Overview** *(10m32s)*

An overview of our coordinate system, and the bear we'll be using as a running example.

AnimalShapes.pde
BearFromDots.pde
Using the data from a pencil drawing, draw a bear from points and lines.

W5 G4 V2 **Translate** *(10m17s)*

We can *move* the coordinate system around. The programmer'a name for such movement is *translation*.

TranslateExplorer.pde
Button.pde
Op.pde
Picture.pde
TextBox.pde
An interactive program to explore the effects of translation.

W5 G4 V3 **Rotate** *(10m38s)*

We can *rotate* the coordinate system into any orientation.

WalkingBear.pde
AnimalShapes.pde
Using rotation, we can easily create an animation of our bear "walking" around the outside of a circle.

TranslateRotateExplorer.pde
Button.pde
Op.pde
Picture.pde
TextBox.pde
An interactive program to explore the effects of translation and rotation.

W5 G4 V4 **Scale** *(10m49s)*

We can *scale* the coordinate system to any size.

TranslateRotateScaleExplorer.pde

Button.pde

Op.pde

Picture.pde

TextBox.pde

An interactive program to explore the effects of translation, rotation, and scaling.

W5 G4 V5 **Scale and Stroke** *(8m61s)*

When we enlarge the coordinate system, all the strokes that are drawn around objects scale up as well, and when we scale the system down, the strokes get thinner. This is almost never desirable, so we see how to fix the problem.

scaleAndStroke.pde

An interactive program to see that strokes change thickness when we scale the coordinate system, and how scaling the stroke solves the problem.

GROUP 5: USING TRANSFORMS

Transforms are powerful, but they can be tricky. We'll discuss how certain types of scaling and rotation can introduce *skew*, which can make objects appear to be leaning. We'll also look at the *transformation stack*, which provides the key technique for using transformations efficiently.

W5 G5 V1 **Transform Interactions** *(12m4s)*

Certain combinations of scale and rotate can lead to *skew*, or the appearance that an object is leaning over.

SkewDemo1.pde

AnimalShapes.pde

Interactive program to see the effect of skew on three animal shapes.

W5 G5 V2 **Transforms In Action** *(10m58s)*

We see that the order in which we apply transformations can greatly influence the result.

TranslateRotateScaleExplorer.pde

Button.pde

Op.pde

Picture.pde

TextBox.pde

An interactive program to explore the effects of translation, rotation, and scaling.

W5 G5 V3 **Stacks** *(6m34s)*

Illustrating the idea of a *stack* using colors.

W5 G5 V4a **The Transform Stack Part 1** *(9m41s)*

W5 G5 V4b **The Transform Stack Part 2** *(6m6s)*

An introduction to the transformation stack and how to use it.

AnimalMobile.pde

AnimalShapes.pde

Using the transformation stack to draw a moving mobile of several animals.

W5 G5 V5a **The Transform Stack In Action Part 1** *(8m16s)*

W5 G5 V5b **The Transform Stack In Action Part 2** *(6m33s)*

We see how to use the transformation stack to create a variety of animations.

Orbits.pde

A little animated planetary system, where the planets and moons are squares. We use the transformation stack to keep everything spinning and moving properly over time.

AnimalOrbits.pde

AnimalShapes.pde

A little planetary system where the planets have animal shapes.

AnimalPendulum.pde

AnimalShapes.pde

A swinging pendulum of a little “flower” made out of animals.

DroppingLeaves.pde
TreeShapes.pde
A tree that drops an endless series of fluttering, moving leaves.

Marionette.pde
An example of using the transformation stack to create a little marionette. This sketch is provided as a possible starting point for your own projects.

Masks.pde
Use transforms to draw and animate a bunch of simple geometric masks.

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.

W5 G6 V1 **Week 5 Recap and Homework** *(8m52s)*

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.

2D Animation & Interaction Course
Week 6: Curves

WEEK 6: CURVES

Week 6 is all about creating, drawing, and using free-form curves. So far, the only curves we've been able have been circles and ellipses, and pieces of them. Now we extend our range to create big, smooth curves that can take on of any shape we want. Yet they're always under our control, and we can adjust their shape with as much precision as we desire.

Curves can be drawn directly, as colored lines or filled-in shapes. We can also use curves as motion paths, so objects can move along them smoothly over time.

Processing offers us two types of curves. The *Catmull-Rom* curves are simple to create and use, but they're not designed to give us a lot of fine-tuning control over the shape of the curve. So they're usually used when you just want to make a nice curvy shape, but the exact path of the curve isn't important. When you do care about the detailed shape of the curve, then the *Bézier* curve is a better choice. It's a little more complicated to create and use, but it offers us much more control over the shape of the curve.

In addition to these two types of curves, this week we'll also look at a few other geometric tools and ideas that come in very handy when working with curves.

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.

W6 G1 V1 **Week 6 Overview** *(4m27s)*

An overview of both types of curves and some of the many things we can do with them.

GROUP 2: CURVES AND SEGEMENTS

An idea that's common to both types of curves is that we draw them out of *segments*. A segment is a single little bit of a curve that's defined by just four points. The curve only goes through two of these points. The other two serve as controls to influence the shape of the curve.

W6 G2 V1 **Curves and Segments** *(6m54s)*

We look at a single segment of both types of curves, and see how segments combine to make up a larger, smooth curve.

GROUP 3: USEFUL GEOMETRY TOOLS

When we work with curves we'll frequently generate the points that control their shape by using combinations of other points. We'll look at a few nice tools, ideas, and shortcuts that can make this everyday job much easier.

W6 G3 V1 **Two Geometry Tools** *(3m42s)*

An introduction to scaling lines and rotating them 90 degrees, and a first look at our running example of generating an egg shape from a simple scaffold.

EggMaker.pde

Interactively manipulate the scaffold that defines an egg, and draw the resulting curve.

W6 G3 V2 **Scaling Lines** *(6m8s)*

We look at the technique for making a given line longer or shorter.

LineScaler.pde

An interactive demo for understanding at the scaling of lines.

W6 G3 V3 **Rotating A Line 90 Degrees** *(6m46s)*

A beautiful little shortcut for easily rotating any line by 90 degrees.

W6 G3 V4 **Normalizing A Line** *(7m57s)*

How to *normalize* a line, or scale it to length 1.

W6 G3 V5 **Using the Geometry** *(8m13s)*

Using all the tools above to create the scaffolding for an egg.

BulletAvoidance.pde

Starting point for a game where we use the mouse to control a flying saucer and avoid a bullet that's shot out from the center.

GameInfo.pde

Interactive sketch to show on-screen the various numbers that go into controlling the flying saucer in the BulletAvoidance game.

EggMaker.pde

Interactively manipulate the scaffolding for an egg.

PileOfEggs.pde

Randomly generate a whole lot of eggs by using random numbers for some parts of the skeleton.

PileOfEggs2.pde

Randomly generate even more eggs, but this time with different sizes, by using random numbers for some parts of the skeleton.

PileOfWobbledEggs.pde

Randomly generate eggs with unpredictable locations, sizes, and shapes.

Bones.pde

Draw a bunch of random "bone"-shaped objects with curves for outlines and curves inside the shape for decoration.

GROUP 4: CATMULL-ROM CURVES

We look at *Catmull-Rom* (or CR) curves (named for the two men who developed the underlying mathematics). A CR curve segment contains four points: the first and last points influence the shape of the curve, which is drawn between the second and third points. We can draw big, complicated, smooth curves by drawing lots of segments. We can also find out lots of useful information about a curve. One of the most important things we can learn is the location of points along the curve, which lets us use the curve to control the on-screen motion of other objects.

W6 G4 V1 **CR Curves Intro** *(10m24s)*

An introduction to CR curves and what we can do with them.

OneClosedCRCurve.pde

Draw a single, closed CR curve.

OneOpenCRCurve.pde

Draw a single, open CR curve.

MovingStarCR.pde

Use a CR curve to animate a star moving along the curve.

CRCurveExplorer.pde
Knot.pde
Curve.pde

Interactively manipulate a big CR curve, with lots of options.

W6 G4 V2 **Closed CR Curves** *(11m24s)*

A closed curve forms a loop; that is, the ends are not dangling free. We see the technique for drawing smooth, closed CR curves.

OneClosedCRCurve.pde

Draw a single, closed CR curve.

OneOpenCRCurve.pde

Draw a single, open CR curve.

MovingStarCR.pde

Use a CR curve to animate a star moving along the curve.

CRCurveExplorer.pde
Knot.pde
Curve.pde

Interactively manipulate a big CR curve, with lots of options.

W6 G4 V3 **CR Curve Segments** *(11m1s)*

We look more closely at a single CR curve segment, and learn the rule for joining up multiple segments smoothly.

OneCRCurveSegment.pde

Draw a single CR curve segment.

OneCRCurveStrokeFill.pde

Program to explore the difference between filling each CR curve segment individually, and filling the entire curve.

CRCurveExplorer.pde
Knot.pde
Curve.pde

Interactively manipulate a big CR curve, with lots of options.

W6 G4 V4 **CR Curve Utilities** *(10m56s)*

We look at two important utilities that help us work with CR curves: adjusting their *tightness*, and finding points on the curve.

CROneCurvePoint.pde

Interactively choose and display one point along a CR curve.

MultiplePointsOnCurve.pde

Interactively choose many points to display along a CR curve. Notice that the points are not evenly spaced out along the curve.

CRTightCurvePointExplorer.pde

Interactively adjust the tightness of a big closed CR curve, and choosing a point along that curve.

W6 G4 V5 **CR Offset Curves** *(12m14s)*

We can use *tangents* and *normals* to create an *offset curve*: a curve that is roughly parallel to, but some distance away from, a given curve.

TangentAndNormalExplorer.pde

Interactively explore the idea and process of offset curves by drawing a starting curve, then finding its tangents, normals, and offset curves.

GROUP 5: BÉZIER CURVES

We turn now to *Bézier* curves (named for the man who popularized the underlying mathematics). A Bézier curve segment contains four points: the curve is drawn between the first and last points, and the middle two points serve as "handles" that influence the shape of the drawn curve. Using the handles we can produce a wide variety of curves between the first and last points. We can draw big, complicated, smooth curves by drawing lots of segments. We can find points along the curve, and find the tangents and normals that let us build offset curves.

W6 G5 V1 **Bezier Curves Intro** *(8m40s)*

We look at the basics of the Bézier curve.

BezierCurveExplorer.pde
Curve.pde
Knot.pde

An interactive program to explore Bézier curves. You can move the knots and the handles, and customize the presentation with lots of keyboard options.

W6 G5 V2 **Bezier Curve Smoothness** *(9m13s)*

We see the rule for joining up Bézier segments so they form a single larger, smooth curve.

BezierCurveExplorer.pde
Curve.pde
Knot.pde

An interactive program to explore Bézier curves. You can move the knots and the handles, and customize the presentation with lots of keyboard options.

W6 G5 V3 **Bezier Big Curves** *(11m22s)*

How to build big, smooth curves out of lots of Bézier segments.

BezierCurveExplorer.pde
Curve.pde
Knot.pde

An interactive program to explore Bézier curves. You can move the knots and the handles, and customize the presentation with lots of keyboard options.

W6 G5 V4 **Bezier Curve Segments** *(2m12s)*

We take a closer look at the individual segments of Bézier curves.

BezierCurveExplorer.pde
Curve.pde
Knot.pde

An interactive program to explore Bézier curves. You can move the knots and the handles, and customize the presentation with lots of keyboard options.

W6 G5 V5a **Bezier Curve Points Part 1** *(7m51s)*
W6 G5 V5b **Bezier Curve Points Part 2** *(5m30s)*

We can find points along a Bézier curve, or along a big curve made of multiple segments.

BezierCurveExplorer.pde
Curve.pde
Knot.pde

An interactive program to explore Bézier curves. You can move the knots and the handles, and customize the presentation with lots of keyboard options.

BezierUtilExplorer.pde
IO.pde
Knot.pde
Utilities.pde

Interactively explore large Bézier curves made of multiple segments. You can control how the curve is drawn and filled, draw normals and tangents and offset curves, and more.

W6 G5 V6 **Bezier Curve Motion** *(9m40s)*

We can move an object along a path defined by a Bézier curve by finding points along the curve.

Fountain.pde

Draw a two-sided heart fountain using lots of circles that leave trails behind.

MovingBezierStar.pde

Animate a star moving on the screen following the path of a Bézier curve.

BezierCurveExplorer.pde
Curve.pde
Knot.pde

An interactive program to explore Bézier curves. You can move the knots and the handles, and customize the presentation with lots of keyboard options.

W6 G5 V7 **Bezier Offset Curves** *(10m29s)*

By finding the tangents and normals along a curve, we can create new offset curves that run roughly parallel to the original.

BezierUtilExplorer.pde
IO.pde
Knot.pde
Utilities.pde

Interactively explore large Bézier curves made of multiple segments. You can control how the curve is drawn and filled, draw normals and tangents and offset curves, and more.

GROUP 6: ROTATING WITH ATAN2

When you want to rotate something to point at something else, you need to find the angle by which to rotate it. The easiest way to do this is almost always to call the built-in function `atan2()`. The `atan2()` function is useful for both types of curves.

W6 G6 V1 **Rotating with atan2** *(9m39s)*

The idea behind atan2, and a couple of examples that use it.

ArrowField.pde

Interactive program that shows a field of arrows. The arrows spin in place so that they're always pointing at the current location of the mouse.

CarOnTrack.pde

A car runs around a curved track. We can use atan2 to rotate the car as it moves so that the tires look stuck to the track.

W6 G6 V2 **Using atan2** *(10m0s)*

A discussion of the range of angles returned by atan2, and another example of the technique in use.

GooglyEyes.pde
ShapesLib.pde
Interactive program where two animals follow the mouse. Either the whole animal can rotate, or we draw silly googly eyes on the animal and let those follow the mouse. In both cases we use atan2 to find the angle.

GROUP 7: RECAP AND HOMEWORK

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

W6 G7 V1 **Curve Examples** *(4m8s)*

We look at a few examples of what we can do with curves.

ChasingCurve.pde
A curve that changes a little bit on every frame to make a nice accumulating pattern.

FoxAndHare.pde
A rabbit follows a big random curve. A fox chases after the rabbit.

FoxAndTwoHares.pde
There are two rabbits, each following their own random curves. At every moment, the fox chases after whichever rabbit is closer.

MovingStarBezier.pde
Animate a star moving on the screen following the path of a Bézier curve.

MovingStarCR.pde
Animate a star moving on the screen following the path of a CR curve.

Bones.pde
Draw a bunch of random "bone"-shaped objects with curves for outlines and curves inside the shape for decoration.

W6 G7 V2 **Curves Recap** *(11m36s)*

We summarize what we've seen this week. Remember that there's a PDF file with this information as well.

W6 G7 V3 **Week 6 Homework** *(6m47s)*

This week's homework assignment. Remember that there's also a PDF file describing the assignment.

2D Animation & Interaction Course
Week 7: Type, Images, and Patterns

WEEK 7: TYPE, IMAGES, AND PATTERNS

This week we introduce two new graphics objects, and new tools to help us create animation.

We first dig into Processing's system for drawing and manipulating type on the screen. Happily, you can use all of your installed fonts in your Processing sketches.

Then we see how to read in and manipulate images (like photographs) in image formats like jpg and tiff. We'll see how to read and write individual pixels on the screen to change our pictures.

Then we'll look at ways to create repeating patterns of numbers. These are the basis for lots of animation: you tell an object how to move, and it follows that path until you change it. Because the motion is controlled by some kind of function or1 procedure, this is sometimes called *procedural animation*. Starting with simple patterns, we can combine them to make interesting motion.

We'll also encounter two new data types: one for holding strings of characters, and one for conveniently representing both the x and y values of a point in a single variable.

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.

W7 G1 V1 **Week 7 Overview** *(5m17s)*

A survey of some things that we can do with on-screen type, images, and repeating patterns.

GROUP 2: STRINGS

The *String* is a built-in data type for storing character strings. It's much better than making your own array of characters, because it's supported by a bunch of useful, built-in operations.

W7 G2 V1a **Strings Part 1** *(8m2s)*
W7 G2 V1b **Strings Part 2** *(8m20s)*

The basics of Strings, how to use them properly, and eight of the built-in routines for working with them: find the number of characters in the String with `length()`, extract one character with `charAt()` or many with `substring()`,find the first location of some characters with `indexOf()`, compare two Strings with `equals()`,cut away white space at the ends with `trim()`,and make every letter a capital or miniscule with `toUpperCase()` and `toLowerCase()`.

floatToString.pde

We can turn a float into a String.

InputStringToFloat.pde

Type in a float and press Enter to convert it into a float.

stringToFloat.pde

We can turn a String into a float, so we can use it like any other number.

W7 G2 V2 **String Tools** *(6m34s)*

Three useful tools for Strings: `combine()`, `join()`, and `split()`.

GROUP 3: TYPOGRAPHY

Placing type on the screen is great for lots of different kinds of sketches. Processing has its own way of dealing with typefaces and setting type on the screen.

W7 G3 V1 **Typography Essentials** *(11m22s)*

How to create a Processing-style font file, load it, and use it to draw text.

CreatingFontsDemo.pde

Load a font and write a word on the screen.

vlw_demo.pde

An interactive visualization of how type files work and how Processing draws type on the screen.

W7 G3 V2 **Typesetting** *(9m54s)*

How to control type: sizing, alignment, and setting type in a box.

FontSizeExplorer.pde

Interactive sketch to demonstrate the visual effects from scaling type up and down.

FontAlignmentExplorer.pde

Interactively change lots of typesetting options, such as a typesetting box and inter-line spacing, or leading.

W7 G3 V3 **Measuring Type** *(8m12s)*

Finding the various sizes of a typeface, and the width of a specific piece of text.

W7 G3 V4 **Typography In Action** *(5m14s)*

A bunch of examples of using type in sketches.

SpinningLetters.pde
drawing.pde

Letters spin in place, either in unison or at different speeds.

TextBounceScale.pde

A piece of text gets larger and smaller over time. We use the scale transformation to change the text's size.

TextBounceTextSize.pde

A piece of text gets larger and smaller over time. We use the built-in *textSize* routine to change the text's size.

TypeOnCurve.pde

Animate text moving along a random curve.

BigNews.pde

An animation of an old-time spinning newspaper.

BounceType.pde

Bounce around the letters of a word.

FortuneCookie.pde

Draw randomized fortune cookies.

Rainfall.pde

Animate drops of rain falling down the screen.

Rally.pde
Sign.pde

Animation of many random signs moving around during a peaceful protest.

TypeOnEllipse.pde

Move the letters of a word around on an ellipse.

W7 G3 **V5 Why Font Files** *(5m2s)*

A discussion of why Processing uses custom font files for sketches.

vlw_demo.pde

An interactive visualization of how type files work and how Processing draws type on the screen.

GROUP 4: IMAGES

It's great to be able to show photographs and other saved images in our sketches. We see that it's easy to load an image file into our sketch and display it. We then see how to individually read and write pixels of the on-screen image.

W7 G4 V1 **Images** *(7m6s)*

How to load an image and display it, and how to control the appearance of the image with coordinate system transform commands.

showImage1.pde

Load an image and display it in the graphics window.

showImage2.pde

Load an image and display it. The upper-left corner of the image is fixed at the upper-left of the graphics window, but the lower-right corner of the image tracks the mouse location.

showImage3.pde

Use the mouse to control the rotation and size of an image.

showImage4.pde

Use the mouse to control the rotation and size of a whole bunch of images, creating an effect like a mechanical camera shutter.

W7 G4 V2 **Get and Set** *(7m55s)*

How to read and write the colors of individual pixels using `get()` and `set()`.

dripWithGetAndSet.pde

Using get() and set() to make a photo appear to drip, as though left out in the rain.

swapWithGetAndSet.pde

Using get and set to swap neighboring pixels, causing the image to become fuzzier as though seen through frosted glass.

W7 G4 V3 **The pixels Array** *(11m17s)*

Another way to manipulate pixels is by using a special array named `pixels`. It requires a bit more care than `get()` and `set()`, but when reading and/or writing many pixels it can be much faster.

xyIndex.pde

Interactive animation of how pixel (x,y) components correspond to the single index in the pixels array.

W7 G4 V4 **Comparing Pixel Methods** *(10m34s)*

Comparing the speed of get and set versus the `pixels` array.

drawComparison.pde

The code shows how to draw the identical picture using either set(), or the pixels array.

photoWiggle.pde

An interactive demonstration of just a few of the endless different real-time effects that you can write by manipulating pixels.

weirdXY1.pde

An interactive demonstration of pixel-writing speed. In real time you can switch between drawing the entire window using either set() or the pixels array.

GROUP 5: PVECTOR

When working with screen points and locations, we need to maintain both the x and y components of those points. Processing offers a nice, built-in data type called the *PVector* that lets us bundle up both values into a single variable.

W7 G5 V1 **PVector** *(7m11s)*

How to create and use a PVector.

boxWiggle.pde

Store the corners of a box in PVectors, and wiggle them around a little bit randomly from one frame to the next.

PVectorBoxNesting.pde

Save the corners of a box in PVectors, and then randomly change them from one frame to the next to create a picture of many nested boxes (or ellipses, if you prefer).

W7 G5 V2a **PVector Arrays Part 1** *(5m56s)*
W7 G5 V2b **PVector Arrays Part 2** *(9m56s)*

How to declare, allocate, fill in, and use an array of PVectors.

Waves3.pde

Use 3 arrays of PVectors to draw 3 “waves” that change over time.

WavesMany.pde

Use a two-dimensional array of PVectors to draw many “waves” that change over time.

W7 G5 V3 PVector Tools *(8m7s)*

There are many built-in tools for manipulating PVectors. We cover 8 that you’ll use often, and 5 more that are useful if you do fancy geometry.

GROUP 6: REPEATING PATTERNS

Patterns of numbers are an important building block for creating animation. We can use some patterns immediately for simple animation, and we can combine patterns to make more complicated sequences for more subtle or interesting animation. You don’t have to be a numbers whiz to use these patterns; it’s enough to get a feeling for them and then just explore, trying out different ways to mix them together.

W7 G6 **V1 Creating Repeating Patterns** *(10m32s)*

We look at the basic ideas behind making a repeating number pattern.

Grapher1.pde
VizSet1.pde
VizSet2.pde

An interactive program for visualizing several different patterns and how they look when we use them to animate a variety of different objects.

W7 G6 V2a **Five Repeating Patterns Part 1** *(7m39s)*
W7 G6 V2b **Five Repeating Patterns Part 2** *(5m49s)*

We look at five common and useful types of repeating patterns, which you can use as building blocks to make new patterns.

Grapher1.pde
VizSet1.pde
VizSet2.pde

An interactive program for visualizing several different patterns and how they look when we use them to animate a variety of different objects.

GROUP 7: RECAP AND HOMEWORK

We look over what we've seen this week, and then you get to put it into action!

W7 G7 V1 **Week 7 Recap** *(10m40s)*

A review of Strings, typography, images, PVectors, and making and using repeating patterns. Don't forget there's also a PDF with this material.

W7 G7 V2 **Week 7 Homework** *(2m1s)*

This week's homework! Remember that there's also a PDF with the homework assignment.

2D Animation & Interaction Course
Week 8: Objects

WEEK 8: OBJECTS

This week we'll look at one big topic, and a bunch of smaller, but still useful, ideas.

We'll start by reviewing how to break up a big program into several files. Processing makes this easy on us by offering tabs in the text editor window.

Then we'll dig into our big topic: *objects*. To programmers, the word "object" is often used casually as we do in everyday life, but sometimes it refers to a very specific idea, generally part of something called *object-oriented programming*. This is a very powerful way to think about and write programs that contain lots of similar, and potentially interacting, objects. The essence is that we can create our own custom types of variables. To do so, we provide a list of variables that describe the object, and a bunch of functions that the object will be able to execute. Then we can send *messages* to these objects, and they will take actions in response. The idea is big and rich; here, we'll cover the basics that will give you a good start in making this technique part of your toolkit.

Then we'll look at a few things that we haven't discussed yet: how to draw graphics into off-screen memory (so you can easily re-use them), finding the time and date, reading and writing text files, and a brief look at Processing's 3D abilities.

We'll wrap up by looking at how you can share your work with other people, and look at to some ways you can expand your range and explore new kinds of applications.

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.

W8 G1 V1 **Week 8 Overview** *(9m8s)*

A survey of projects, object-oriented programming, off-screen drawing, time and date, disk files, 3D, sharing your work, and how to stretch your already considerable skills.

GROUP 2: BIG PROJECTS

When we write a big project, it's often handy to break it up into multiple source files. This technique will be particularly useful when we write our own objects.

W8 G2 V1 **Big Projects** *(6m38s)*

How to organize a big project using Processing's built-in tools.

GROUP 3: OBJECT-ORIENTED PROGRAMMING

The principles of *object-oriented programming* let us organize and write our programs in a new and powerful way. There's nothing you can do with these techniques that you can't already do, but they offer a huge conceptual advantage when your programs get big.

W8 G3 V1a **Objects Overview Part 1** *(6m52s)*
W8 G3 V1b **Objects Overview Part 2** *(8m46s)*

An introduction to concepts and techniques of object-oriented programming.

Traffic1.pde
Car.pde
Interactive demonstration of a custom class that holds a single cartoon car. Cars respond to a variety of messages. You can select a car with the mouse, and then use the keyboard to send it messages to change the car's speed and direction.

W8 G3 V2 **Creating Objects** *(9m23s)*

We discuss the idea of a *constructor* and how to use it to create a new instance of a class.

W8 G3 V3 **Implementing Objects** *(8m7s)*

We flesh out the Car class from the previous examples.

W8 G3 V4 **Encapsulation** *(10m57s)*

The idea of *encapsulation* encourages us to design our objects so that they are completely responsible for their own descriptions and behaviors.

Traffic2.pde

Car.pde

We modify the Car class from previous examples by replacing the cartoon drawing with photographs. The main program doesn't change at all; everything happens inside of the class itself.

W8 G3 V5 **Arrays of Objects** *(12m25s)*

How to create and work with arrays of objects built from a custom class.

ArrowClassField.pde

Arrow.pde

We draw a field of arrows, and a yellow dot controlled by the mouse. Each arrow spins to point to the mouse. Big arrows spin more slowly than small ones. This program is a re-implementation of an earlier program, but here we use an array of instances of a custom Arrow class.

ArrowClassFieldDistance.pde

Arrow.pde

A modification to the arrow sketch. Here, the speed of each arrow's totation is determined by its distance from the mouse. All the changes are in the Arrow class, and the rest of the program is completely unaffected.

ArrowHandClassField.pde

Arrow.pde

A modification to the arrow sketch. An Arrow object draws itself using a frame from a short animation of a hand that goes from a fist to a pointing finger.

W8 G3 V6a **Subclasses Part 1** *(9m51s)*

W8 G3 V6b **Subclasses Part 2** *(8m14s)*

We can extend a class by building a *subclass*. This inherits all the variables and methods of the starting class (called the *parent class*), but then adds additional variables and methods.

CarAndAeroCarSkeleton.pde

A minimal "skeleton" program showing how to define and use a subclass.

MovingCar.pde
MovingCarSubclass.pde
Car.pde
An interactive demo of a MovingCar, which extends our Car class by adding a moving box on top to help our friends move to a new place.

W8 G3 V7 **Drawing With Subclasses** *(8m41s)*

Many classes start their drawing process by calling some transformations, so subclasses need to call those transformations as well. We see see a nice way to keep all the transformations in one place, which is not only efficient, but it lets subclasses automatically inherit any transforms we apply to the parent class.

CarAndAeroCar.pde
AeroCar.pde
Car.pde
A very simple extension of the Car class that adds a wing to the car. We use the transforms technique from the video.

MovingCar.pde
Car.pde
MovingCarSubclass.pde
We improve our previous MovingCar example by using the new transformation technique for both the parent and child classes.

W8 G3 V8 **Subclasses and Variables** *(8m35s)*

How child objects can refer to variables (and methods) in their parent class.

MovingCarSubclass.pde
Car.pde
MovingCar.pde
A Car class and a subclass with a moving box on top.

W8 G3 V9a **Subclass Example Part 1** *(10m50s)*
W8 G3 V9b **Subclass Example Part 2** *(12m51s)*

We illustrate the ideas of classes and subclasses by creating an aquarium populated by three different types of fish. The different varieties of fish are all subclasses of the same parent object.

Aquarium.pde
Fish.pde
FishTank.pde
HungryFish.pde
LightFish.pde
SleepyFish.pde

This aquarium program shows how objects can help us structure our code cleanly. The main file, Aquarium.pde, only knows about one object: an instance of type FishTank. The tank, in turn, is populated by lots of objects that are subclasses of type Fish. Everything in the system takes care of itself, and the details of each type of object are isolated to that object's class.

GROUP 4: ODDS AND ENDS

In this section we cover three topics that didn't have a natural fit in the previous weeks. Off-screen drawing lets us draw images once and re-use them, potentially saving us a lot of time, and thus speeding up our programs. We'll also see how to get the current time and date, and how to read and write text files.

W8 G4 V1a **Offscreen Drawing Part 1** *(9m52s)*
W8 G4 V1b **Offscreen Drawing Part 2** *(8m50s)*

How to create, draw into, and display an off-screen buffer.

DrawWithOffscreen.pde

A simple interactive program to create, draw into, and display an off-screen buffer, optionally wedged between two ellipses.

SpinFuzz.pde

A demonstration of how an off-screen buffer can speed up a program dramatically. We create an off-screen image and then copy it to the screen with transformation commands, only re-creating the image when you press a keyboard key to change the picture.

StencilWithHoles.pde

An interactive demonstration of using an off-screen buffer to create graphics with "holes," or transparent regions.

W8 G4 V2 **Time and Date** *(10m48s)*

How to get the current time and date, and use them in a sketch.

simpleClock.pde

A simple clock with hour, minute, and second hands.

diskClock.pde

A non-traditional clock where the hands stay still, and three disks rotate to show us the time.

hourDotClock.pde
Dot.pde

An unusual clock that marks the passage of a single hour. It starts as a nice, orderly grid of circles. Each second, another circle breaks free of the grid and floats down the screen to a new, random position.

yearClock.pde

A combined clock and calendar that shows the current moment on a curve that runs from January 1 to December 31.

W8 G4 V3a **Disk Files Part 1** *(12m0s)*
W8 G4 V3b **Disk Files Part 2** *(10m5s)*

How to read, write, and use the contents of text files.

drawFileXY.pde

Draw a shape by reading points from a disk file.

fileReading.pde

Read a text file and print out each line, followed by each word of that line.

fileWriting.pde

Write the X and Y positions of the mouse to a text file each time you press the space key.

GROUP 5: OVERVIEW OF 3D

Processing offers some basic tools for creating and displaying shapes in 3D. This subject is very big, and we consider it an advanced technique. This video, and the examples, give you a taste of what you can do, and some starting points if you want to dig into it yourself.

W8 G5 V1 **Overview of 3D** *(6m35s)*

A survey of some things you can do in 3D with Processing.

XYZrotateBoxes.pde

Draw four boxes in 3D to show the coordinate system. The origin is in gray, and the red, green, and blue boxes are along the X, Y, and Z axes respectively. Move the mouse to tumble the system around in 3D.

necklace10.pde

Draw a "necklace" consisting of a central sphere surrounded by a ring of "beads", each of which is a rectangle. In this version, each rectangle rotates around its own center.

necklace11.pde

In this version of the “necklace,” each rectangle rotates around its own center.

necklace12.pde

In this version of the “necklace,” each rectangle rotates around its own center. We add a little bit of extra twist so that the whole ring is twisted one full turn around the central sphere.

necklace13.pde

In this version of the “necklace,” each rectangle rotates around its own center. We add a little bit of extra twist so that the whole ring is twisted two full turns around the central sphere.

necklace14.pde

In this version of the “necklace,” each rectangle rotates around its own center. We add a little bit of extra twist so that the whole ring is twisted one full turn around the central sphere, and then just for fun we follow that up with a rotation around Y that increases over time.

necklace15.pde

In this version of the “necklace,” each rectangle rotates around its own center. We add a little bit of extra twist so that the whole ring is twisted one full turn around the central sphere, and then just for fun we follow that up with a rotation around Y that increases over time, and then a little bit around X as well.

withTexture.pde

In this version of the “necklace,” each rectangle is drawn with a photograph pasted onto its surface.

greeble07.pde
Greeble.pde

Draw "greebles" on a flat sheet. Greebles are little collections of shapes that model-makers place on the outside of things like spaceships to give them a complicated-looking surface.

meshBuilder.pde

Draw one of several different meshes in 3D. You can choose different options for how it's drawn, and you can animate the lights and camera.

GROUP 6: SHARING YOUR WORK

We cover some of the popular ways to share your Processing programs with other people.

W8 G6 V1 **Sharing Your Sketch** *(10m46s)*

You can share your work with other people in lots of ways. Here we cover how to save it as a stand-alone application program, how to include it in a web page, and we see a system that will even run your work on smart phones and tablets.

yellowbox.pde

A simple interactive program. If the yellow box is moving, click in it to stop it. If it’s not moving, click in it to start it.

GROUP 7: GOING FURTHER

We've covered a huge amount of information in this course! If you're thirsty for more, here are some ideas for your next steps.

W8 G7 V1 **Going Further** *(8m17s)*

We discuss how to expand your power using one or more of Processing's many user-contributed libraries. We also talk about running your program on one of today's inexpensive, stand-alone computer boards like an Arduino, letting you take your program out into the real world and use it to control real devices.

GROUP 8: RECAP AND HOMEWORK

We review this week's material, and see the new assignment.

W8 G8 V1 **Week 8 Recap** *(9m25s)*

We survey what we've learned this week, starting with object-oriented programming and moving on to other topics like off-screen drawing, finding and using the time and date, and reading and writing disk files. Don't forget there's a PDF available with this information.

W8 G8 V2 **Week 8 Homework** *(2m44s)*

In this assignment you get to put together everything we've seen in this course. Remember that the assignment is also summarized in a PDF file.