Getting Started With Processing

It is important to know a little about what Processing actually is before we start. First, Processing is not actually a “programming language,” it’s just a fancy library for the Java language. So, when we are talking about programming in processing, it’s really programming in Java with some extra tools already given to you. Second, Processing is also an IDE (Integrated Development Environment) which basically in this case is just a decent text editor and a compiler. For future reference, a compiler converts your text into something that a computer can read and run, but right now that is irrelevant. If you haven’t already downloaded processing, go to processing.org and download the version supported by your operating system.

Structure

Programming is doing whatever kind of math that is necessary to solve your problem. One of the fundamental rules that makes doing math the way we know it possible is **order of operations.** This is the time where we just start manipulating this shit in code. Open up the file **Learning Processing\_1\_1** in the Processing IDE. This file goes along with the more detailed explanation in this document. You can either create a new file, or just modify the one given.

The setup() function

First, what is a function? Basically, a function is the same thing you learned about in algebra classes. You have some collection of math operations that when you plug some numbers into it, you get some result out of it, usually a number. The main difference when talking about functions in programming vs functions in algebra is that your function doesn’t have to return a number at all. Your function could just do a bunch of things and never return a value. Notice in the file the word **void** is in front of the word setup(). **void** signifies that there will be no value returned from the function, but that function can still call on other functions that might return a value. In this case, this setup function is required by Processing for various reasons that we will get into later. Every sketch using Processing’s libraries will require this setup() function, so anytime you ever use processing, you’ll have to apply the setup() function.

The syntax looks like this…

void setup() {

//place your shit in here;

}

Void – represents that the function does not return a value

Setup – is the name of the function

() – is a place where you could put numbers into the function to change the result of the function. The programming name for these numbers that get plugged into a function is an **argument**.

{} – is where all of the things your function is going to do for you will go.

; - the semicolon tells the computer that you are done with giving it a particular task to do. The semicolon goes after every separate line of code, but not at the end of a function. That will get a more accurate explanation down the line.

Now, what do we put in this setup() function? We need to do one more thing to appease the Processing environment. Notice the **size(500,500)** function we called first in the setup() function. This function tells processing what size to make the window you want to work in, in pixels. It takes two arguments, **width** of the desired window, and **height** of the desired window. These numbers must be formatted as **integers** (no decimal places), because we can’t really make a fraction of a pixel.

Now we technically have everything we need to start a Processing sketch. If we stop here and press the **“Run”** button on the top left part of the screen a processing sketch will start, and you’ll get a window that is gray of the size you specified in the size function, 500 pixels wide and 500 pixels tall. The code would look something like this…

void setup(){

size(500,500);

}

Hey, we were able to type something into a text editor, and the computer did a thing. Holy shit. As you might imagine, we are going to need some more tools to start doing things we actually want to do here. A good next step to take is to learn how to draw a primitive shape on the screen. Processing has these super easy functions for drawing ellipses, rectangles, dots, and any custom shape you want on the screen. Let’s start with the ellipse tool.

The ellipse tools is just a premade function given to you with the Processing library. This function takes 4 arguments, and draws an ellipse wherever you tell it to be on the screen.

ellipse( X, Y, size X, size Y)

X – location on the screen on the X- axis

Y – location on the screen on the Y – axis

Size X – wide your ellipse will be

Size Y – how tall your ellipse will be

In the case of our program here, if we were to add this function with some arguments to the setup() function after we call the size function we can draw an ellipse on the screen too! That would look like this.

void setup(){

size(500,500);

ellipse(250,250,20,20);

}

Lets break this down in words for moment

* Declare a setup() function and give it no arguments.
* Within this setup() function we are creating a window 500 pixels wide, and 500 pixels tall.
* Now we draw an ellipse on the screen with these attributes
  + 250 pixels across the x-axis
  + 250 pixels across the y-axis
  + 20 pixels wide with respect to the x-axis
  + 20 pixels wide with respect to the y-axis
* Close scope of the setup() function with a closed bracket “}”

If you were to run this in the Processing IDE you would see a window with a gray background, and a small, white ellipse in the center of the screen. Now try changing some of the numbers you put into that ellipse function.

Notice, you can manually change the location and size / shape of this ellipse. This is not so useful right now, but we’ll get into ways of automatically changing the location and size of the ellipse shortly.

For now, lets look at two more basic functions that Processing offers.

What if we don’t want the background of our window to be such a shitty gray color? Luckily, we have a super easy function for that. It’s called the background function. The background function looks at every pixel in the window you have created and sets it to a given color.

The function looks like this

background(color value);

For now, if you pass 0 into the function for the color value, it’ll set every pixel to black, and if you pass 255 into the background function it’ll set every pixel to white. Any value between 0 and 255 will be a shade of gray.

If we add this to our code it’ll look something like this.

void setup(){

size(500, 500);

background(0);

ellipse(250,250,20,20);

}

Now this code produces the same white ellipse with a black background behind it. This is where order of operations comes into play. Let’s say we decided to switch where we put the background function.

void setup(){

size(500, 500);

ellipse(250,250,20,20);

background(0);

}

If you were to run this code in processing all you would see is a black background! Take a moment and consider why that might be. You called the ellipse function, right?

Processing recognizes that you drew an ellipse on the screen, but it also followed your orders in coloring every pixel on the screen black **AFTER** you drew that ellipse. So, no you wont see that ellipse. This is something to always consider when writing programs, order of operations is everything. You need to keep track of your order of operations as well as the programming language’s order of operations.

Let’s cover one last thing in this intro to Processing. What if you wanted to change the color of that ellipse? Fortunately, we have a function for that too! It’s called **fill().** The fill() function will change the color of the inside of any shape you draw to the screen in processing. The fill() function takes one argument.

fill(color / shade);

So, if we wanted to change our ellipse to gray, we would add the fill function to our code.

void setup(){

size(500, 500);

background(0);

fill(127);

ellipse(250,250,20,20);

}

Note: We don’t have to use the fill function right before we draw our ellipse! Our code above would come up the exact same result on screen as if we did this.

void setup(){

size(500, 500);

fill(127);

background(0);

ellipse(250,250,20,20);

}

The fill function just loads a color to apply to any primitive we might draw on screen. So if we drew another ellipse on screen after drawing our first ellipse, we don’t need to call the fill() function again, because it will take our last designated fill color and apply it to our new ellipse. If we wanted to make our ellipses different colors, then we would have to call the fill() function again, like this.

void setup(){

size(500, 500);

background(0);

fill(255);

ellipse(200,250,20,20);

fill(127);

ellipse(300,250,20,20);

}

For now, try and play with all these things. Punch some numbers into the fill(), ellipse(), background(), and size() functions to get an intuition for what each one actually means on screen.