­Lesson\_1\_3 Variables

So we last learned about how to utilize the mouseX and mouseY environment variables, but now we are going to create our own variables. First, lets talk about how a computer stores data.

If someone asked you to count to ten, you would say, 1, 2, 3, 4,5 … 10. We count in what is called base 10. Every time we reach an increment of 10 we start over counting again. 11, 12, 13… etc. This is not how a computer counts. Every piece of hardware in a computer contains a massive number of transistors to execute various tasks. We use transistors to put data in long term storage (SSD, HDD is a different process), your processor uses transistors to count and do math operations such as addition and multiplication, we use transistors to send information between different pieces of hardware within the computer itself.

What we know about transistors is that they only have two possible states, on and off. So, if you want to store data on a transistor you have to convert it to information it can understand. We call this type of counting base two, or binary. Every single thing on your screen right now is stored as a huge collection of 0s (off) and 1s (on). I know, you’ve definitely heard that before, but we are going to talk about it in a little more depth here because every mathematical operation that your computer will ever execute will be converted to binary before it is calculated.

If everything is binary, how does it know the difference between a word document and a dick pic? The short answer is, we tell it what kind of data it is! For large files, we use a file extension to designate the difference between file types. A JPEG image has the extension of .jpg and a Microsoft word document has a .docx file extension. Now the computer knows how to read the giant binary file you have saved.

This works the same for smaller data types, too. A computer stores a character such as ‘a’ and an integer in binary, but we have to tell it what kind of data it is. When we are working with programming, we will spend a lot of time working with different data types.

Some data types we will see when working with Processing / Java include…

* Integer (0, 1, 2, 3, -4, -23… etc.)
* Floating point (1.24432, -2.45342, 23.3375, -55.2243… etc.)
* Char (‘a’, ‘g’, ‘r’ ‘~’, ‘ ‘)
* String (‘Some Words’, ‘Some Other Words’)
* Boolean (true, false)

Some languages require you to explicitly declare the data type you are working with, some languages store everything as just a variable, and some languages, like Java, are somewhere in-between. If, you’re wondering why some languages do this, and some languages don’t, it’s a matter of efficiency. When we start writing programs that utilize all of your computing power, you are going to want to make your program as efficient as possible, maybe change some floating point numbers to integers, because integers take a smaller number of operations to compute. Eventually, we will discuss this more, but now is the time to just start doin’ this thang.

Side note: The other thing that is different about how computers count is that they do not start at 1, but rather 0. This can be difficult to get used to.

The first data type we are going to work with is an integer. In case you don’t remember, and integer is defined by a number that is not broken down into a fraction. It can be a 1 or a 2, but nothing in between. When we want to create (or **declare**) a variable, we have to tell the computer that we intend to make a variable. The computer will then create a place in memory to store the variable. We can do this by simply typing…

int myIntegerVariable;

When the computer reads this line of text, it allots a space in our RAM (Random Access Memory) to store the variable. But what the computer doesn’t know is what the value the variable is. In fact, there is no data stored here at all. The command we typed in here has only told the computer that we will eventually be associating some data with a given name (in this case this data slot is called “myIntegerVariable”). Processing actually won’t even let us try and use this variable, because it isn’t initialized yet. Initializing is something we will have to do to every variable in Java. This is one of those simple things that changes between programming languages, that can account for some differences efficiency between languages.

If we want to declare (what we just did) and initialize a variable, we can either do this in two lines

Int myIntegerVariable;

myIntegerVariable = 5;

Or we can do it in a single line

Int myIntegerVariable = 5;

For now, it’s much easier to read if we just do it all in one line, but there will be times in the future when it’ll make sense to do it at different times.

Okay, so now we can store data for our programs. What do we do with it? Lets take a quick look at a few examples using an integer variable.

Int x = 250;

Int y = 250;

void setup(){

size(500,500);

}

Void draw(){

Background(0);

Fill(255);

Ellipse(x, y, 20, 20);

}

* First, declare the variable ‘x’ and initialize it with a value of 250
* Next, declare the variable ‘y’ and initialize it with a value of 250
* Make our setup function and set our window size 500 x 500 pixels
* Declare our draw function, so every time draw runs…
  + Set all the pixels to black
  + Set a fill color for a shape to white (255)
  + Draw an ellipse with (here’s where it get’s interesting)
    - Location x with a value of ‘x’
    - Location y with a value of ‘y’
    - Diameter x of 20 pixels
    - Diameter y of 20 pixels

If you plug that code into processing, you’ll see that it looks exactly the same as the first chapter! Yayy! Actually, this is novel. Big-fuckin’-stuff. The value of x and y no longer have to stay at 250. In fact, we now can tell the computer to perform literally any mathematical operation on either of these variables. Let’s do this again, but a little bit different.

Int x = 250;

Int y = 250;

void setup(){

size(500,500);

}

Void draw(){

Background(0);

Fill(255);

Ellipse(x, y, 20, 20);

x = x + 1; // this line

}

So, hopefully you plugged this code into Processing, out of pure curiosity, and saw that the circle lazily moves off the screen! To the right! Why is this happening?

We know that the draw function will happen by default at 60 frames per second. The only line of code that was added to this sketch was the one marked “this line.” Essentially, it’s a counter. Every time the draw function runs, we are telling it to draw the background and the ellipse, but after that we said, set the variable ‘x’ equal to ‘x’ + 1. 60 times per second! And there are only 500 pixels on the screen width-wise, so it eventually makes it’s way off the screen.

You can put a variable in just about every place in a program, but there are things to think about when we start using variables in our programs.

First, and again, order of operations. You can’t use a variable that doesn’t exist yet.

Obviously, this won’t fly.

x = x+1;

int x = 0;

Second, how the computer handles variables. There is this thing called variable scope. Basically, it’s about when the computer creates and destroys each variable. We’ll leave this lesson with one last example to show the idea of variable scope.

Void setup(){

Size(500,500);

Int x = 250;

}

Void draw(){

Ellipse(x, 250, 20, 20);

}

And it gives us an error message! Why does it give us an error message? If you declare a variable inside a function, other functions can’t use that variable! Every time you run a function with a variable declared inside of it, the computer creates the variable, initializes it, and then destroys it after it’s done with the function.

I lied. One more example.

Void setup(){

Size(500,500);

}

Void draw(){

Int x = 250;

Background(0);

Ellipse(x, 250, 20, 20);

X = x+ 1;

}

And the ball never moves! The ball never moves because Java is creating the variable x every time it runs the draw function again! Even though we add 1 to the variable x every time, x gets destroyed and it’s space in RAM is freed up for any other function to use. Then it gets created again at the beginning of draw and used and destroyed again.

Pretty cool…