25.09.18 //

INTRODUCTION TO PROCESSING

DART 631
RILLA KHALED

HELLO WORLD.

- Download Processing.
- Launch Processing.
- Type exactly this into the text entry area

```
println(“Hello, world!”);
```

- Press the play button at the top left of the window.
- Look in the black area at the bottom of the window.

All of the code for those applications was typed into a window like this one.

A little hard to believe, but that’s the power of Processing, and programming in general.

Let’s go and look at the interface

WHY PROCESSING?

It was specifically made for creative people to learn to program in.

It's a proper programming language, built on top of Java.

It's free!

And open source and has some great libraries and tutorials and textbooks and has an amazing community!

It's particularly good for writing unusual and interesting programs in.

Important that Processing **doesn't push us toward particular ideas.**

All we have is the **expressiveness of code** and our imaginations. Perfect.

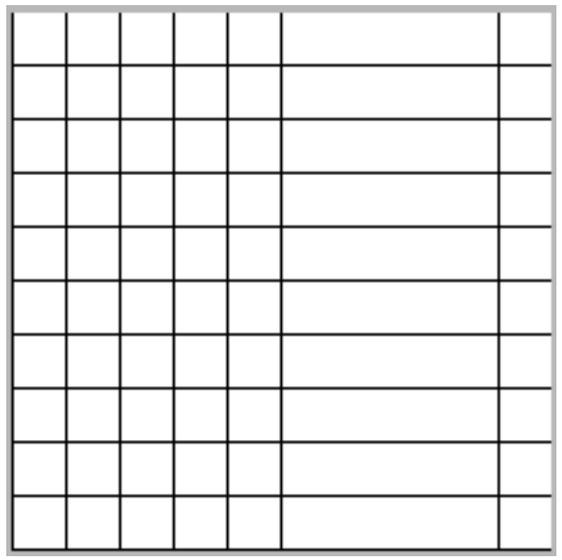
MINI EXERCISE.

1/ Set your Processing sketchbook folder to a sensible location (maybe named after this course) and save your Hello World sketch as HelloWorld into it. Quit Processing and then start it again and load the HelloWorld sketch.

2/ Load up some of the Examples from the File menu and play around with them. Dare to change a number in the code and see what happens!

3/ Go to http://www.processing.org/exhibition/ or http://www.openprocessing.org/ and download and run some of the code you find interesting.

THE GRID.



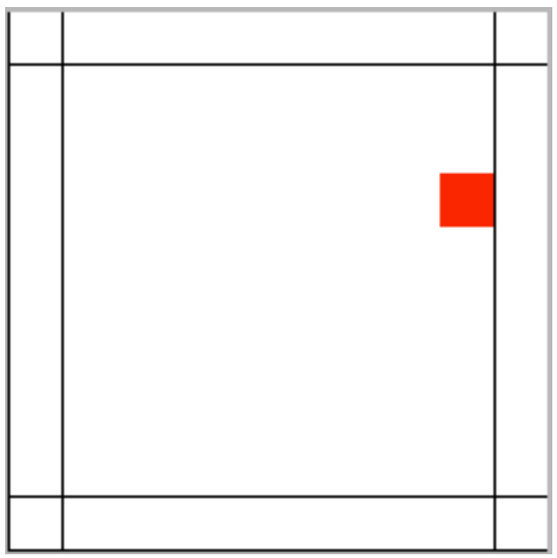
Actually it’s more like this.

When we display things on a computer screen, it’s made of tiny squares called pixels.

“Pixel” is actually short for “picture element”, which basically makes sense.

You display something by changing the colours of the different pixels.

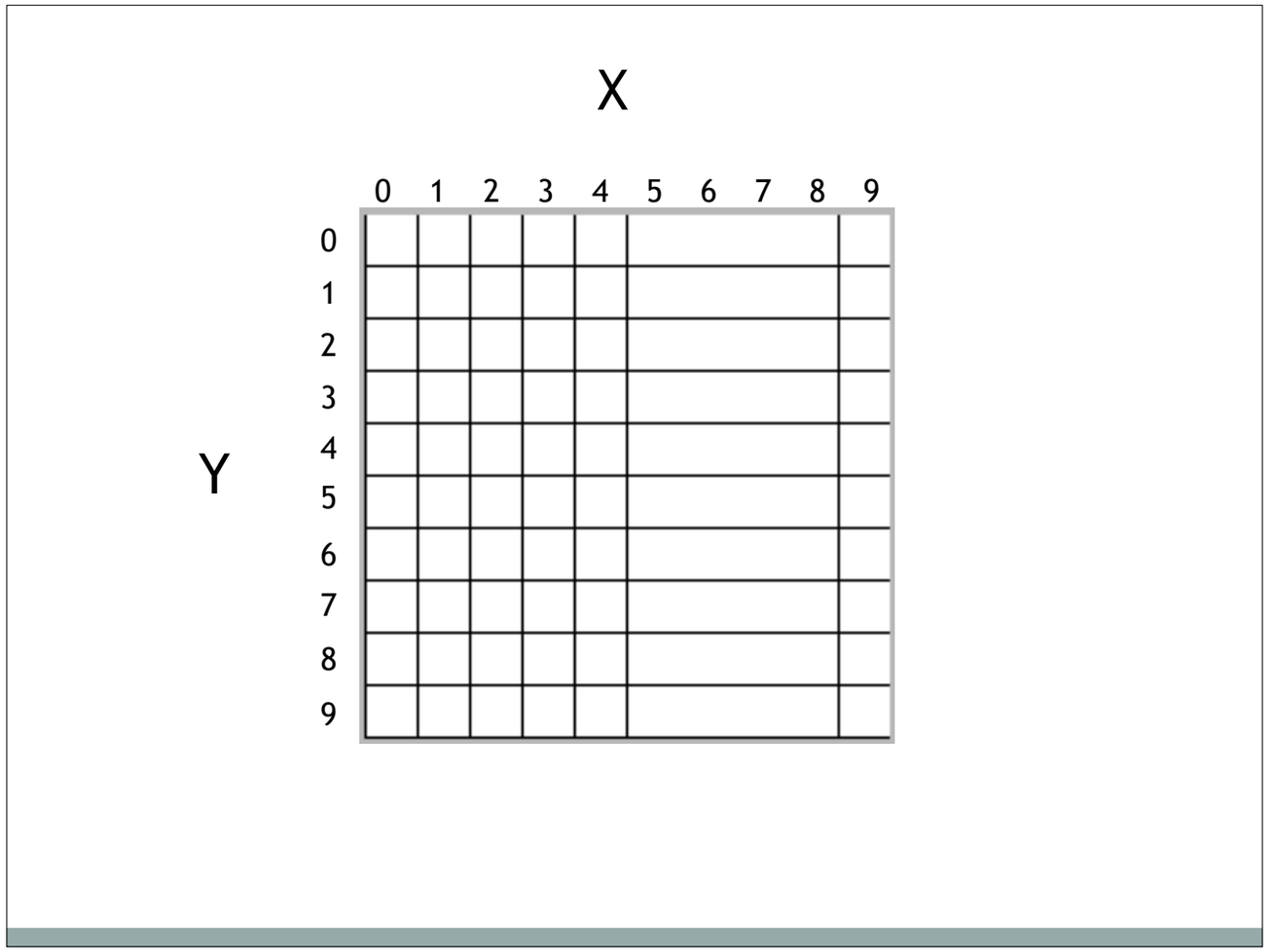
You can even zoom in on your computer screen and see them!



If we’re using a graphics program it’s easy to draw a dot where we want to because we just click the mouse there

But if we’re using a programming language we have to **tell it** to draw a dot

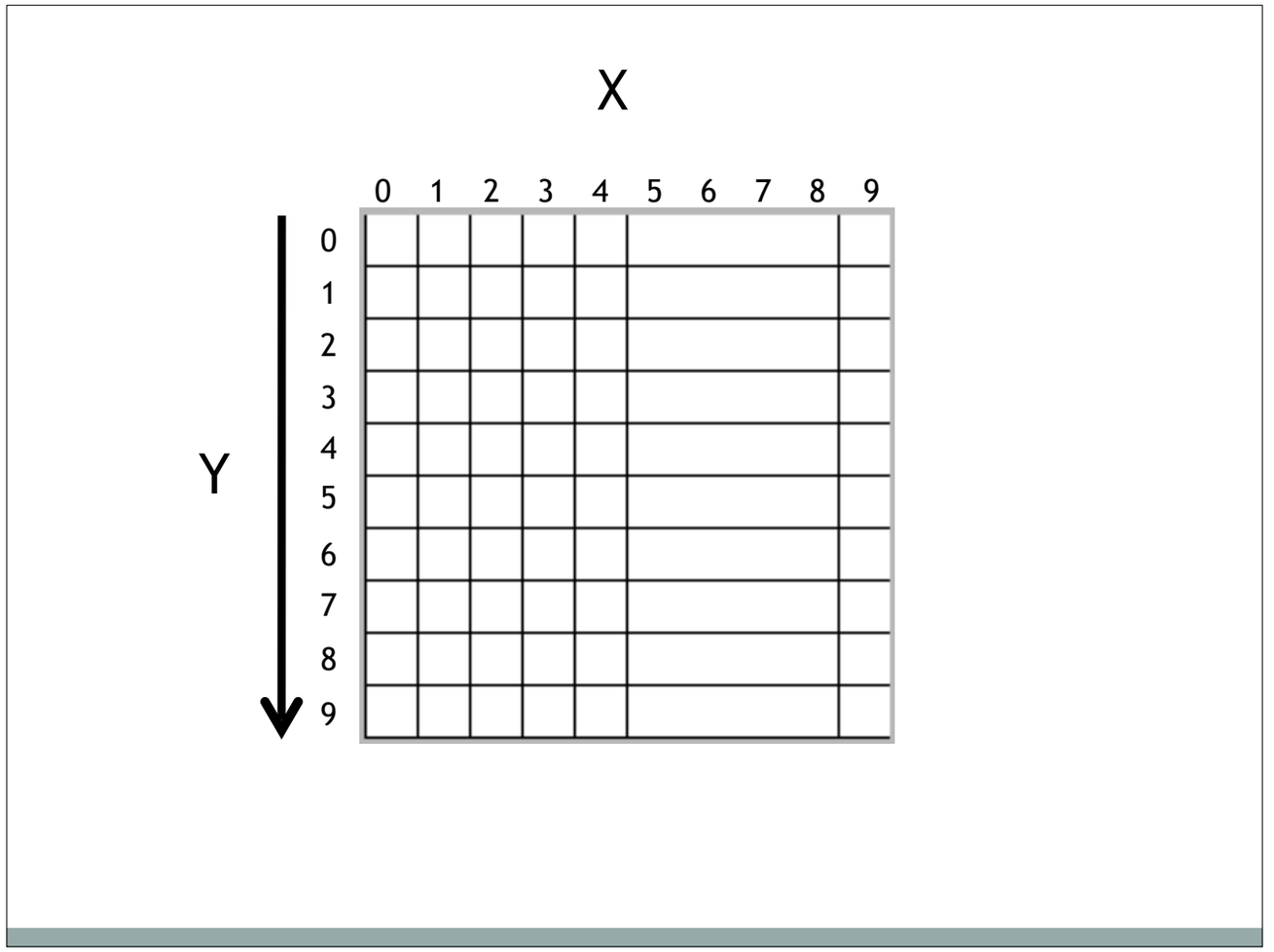
So how do you draw something like a dot or a line?



By naming the pixels with numbers.

Enter the coordinate system!

Pixels are numbered along the X and Y axes from 0 up, starting at the top left corner



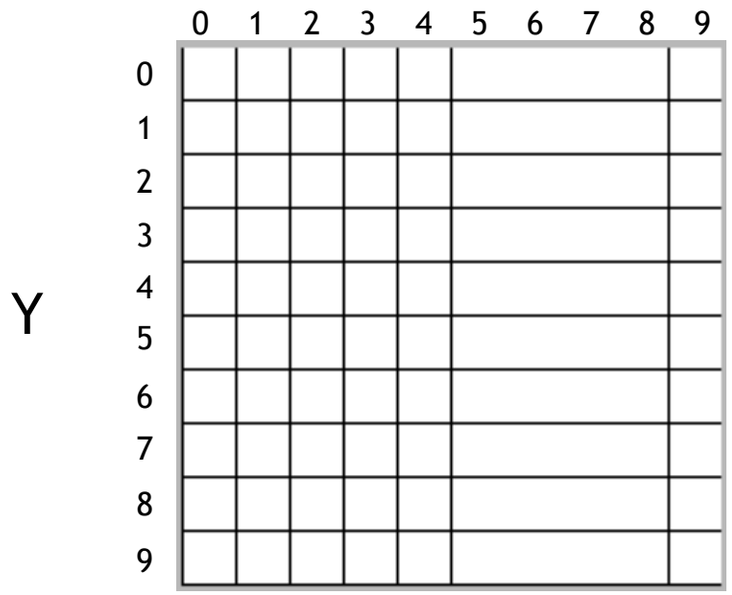
Note that the Y numbers go from the top down to the bottom, unlike in a graph

(X,Y)

We write coordinates like this

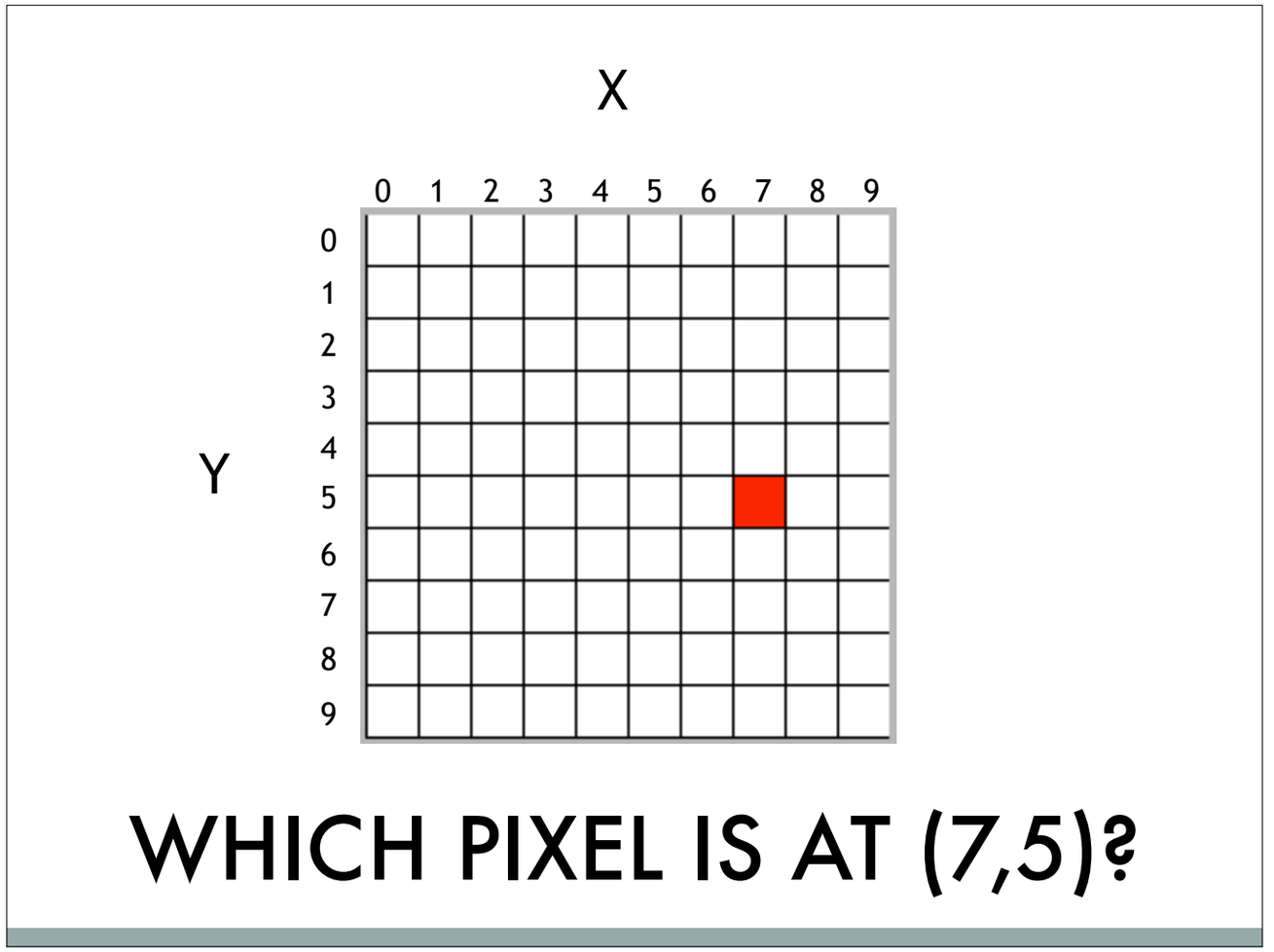
The X coordinate first, then the Y coordinate

X



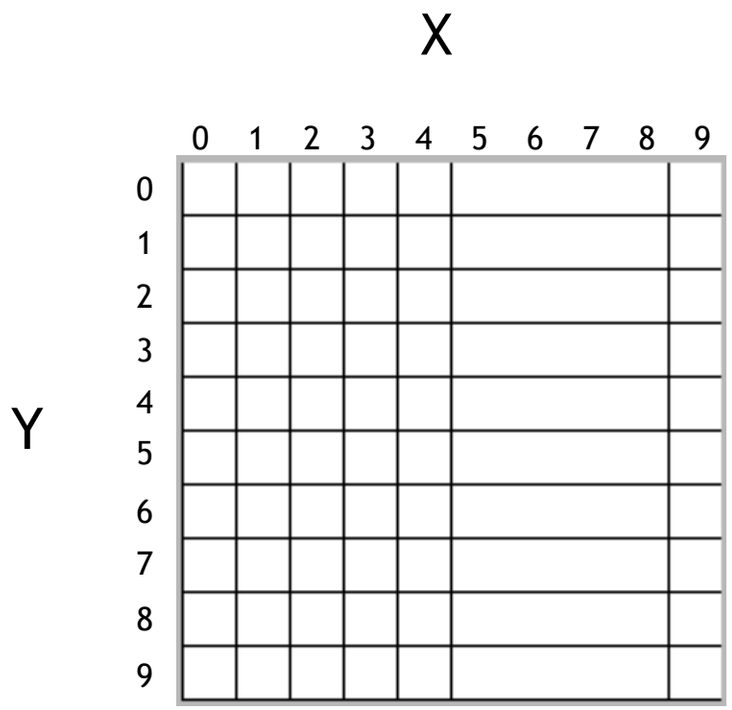
Y

WHICH PIXEL IS AT (7,5)?

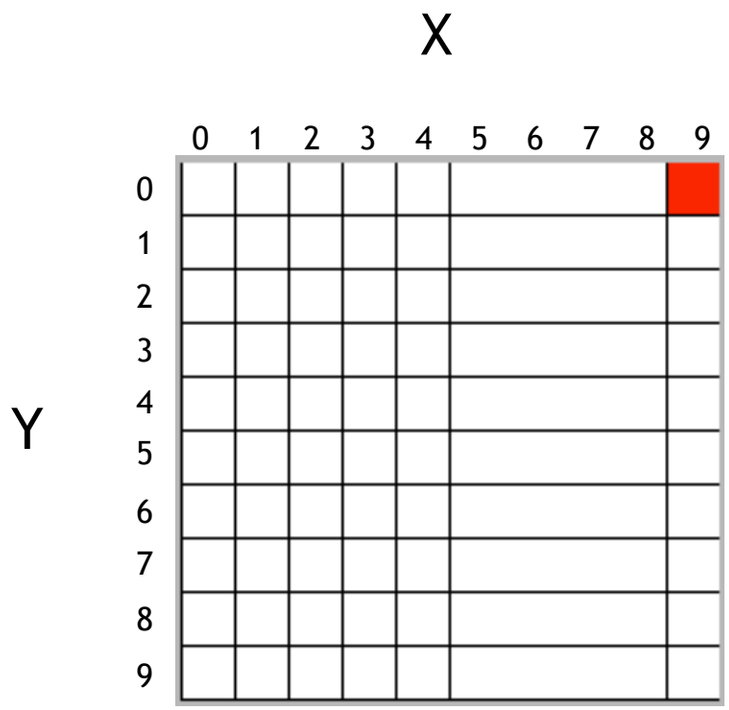


Computers are weird.

We start counting at 0.

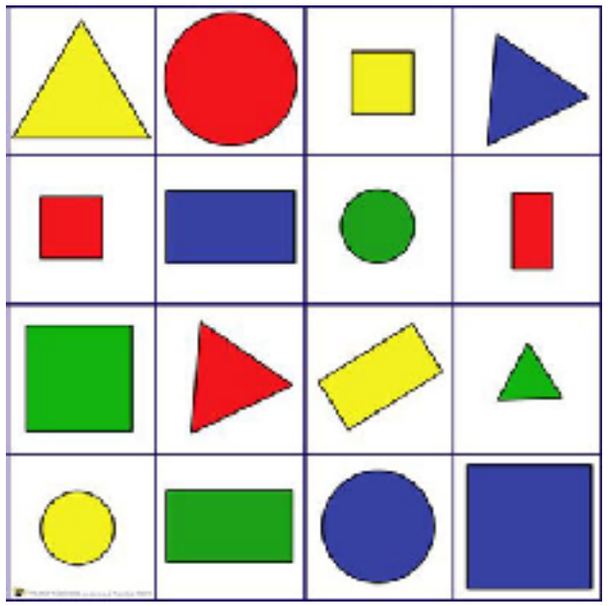


WHICH PIXEL IS AT (9,0)?



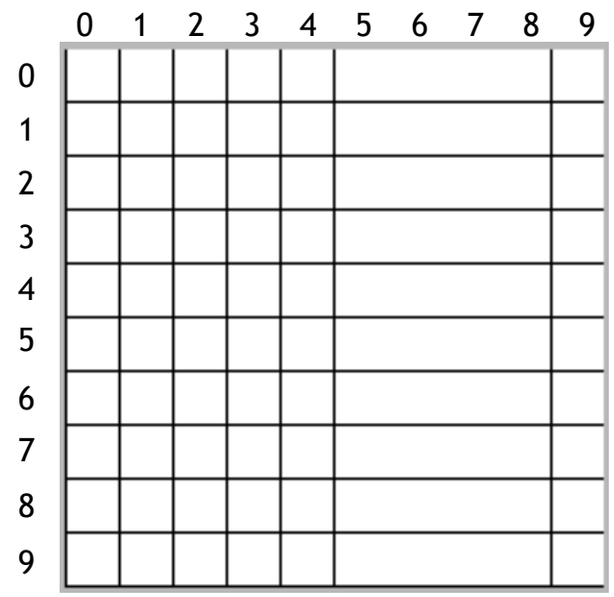
WHICH PIXEL IS AT (9,0)?

SHAPES.



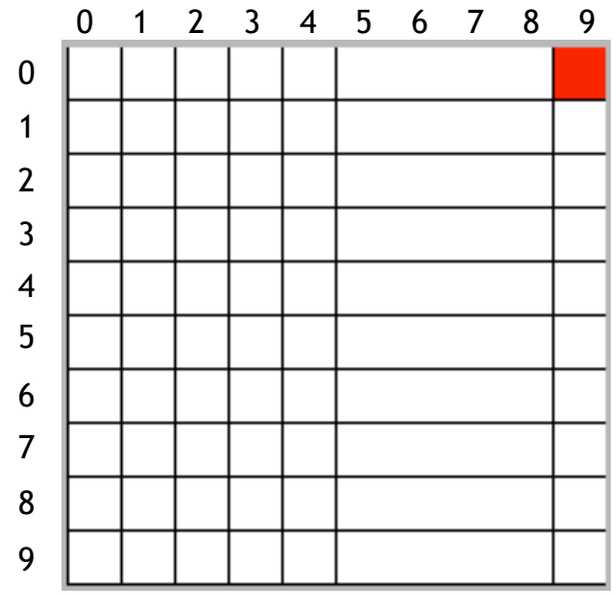
So let’s think about drawing specific shapes, which is one way of drawing things on the screen

What information do you need to draw a point?

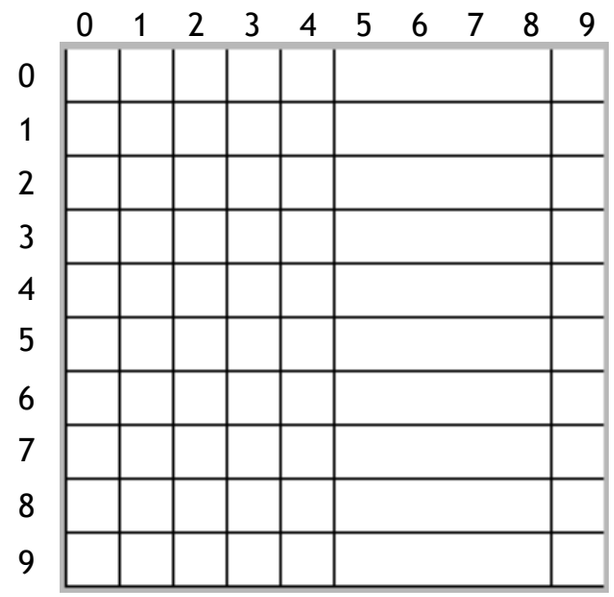


What information do you need to draw a point?

Just the (x,y) coordinate, such as (9,0) in this case.



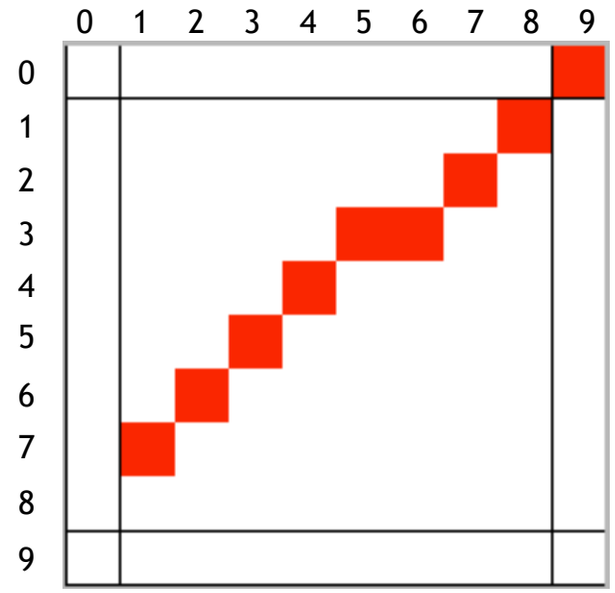
What information do you need to draw a line?



What about a start point, an angle, and a length? Sounds complicated!

What information do you need to draw a line?

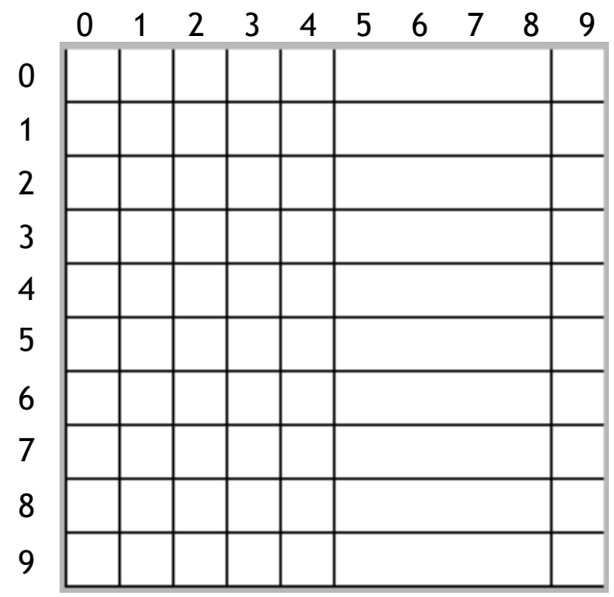
The starting and ending (x,y) coordinates, such as (9,0) and (1,7) in this case.



Notice how weird the line looks at this resolution.

It would look way smoother if we had more pixels to work with, just part of the secret shame of pixels.

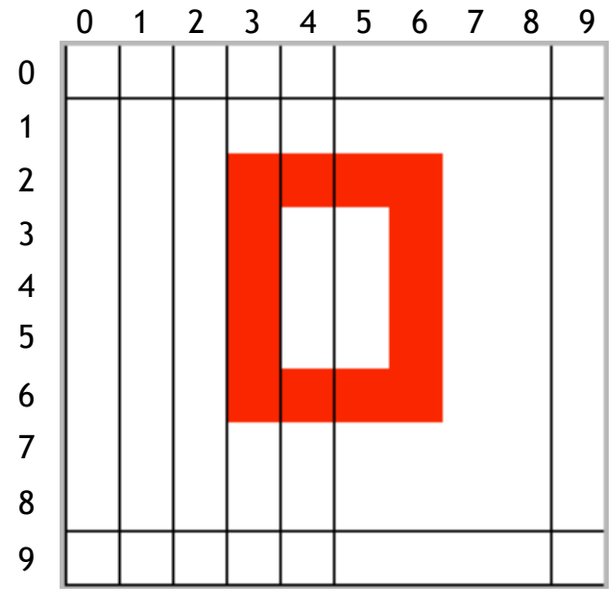
What information do you need to draw a rectangle?



What about all four points?
What about the top left and bottom right corners?
What about the center (x,y) coordinate and the height and width?

What information do you need to draw a rectangle?

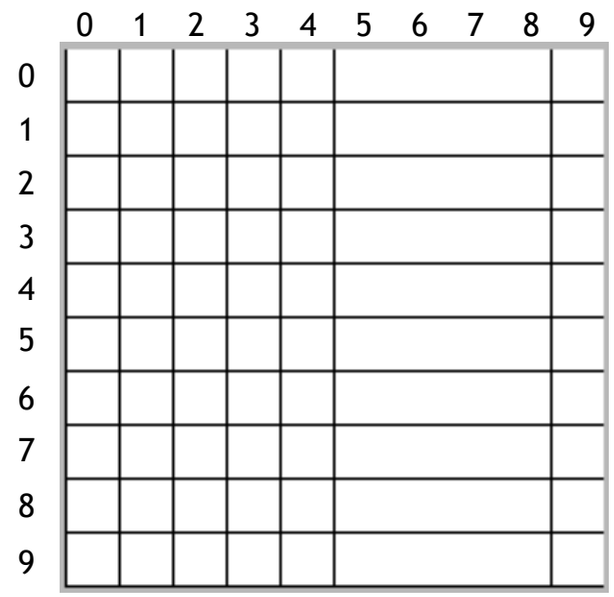
The top left corner's (x,y) coordinate, and its width and height. (3,2) and 4x5 in this case.



This is the usual convention in programming, but you can do it in other ways

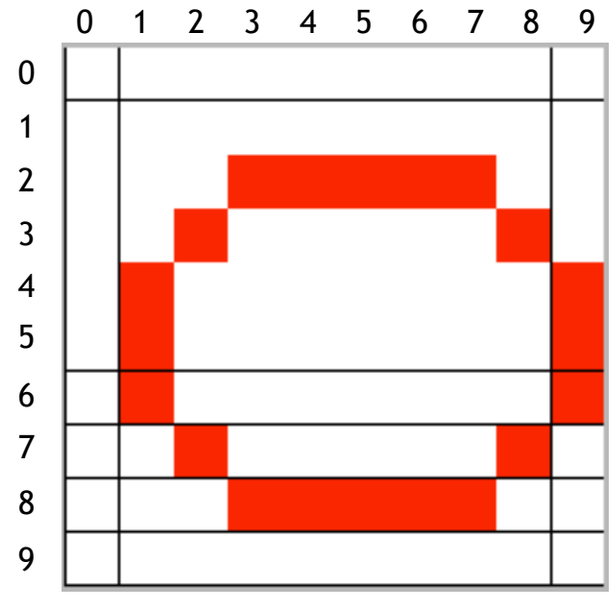
Why not just a width and height? (Because you need somewhere to start!)

What information do you need to draw an ellipse?



What information do you need to draw an ellipse?

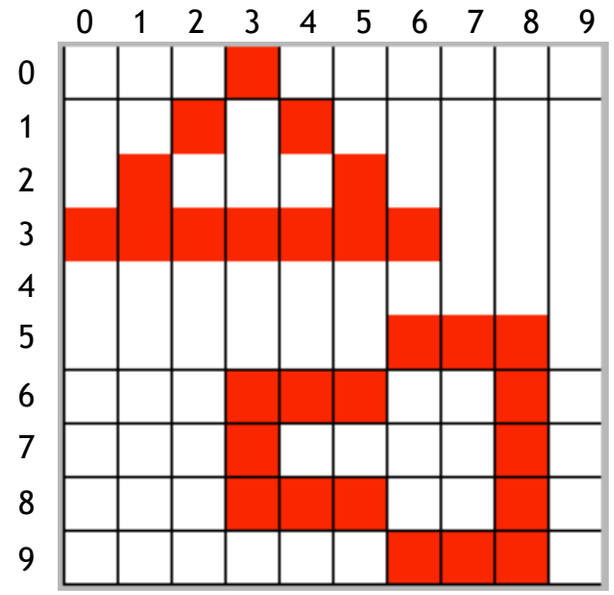
The center's (x,y) coordinate, and its width and height. (5,5) and 9x7 in this case.



Note that this is again the conventional way of drawing this shape on a computer.

You could draw an ellipse according to its top left "corner", it's just a bit weirder.

And so forth.



A triangle has three points, a quadrilateral has four, and so on and so on.

Again, notice how bizarre simple shapes look at low resolutions!

PROGRAMMING SHAPES.

Now we just need to know how to “tell” Processing to draw shapes, instead of thinking about it.

So begins our voyage into actually being able to do something with Processing.

IT'S THIS EASY...

```
point(x,y);
line(x,y,x,y);
rect(x,y,width,height);
ellipse(x,y,width,height);
triangle(x,y,x,y,x,y);
quad(x,y,x,y,x,y,x,y,x,y);
```

And so on.

```
point(x, y);
```

This is a “function call”. It’s the main way that we tell Processing to do something.

Functions: “do a thing”. Also interchangeable with the notion of a method. Or a command.

```
point(x, y);
```

First we have the actual function name, “point” in this case, which tells Processing the kind of thing we want.

Function names are usually pretty self-explanatory – “point” means it will draw a point.

Eventually you will be creating your own functions and it would be a good idea to remember to name them with self-explanatory names.

The function name has to be typed correctly, and it’s *case sensitive* so remember to make sure.

```
point(x, y);
```

Then an opening parenthesis, the “(“, which tells Processing we’re going to specify some parameters which affect the thing we want to do.

Processing, like all programming languages, is very touchy about what we call “syntax”. That means that if it expects an opening bracket, it better be there.

Else it won’t work.

```
point(x, y);
```

Then the parameters separated by commas, in this case the “x” and “y” coordinates of the point.

When we’re writing this kind of “specification” of a function, we often use placeholders like “x” and “y” in the example, rather than what would really be there, which is numbers*.

So a real version might be `point(5,8);`

Functions want you to give them *exactly the right number of parameters* or they won’t work.

* Or variables. Which we will get to later.

```
point(x, y);
```

Then a closing parenthesis, the “)”, which tells Processing we’re done with specifying the parameters.

One again, don’t miss out on the closing bracket or it will cause you headaches.

```
point(x, y);
```

Then a semi-colon, the ";", which tells Processing we're done with that instruction.

You need them at the end of every instruction in Processing (and most other programming languages).

(But you're also probably going to keep forgetting to put them there.)

SO NOW THIS SHOULD MAKE MORE SENSE.

```
point(x,y);
line(x,y,x,y);
rect(x,y,width,height);
ellipse(x,y,width,height);
triangle(x,y,x,y,x,y);
quad(x,y,x,y,x,y,x,y,x,y);
```

Let's try them out.

Head into Processing and try out variations of these commands

`size()` MATTERS

It's a bit cramped in that tiny window that pops up for us when we try to draw shapes.

In fact, we can tell the window to be any size we want by starting our program with this:

```
size(width,height);
```

Where the width and height are in… pixels.

Head over to Processing and try that out.

THE POINT OF ORIGIN

Remember how there were other ways to draw a rectangle or ellipse depending on where you started from? You can change the way it works if you want to:

```
rectMode(CORNER);
rectMode(CENTER);
rectMode(CORNERS);
ellipseMode(CENTER);
ellipseMode(CORNER);
ellipseMode(CORNERS);
```

(The defaults are in red.)

Note that if you change the mode, you will be specifying different things.

For rectMode(CORNER) or rectMode(CENTER) you specify the origin's (x,y) and the rectangle's width and height

But for rectMode(CORNERS) you just specify the top left (x,y) and the bottom right (x,y)

Head into Processing and try it out

QUESTIONS.

1/ How can we draw a square in the middle of the window?

2/ How can we draw an ellipse that is half-way off the right side of the window?

Note that we'll need to use size() so we know what size the window is.

[1]

```
size(800, 600);
rectMode(CENTER);
rect(width/2, height/2, 100, 100);
```

—-

[2]

```
size(800, 600);
ellipse(width, height/2, 100, 100);
```

http://processing.org/reference/

Let’s head over to the reference and look at how to use it because it will make life a lot easier for you

And it will mean I don’t have to spend as much time specifying different methods

You will love the reference. You will love it.

GREY.

We can actually tell Processing what shades of grey to make our shapes.

GREYS ARE NUMBERS.

We can specify a shade of grey in Processing by giving a number between 0 and 255.

0 means black.

255 means .

127 means medium grey.

Get it?

SETTING GREYS.

There are three main functions for setting greys.

One is for the entire background of the window:

```
background(shade);
```

One is for the inner colour of a shape:

```
fill(shade);
```

And one is for the line around a shape:

```
stroke(shade);
```

This is all in the reference of course.

It’s important to notice that these functions “stay on” from when you use them, until you change them.

So if you set fill(0); every shape will be filled with black from then on, until you use a different fill.

Try it out.

COLOUR.

We can actually tell Processing what shades of grey to make our shapes.

COLOURS ARE NUMBERS, TOO!

In Processing we will generally use what is called RGB colour.

It's called RGB because it's based on specify the amount of Red, Green, and Blue we want to combine.

We can specify the amount of each colour by giving a number between 0 and 255, just like with the greys.

So an RGB colour is specified by three numbers: (r,g,b)

Alternative: HSB - hue, saturation, brightness

COLOURS ARE NUMBERS, TOO!

(255,0,0) is pure red.

(0,0,255) is …

COLOURS ARE NUMBERS, TOO!

(255,0,0) is pure red.

(0,0,255) is pure blue.

COLOURS ARE NUMBERS, TOO!

(255,0,0) is pure red.

(0,0,255) is pure blue.

(85,107,47) is …

COLOURS ARE NUMBERS, TOO!

(255,0,0) is pure red.

(0,0,255) is pure blue.

(85,107,47) is a kind of olive green.

COLOURS ARE NUMBERS, TOO!

(255,0,0) is pure red.

(0,0,255) is pure blue.

(85,107,47) is a kind of olive green.

Check out https://www.rapidtables.com/web/color/RGB_Color.html for a bunch of individual examples of RGB colours.

Or use a graphics programming like Photoshop and use the colour-picker tool to find out the RGB values for a colour.

SETTING COLOURS.

It's exactly the same, but with three numbers (parameters) instead!

The entire background of the window:

```
background(r,g,b);
```

The inner colour of a shape:

```
fill(r,g,b);
```

The line around a shape:

```
stroke(r,g,b);
```

Try it out.

NOT SETTING COLOURS.

You can also tell Processing to have *nothing* filling a shape or its line – that doesn't mean "white" or something, that means *nothing*.

The inner colour of a shape:

```
noFill();
```

The line around a shape:

```
noStroke();
```

Try it out.

SLIGHTLY SETTING COLOURS.

You can also tell Processing to use different levels of opacity/ transparency by adding an “alpha” value to the parameters: you’ll need a parameter list of 2 if you're working in grey scale with alpha, and a parameter list of 4 if you’re working in RGB with alpha.

0 means completely transparent (the same as `noFill()` for instance), 255 means completely opaque.

Semi-transparent red fill:

```
fill(255,0,0,127);
```

A completely transparent black line! Pointless!:

```
stroke(0,0);
```

Try it out.

WHAT WILL THIS DO?

```
size(500,500);
background(100,0,0);
fill(255);
rect(0,0,250,250);
noStroke();
fill(0,0,250,250);
ellipse(250,250,250,250);
background(0,100,0);
```

size(500,500);
background(100,0,0);
fill(255);
rect(0,0,250,250);
noStroke();
fill(0,0,250,250);
ellipse(250,250,250,250);
background(0,100,0);

It’s a traaaaaap!

EXERCISE

Use the shape, point, line functions and colours (grey or otherwise) to draw a stick person.

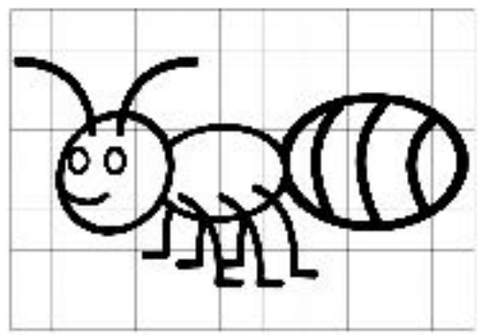
https://processing.org/reference/ is your close friend.

Time: **15 mins**

WAIT...

We type in commands to draw shapes, it draws them, and then it stops, and that's that. Like grid-based drawing mashed up with photoshop.

But wait, why would we use it that way?



What about interactivity? And computational magic?

Draw the shapes faster? Draw more shapes?

Looking for:

Repetition and **Reaction/Interaction**

SETUP, LOOP, AND REACT.

There are three fundamental ways that programs act across most programming language.

We setup the initial conditions of the program.

We repeatedly run a loop of code that makes the program work, usually by varying what the code does each time through.

And we make the program able to react to input from the user or from other events.

SETUP, LOOP, AND REACT.

Before a program gets going, we often want to set a number of starting conditions so that it is ready to go.

This might include: window size and shape, what things exist in our computed world, etc.

SETUP, LOOP, AND REACT.

Once the program is setup, we generally want it to run and keep going until we tell it to stop. Thus we need it to run a loop of instructions. This introduces time into the program, it now has a duration.

In interactive applications, we often want to do a bunch of things for every frame – maybe we want to simulate physics, move graphics, check for user input, etc.

SETUP, LOOP, AND REACT.

While the program is running through its repeated loop of instructions that make everything run smoothly, it may need to respond to input, especially from the person using it.

We need to know when they click a button or move a mouse and so on, and we need to react to that.

SETUP, LOOP, AND REACT.

We setup the initial conditions of our program.

We run through a loop that makes the world of the program run properly.

We react to to input or other events.

Processing has pre-established ways of doing all of these things.

Note that I'm writing this as if we are the program/game – this can be a helpful way of thinking about it, as we often need to "simulate" the program in our minds in order to understand it.

SETUP, LOOP, AND REACT.

```
void setup()
{
        // Set up things like window size
        // knowledge we already have about this little world
}
```

This is the `setup()` function, and it is how we tell Processing the things we want it to do only once at the very start of the program.

Let's have a look at this…

SETUP, LOOP, AND REACT.

```
void setup()
{
        // Set up things like window size
        // knowledge we already have about this little world
}
```

For the moment don't worry about this "void" thing. We will come to it later.

For now, just remember that it has to be there every time you write a `setup()` function.

SETUP, LOOP, AND REACT.

```
void setup()
{
        // Set up things like window size
        // knowledge we already have about this little world
}
```

This the name of the function. It's called "setup" because it's where you setup your program.

Remember, because it's just "setup" it only does what you tell it here one time.

SETUP, LOOP, AND REACT.

```
void setup()
{
        // Set up things like window size
        // knowledge we already have about this little world
}
```

These empty parentheses are empty because **`setup()`** doesn't need any parameters, unlike something like **`rect()`**.

Remember to put these empty parentheses in every time!

SETUP, LOOP, AND REACT.

```
void setup()
{
        // Set up things like window size
        // knowledge we already have about this little world
}
```

This opening curly bracket tells Processing we're about to tell it what we want to do during the **`setup()`** function. We're saying "here come the instructions I want you to run exactly once at the start of the program!"

SETUP, LOOP, AND REACT.

```
void setup()
{
        // Set up things like window size
        // knowledge we already have about this little world
}
```

This is where we type in all the instructions of what to do during setup. They will happen at the start of the program.

Those double slashes, “//”, are used for comments. Comments aren’t part of the instructions of the program, you put them there to help you with your programming.

Comments are incredibly important, they will help you to no end and I will talk about them more next week.

SETUP, LOOP, AND REACT.

```
void setup()
{
        // Set up things like window size
        // knowledge we already have about this little world
}
```

This closing curly bracket tells Processing that we're done with telling it the instructions for `setup()`

As with all of these bits of syntax, it has to be there! You will inevitably forget to put these curly brackets in sometimes and it will be a headache. Oh well.

SETUP, LOOP, AND REACT.

```
void draw()
{
  // Do these things over and over again,
  // like moving an avatar, etc.
}
```

In Processing, the function that repeats over and over again is called `draw()`

It will run the instructions inside once every frame of your program.

In a lot of game libraries the function that gets repeated over and over is called “update”

SETUP, LOOP, AND REACT.

So now we can write a simple program that will do some setup in `setup()` and then repeatedly run some instructions in `draw()`!

Let's take a look at one and see if we can work out what it will do…

In particular, what's wrong with this program?

```
void setup()
{
  size(500,500);
  background(255,0,0);
  rectMode(CENTER);
  rect(250,250,400,400);
}

void draw()
{
  fill(0,0,255,255);
  rect(250,250,200,200);
}
```

SETUP, LOOP, AND REACT.

So now we can write a simple program that will do some setup in `setup()` and then repeatedly run some instructions in `draw()`!

Let's take a look at one and see if we can work out what it will do…

… oh, nothing changed. Processing was just drawing the exact same rectangle over and over again in `draw()`…

What could we do to make the repetition visible?

Nothing actually changes in the draw() loop, and so we don't see anything change.
If we change the alpha of the fill of the draw() rectangle, we can see a change…

```
void setup()
{
  size(500,500); // Setting a size for the window
  background(255,0,0); // Setting the background to red
  rectMode(CENTER); // Setting the rectangle drawing mode to draw from the center
}

void draw()
{
  fill(0,0,255,1); // Setting the fill to be blue at a very low alpha (very transparent)
  rect(250,250,200,200); // Drawing a square in the middle of the screen.
}
```

SETUP, LOOP, AND REACT.

Okay, so now we know that we can setup our initial conditions in `setup()` and have things happen over and over again in `draw()`.

But what we'd really like is for the program to react to us while it's running, rather than just ignoring us like it does now.

So…

SETUP, LOOP, AND REACT.

So now let's look at some simple ways that Processing can react to input from the user.

The two easiest things we can use right now are the mouse and the keyboard.

With the mouse we might want to know *where* it is and *when* someone clicks a button.

With the keyboard we might want to know *when* someone presses a key and which one it was.

SETUP, LOOP, AND REACT.

Processing is actually *always* keeping track of where the mouse is in its window as an (x,y) coordinate in pixels.

It stores the latest (x,y) coordinate in two variables called

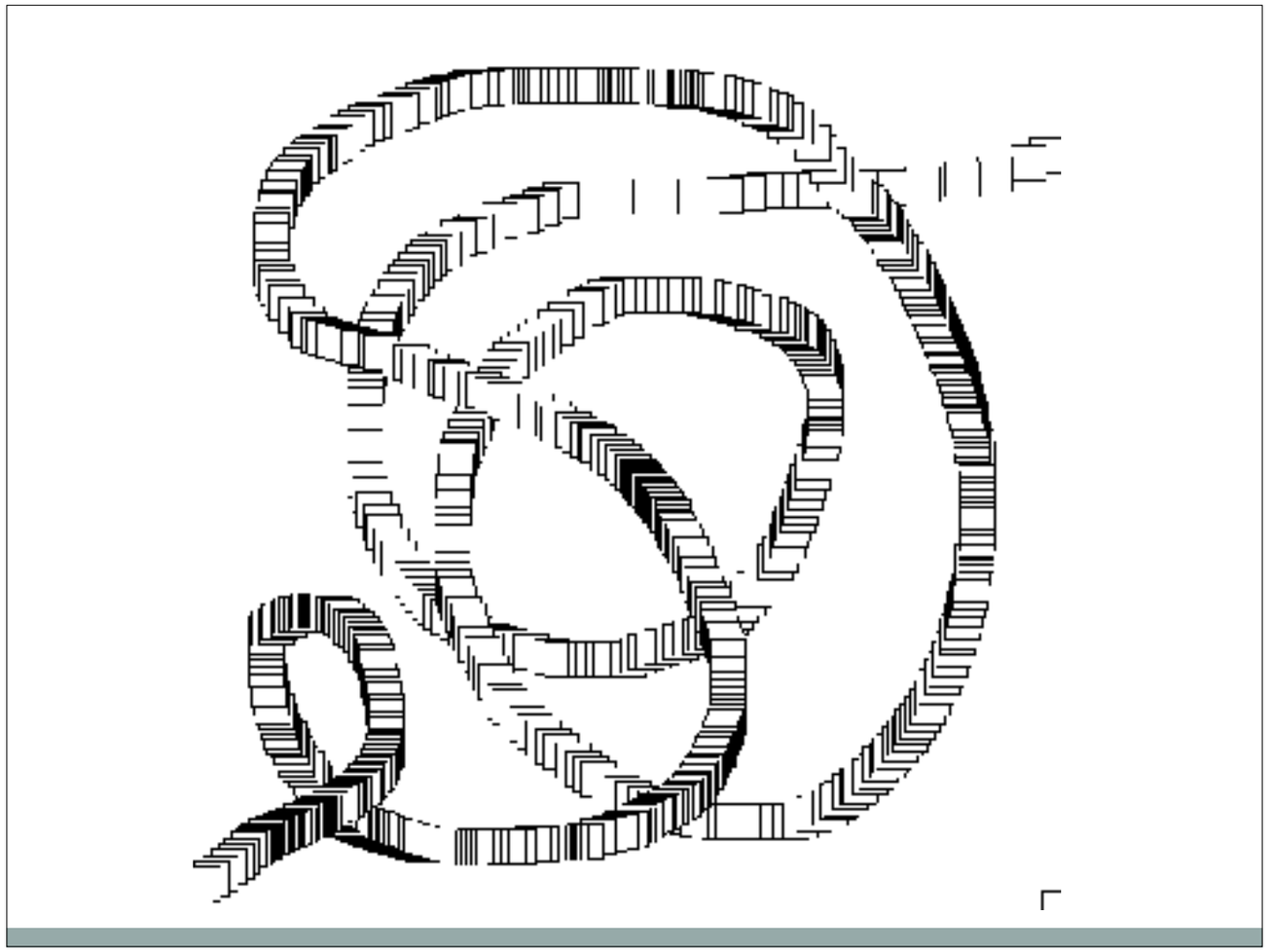
`mouseX` and `mouseY`

If you use `mouseX` and `mouseY` somewhere in your program, they will be replaced by the x and y coordinates of the mouse respectively.

Let's look at some code.

```
void setup()
{
  size(500,500);
  background(255);
  rectMode(CENTER);
}

void draw()
{
  rect(mouseX,mouseY,20,20);
}
```



Why does this look like an awesome 3D drawing instead of just a rectangle following the mouse?

How would we fix it if we wanted to?

(By adding a background() instruction to the draw() loop as well.)

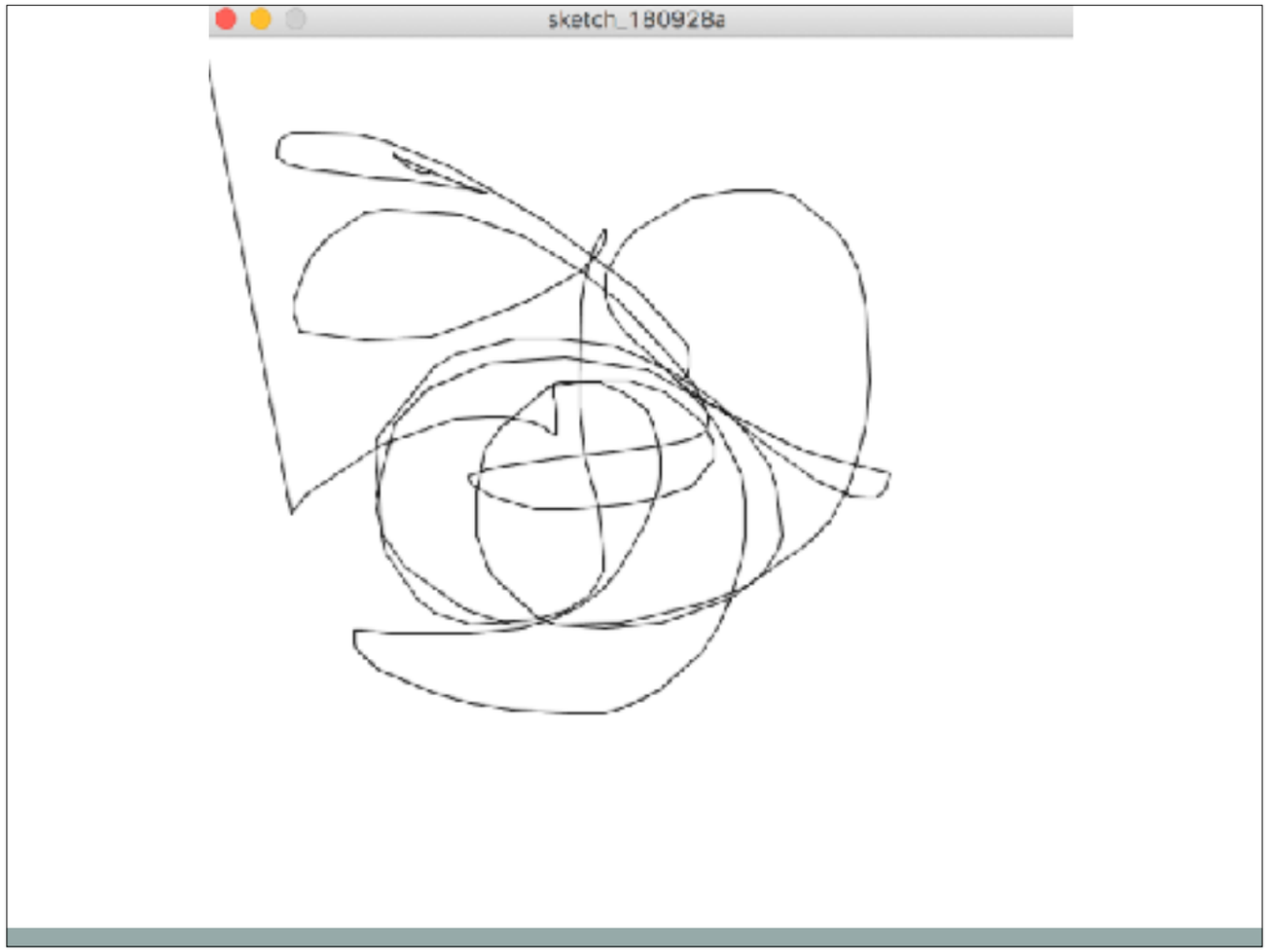
SETUP, LOOP, AND REACT.

Processing also stores the last values of `mouseX` and `mouseY` in `pmouseX` and `pmouseY`

Let's look at more code.

```
void setup()
{
  size(500,500);
  background(255);
}

void draw()
{
  line(pmouseX, pmouseY, mouseX, mouseY);
}
```



Why did the line “jump out” from the top left corner at the beginning?

(It’s because when the program starts off there is no value for pmouseX and pmouseY, and so it has them as 0 and 0 instead)

(In fact, if you look back at the cool 3D rectangle trail, you’ll see that **that** started with a rectangle in the top left corner (0,0) too!)

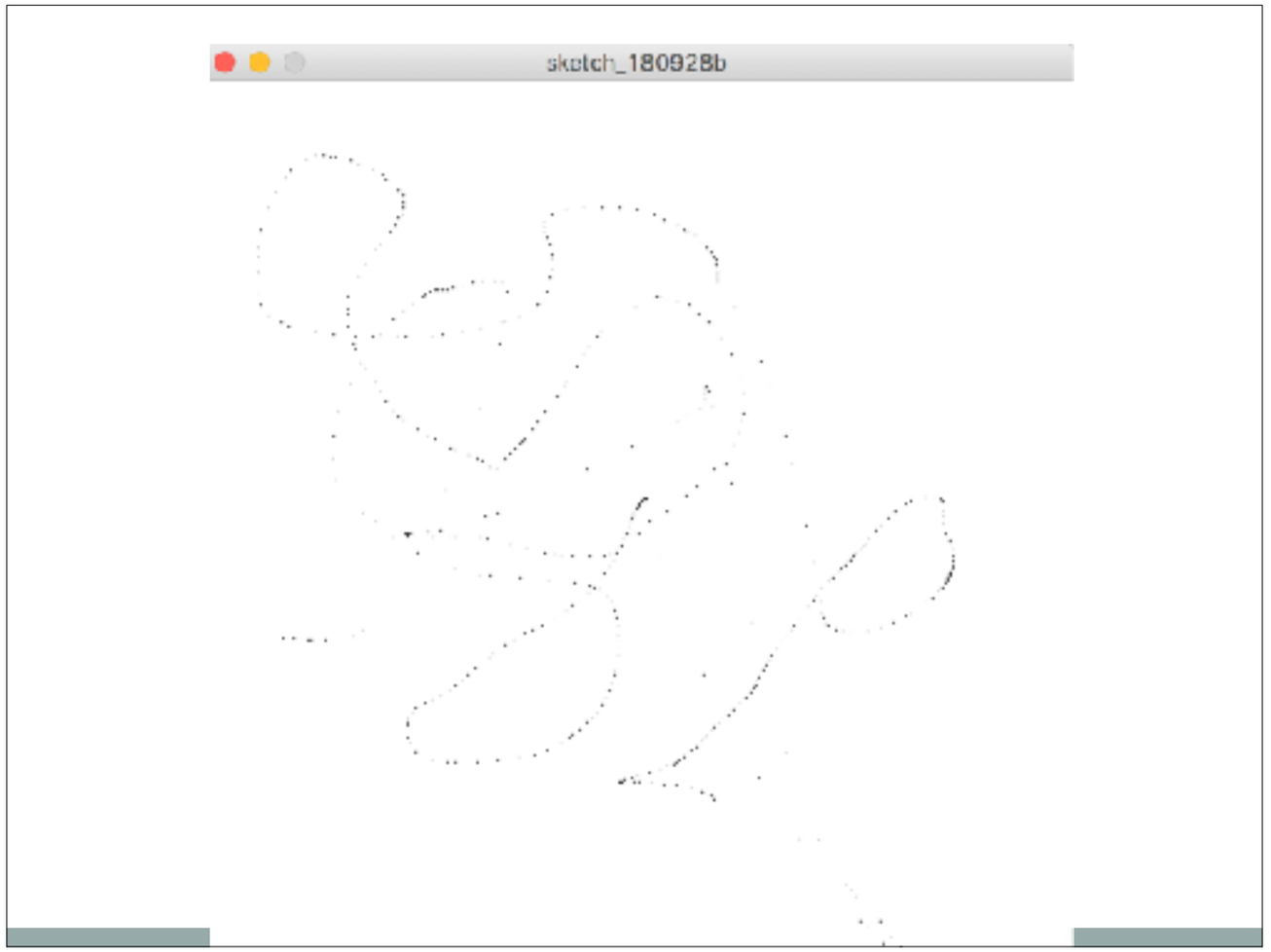
(So apparently mouseX and mouseY don’t exist until you **move the mouse**.)

SETUP, LOOP, AND REACT.

What if we were to use mouseX and mouseY with the `point(x, y)` command?

```
void setup()
{
  size(500,500);
  background(255);
  stroke(255,0,0);
}

void draw()
{
  point(mouseX, mouseY);
}
```




```
void setup()
{
  size(500,500);
  background(255);
}

void draw()
{
  point(mouseX, mouseY);
}
```

TEXTBOOK ON PROCESSING

“Learning Processing” by Daniel Shiffman.

Available as an ebook from the Concordia Library.

OFFICE HOURS

Thursday afternoon, 2 - 4pm

WOMEN AND HOMELESSNESS SYMPOSIUM **THIS THURSDAY 2-4PM**

We will meet outside this classroom at 1:50pm this Thursday.

I want us to be at the location - 9th floor JMSB - at 2pm sharp.