CREATIVE CODING WITH PROCESSING

developed and lead by:
kathryn barrett
hi@kathrynbarrett.ca | @kat_barrett
november 17 2013

WHAT IS PROCESSING?

- Processing is a language environment that is appealing to artists and designers learning programming for the first time
- It's built on top of the Java language - meaning it employs all the same principles, structures, and core concepts of Java and other similar languages
- Processing doesn't cost a dime and is open source!
- Unlike some languages where you input text and receive a text output (like Ruby), Processing allows you to input text and receive a visual output
- AKA if you're a visual learner, this language is for you!

WHY PROCESSING?

- It can shed new information on already existing things:
  Forms https://vimeo.com/38421611

- We can find and display common patterns:
  To understand is to perceive patterns https://vimeo.com/34176163

- It can help us better understand large amounts of data:
  Just Landed https://vimeo.com/4587178

- It looks cool!
  Augmented Structures https://vimeo.com/29458889

- It's fun!
  Digital Ribbons https://vimeo.com/2430949

PROCESSING COMMUNITIES

- openprocessing.org is probably the most popular. Thousands of completed and experimental sketches live here. You can view/ download someone's code, learn from it and build upon it. (under creative commons licensing)

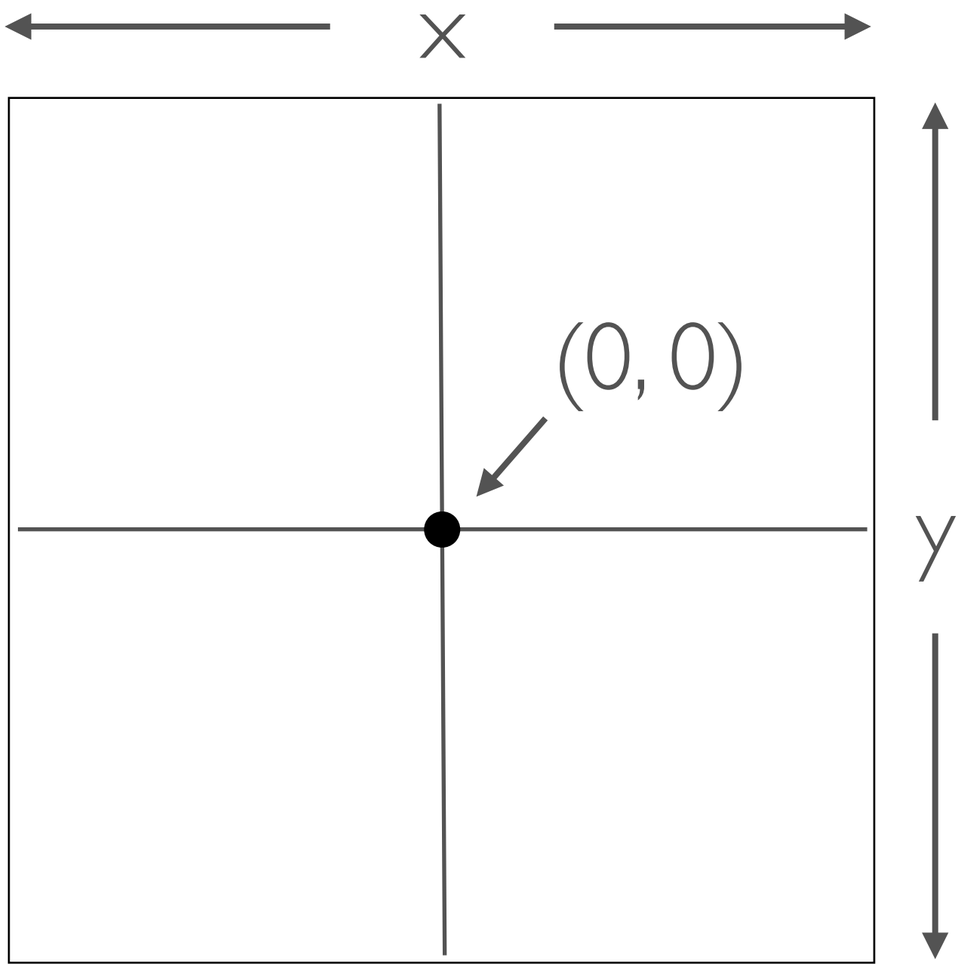
- forum.processing.org is the official forum for Processing. This community is AWESOME. There is never a question too out there, and everyone who frequents this website is there to share knowledge and help each other learn

THE SKETCHBOOK

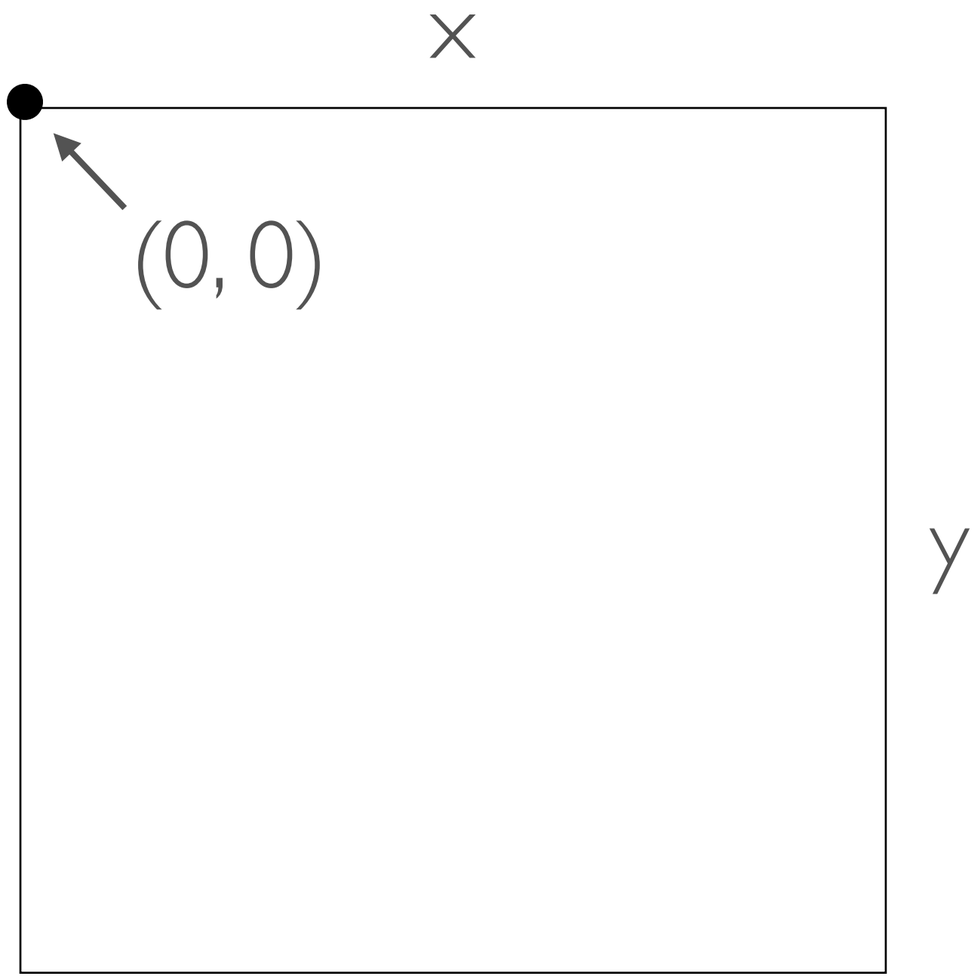
- Processing programs are often referred to as “sketches”
- The folder where you store your sketches is called your sketchbook
- The file extension for Processing is “.pde”
- When saving sketches, do not use spaces or hyphens, and do not start your sketch name with a number

THE BASICS

PIXELS



common graph



your computer (in pixels)

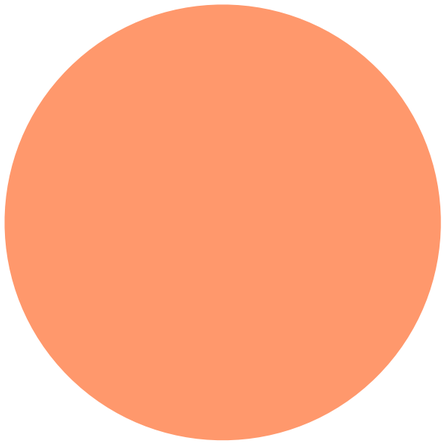
SHAPES

Rectangle

- rect (x, y, width, height);
  rect (100, 200, 50, 125);

Circle

- ellipse (x, y, width, height);
  ellipse (125, 275, 100, 100);



FUNCTIONS

- When drawing a rectangle or circle on our screen, we are providing a command for the computer to follow

- This is otherwise known as a function. The term rect and ellipse as seen in the last slide are built in functions that Processing recognizes

- Notice how we also specify where the shape is located and how big it is in parenthesis. This is known as the function's argument

- Each function and its argument must always end in a semicolon

SETUP

SKETCH SETUP

- In order to write our very first program, we must initialize the sketch's setup. Functions that only happen once are located here. For example, the size of a sketch

- When we place functions inside the setup, this is known as nesting. The functions themselves are known as a block of code

- We start a block of code with an opening "{" and end it with a closing "}"

SKETCH SETUP

```
void setup ( ) {

size (600, 600);

}
```

begin setup of our sketch

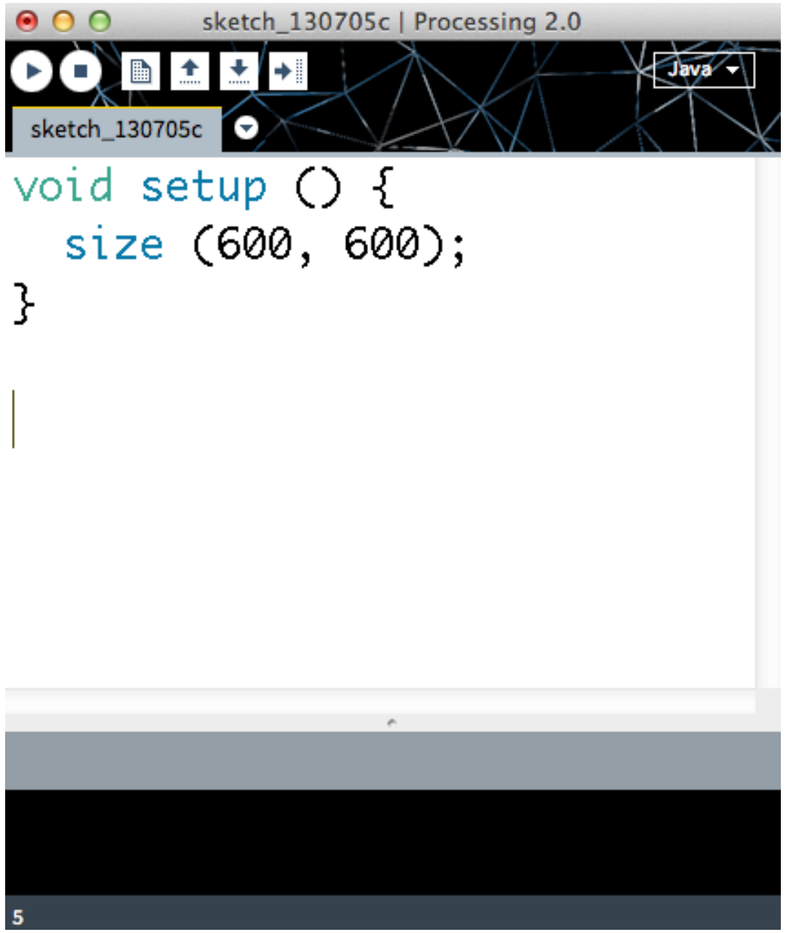
this is the size of our sketch (in pixels!)

end the setup of our sketch

** “void” refers to a function that returns no value.

RUNNING YOUR SKETCH

- To visually see what you just coded, press the play button in the top left hand corner



DRAW

ALGORITHMS

- Computer programming is all about writing specific instructions that are in sequence - otherwise known as an algorithm

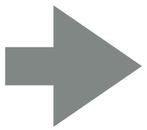
- If asked the question “how do I drive a car?” your answer (in sequential steps) would be known as an algorithm

DRAW

- The function draw acts like a loop. It draws over top of your output window over and over and over again until you quit the sketch

- It's important to remember the term algorithm here. Draw always always *always* displays code in sequence

```
rect (100, 200, 50, 125);
ellipse (125, 275, 100, 100);
```



In this example, the rectangle is drawn first, followed by the circle. However, the rate at which draw loops is extremely fast, and therefore not actually visible to the naked eye

COLOR

- The fill function specifies what color to make a shape:
  fill (255, 10, 100);

- The stroke function specifies what color to make the border around a shape:
  stroke (100, 50, 255);

- But wait… what do those 3 numbers in between the parenthesis refer to?

RGB

- Individual colors are expressed in a range from 0 (none of that color) to 255 (as much of that color as possible)
- 0 = black | 255 = white
- 3 values - red, green, blue combined
  e.g. 155, 15, 215 = purple
- To get the RGB values of a color in Processing:
  > Tools > Color Selector

COLOR TRANSPARENCY

- In addition to red, green and blue, there is also an optional fourth component that specifies transparency

- This value also ranges from 0 to 255, with 0 being completely transparent and 255 being completely opaque

- The following is an example of how it would look:
  fill (255, 160, 52, **100**);

DRAW

```
void draw ( ) {

background (255);

fill (240, 10, 10, 155);

ellipse (100, 100, 100, 100);

}
```

let's draw something in our sketch

the background color of our sketch in RGB

the color of our rectangle in RGB

the location and size of our rectangle

let's stop drawing something in our sketch

TWO OR MORE SHAPES

```
void draw ( ) {

fill (240, 10, 10);
```

this fill is being applied to this rectangle

```
rect (50, 100, 400, 200);

fill (50, 255, 10);
```

this fill is being applied to this circle

```
ellipse (200, 100, 50, 50);

}
```

Always specify how your shape is going to look BEFORE you draw your shape

1. fill
2. shape

THE REFERENCE

- All the functions that we've looked at so far are all part of Processing's built in library. You can browse every term in the library by using something called the Reference

- To find the reference:
  > Help > Reference
  We will only go over a few of these today, but once you feel comfortable, take a look at others

- If you forget what a term in your code means or want a proper definition with examples, highlight the term and go to > Help > Find In Reference

- THE REFERENCE IS YOUR BEST FRIEND!

ERRORS

- Processing is case sensitive: lowercase and CAPITAL letters matter!
- For example, typing “Rect” instead of “rect” will cause an error and will not open the output window
- Errors also often come from missing squiggly brackets, parenthesis, and semicolons
- SAVE OFTEN!

TIPS & TRICKS

- Processing doesn't care about s p a c i n g ! You can space out functions, variables, or anything else in Processing as much or as little as you like

- As we add more and more code to our exercises, you might start to see things getting a little messy and confusing (squiggly brackets not lining up, or nesting not being visually appealing). Use the Auto Format tool (> Edit > Auto Format) to organize your code. This will also become your best friend!

GOOD HABITS: COMMENTING CODE

```
void draw ( ) {                                  // this is where I start drawing

background (100, 215, 225);      // the background color is blue

fill (240, 10, 10);                              // the rectangle is red

rect (50, 100, 400, 200);                // x, y, width, height

}                                                // this is where I stop drawing
```

INTERACTION

MOUSE X AND MOUSE

- Let's recall from the beginning of the workshop:
  rect (x, y, width, height);

- Create a rectangle in void draw and try replacing the x and y values with the terms mouseX and mouseY

- Based on your interaction, define mouseX and mouseY

RECTMODE

- Notice how your mouse cursor is at the top left corner of your rectangle
- This means that your mouse cursor is taking the coordinates (0,0) of the actual rectangle and moving it accordingly
- To make the cursor centered within the rectangle, we can use a function called rectMode and an argument called CENTER
- Apply the following above your rectangle:
  rectMode (CENTER):

LET’S PAINT!

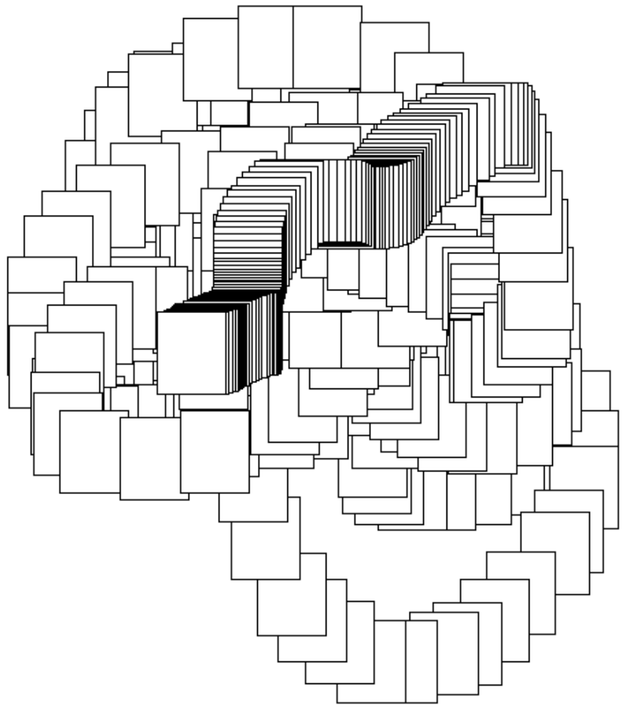
UNDERSTANDING DRAW

- Recall that setup only happens once and draw is a loop, drawing whatever is in that function over and over and over again
- In the program that we just wrote, the background color is doing just that - and so is the rectangle
- This means that once a rectangle is drawn, the background instantly draws right over top of it. We only ever see one rectangle!
- Let's change that!
- Copy the background and paste it into setup instead. What happens?

OHHH... AHHHHH

- Now, the background is only being drawn once - and our rectangle is getting drawn over and over and over again on top of it

- Just like painting!



ALMOST…

- In any digital paint program, you must click and drag your mouse to draw. Once you release, drawing stops

- Let’s create this instance by placing our “paintbrush” in it’s own function and only calling that function in draw when we press our mouse

PAINTBRUSH

- Below draw, create a new function called void paintbrush ( ) {
- Within this function, copy and paste your rectangle (and it's fill and rectMode) from draw
- Draw should now be empty!
- To call our new function in draw, we're going to add the following: paintbrush ( ) {

ADDING INTERACTION

- As of right now, we have no user interaction
- In English, our next exercise is going to sound like this: when mouse is pressed, start drawing. When mouse is not pressed, do nothing
- To do this, let's take a look at boolean expressions and a built in function called mousePressed

CONDITIONALS

- A boolean expression evaluates to either a true or false statement.
  I am hungry = true. I am afraid of programming = false

- An if, else statement is a specific type of boolean expression

- In essence, it does exactly what it sounds like:
  If this is true, do this. Else it must be false, so do something different

- A real world example could be: If weather is warm, leave jacket at home. Else, weather must be cold, bring jacket

MOUSEPRESSED

- Now that we have an understanding of if, else, we can use an argument to tell that function what to do

- In this case, we're going to use mousePressed

- if (mousePressed) {
  }
  else {
  }

NESTING

- Now nest your rectangle within the if block

- Nested within the else block is going to nothing (since we want nothing to happen when we click our mouse)

COOL, RIGHT?

- We now have a basic paint program
- Experiment using mouseX and mouseY as width and height. What happens?
- Experiment by adding multiple shapes, sizes and colors so you're drawing many things at a time but in different locations within the sketch

OTHER FUNCTIONS FOR SHAPE STYLE

- stroke (0); // apply a black stroke to shape
- strokeWeight (5); // apply a stroke thickness of 5 pixels to shape
- noStroke ( ); // do not apply a stroke to shape
- noFill ( ); // do not fill shape with colour

KEYPRESSED

- But what if we dislike what we just drew? As of right now, we would have to stop the sketch and run it again to clear

- Let's use another function called keyPressed to say: whenever a key is pressed on your keyboard (any key!), your sketch will draw another background overtop of what you've drawn and simply hide it

- Just like mousePressed, we're going to use an if statement!

KEYPRESSED

- In draw, add the following:

```
if (keyPressed) {
background (255);
}
```

NEXT STEPS

- Experiment! Use the reference to add new forms of interaction
- Share sketches and comment on others on openprocessing.org
- Prototype and build to better understand concepts. Being able to visually see code is a huge advantage!
- Remember: There are many different ways of doing the same thing. Although I've taught you the most basic, I encourage you to explore other ways of solving problems

THANK YOU!

developed and lead by:
kathryn barrett
hi@kathrynbarrett.ca | @kat_barrett
november 17 2013