Introduction to Processing: Part 1

Tech Focus III: Caring for Software Based Art

September 26, 2015

Processing

Software is the best way I've found to express myself. When I work in other media, the results somehow always seem worse in reality than in my head. The software I create, however, has a magical quality: it ends up being better than what I originally imagined.

– Martin Wattenberg

From: Processing (Reas & Fry, 2007)

Art History and Intellectual History

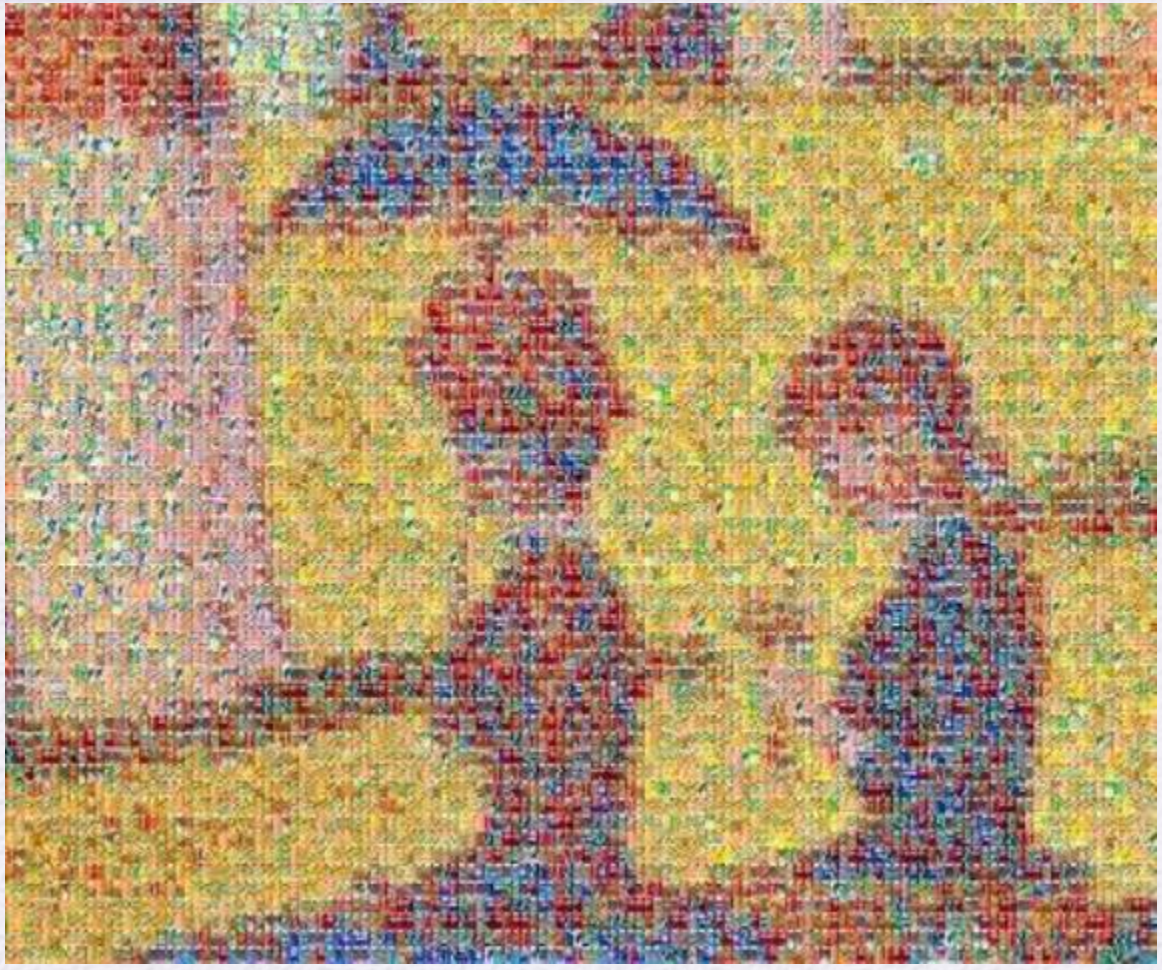
- There is a school of thought … that claims that mathematical and scientific discoveries … were previously seen in the visual arts.
- The most commonly cited example is perspective: artists figured it out well before the mathematicians did!
- Another example is a comparison between Pointillism and Post Expressionism with 20th century development of display technologies.

Art & Physics: Parallel Visions in Space, Time, and Light
By Leonard Shlain (William Morrow Paperbacks, 2007)

Pointillism …

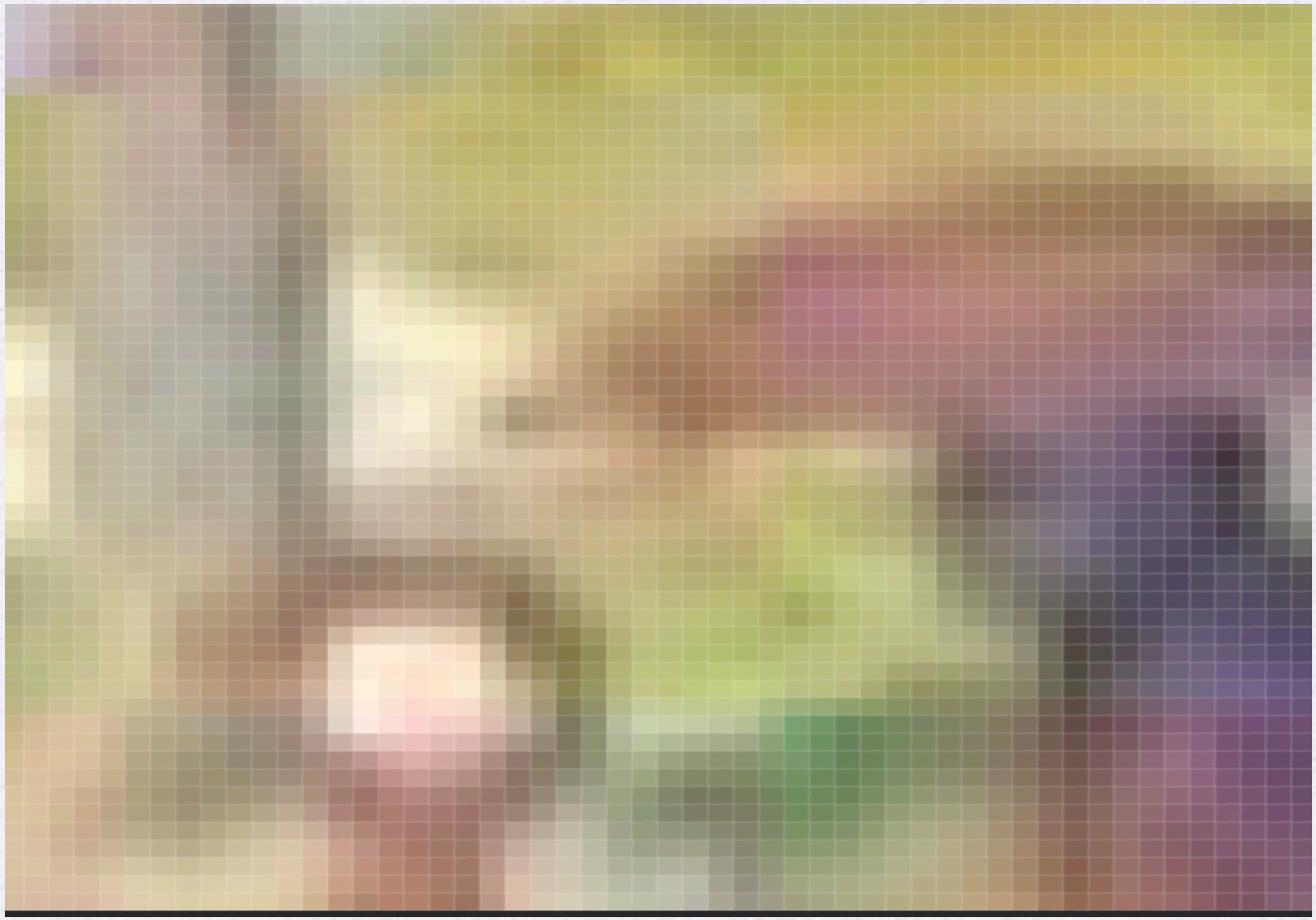
A Sunday on La Grande Jatte, Georges Seurat, 1884 Art Institute of Chicago

Pointillism … Magnified



A Sunday on La Grande Jatte, Georges Seurat, 1884 Art Institute of Chicago

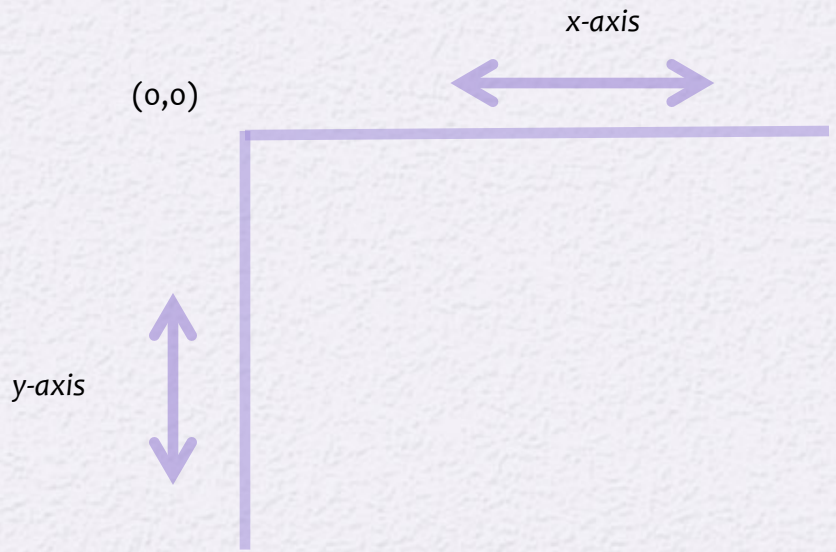
… and pixelated:



Detail at 1600%: A Sunday on La Grande Jatte, Georges Seurat, 1884 Art Institute of Chicago

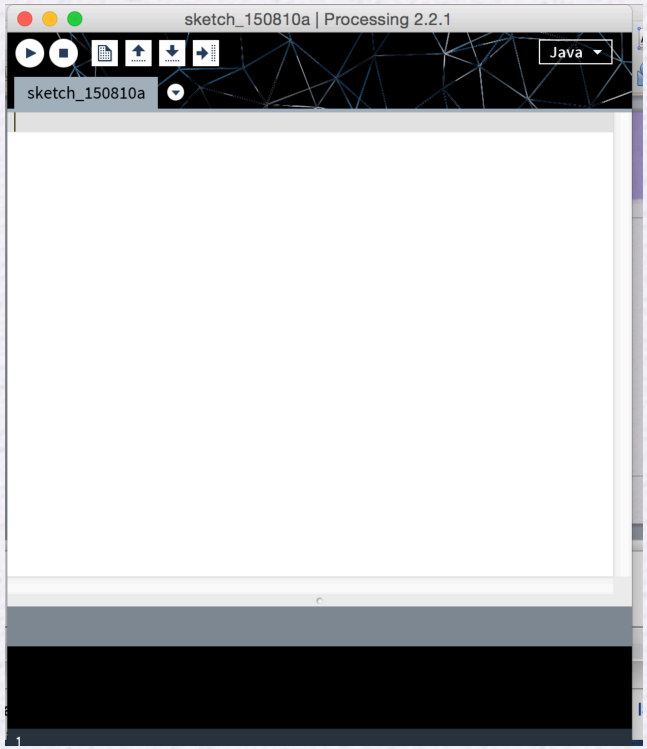
Processing: Addressing Pixels on a Grid

- Every pixel has an x-coordinate, a y-coordinate and a color.



Processing

- Open *Processing* with a blank sketch



Processing

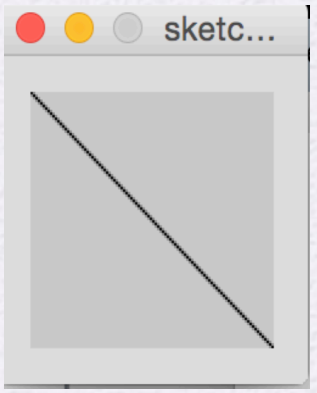
- *Processing* is widely used to teach programming skills.
- *Processing* was initially written to use Java but it also now supports python and there is a JavaScript version available as well.
- Our focus today will be on using *Processing* as an artist, so that you can see how it feels to be a digital artist.
- I would be happy to answer specific programming questions and provide additional materials as well during our discussion and work periods for anyone who is interested!

Drawing a line:

Type this code into your Sketch

```
line(0,0,200,200);
```

Then click on the yellow "play" button:



Use the square "stop" button before playing a second time.

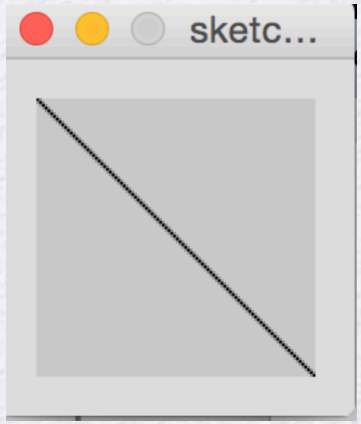
Notice that every statement in Processing ends with a semi-colon (and not a period … or a question mark!)

Drawing a line:
Let's look at the code.

```
line(0,0,200,200);
```

Here is how a line is defined in Processing:

line (a,b,c,d) is defined where *(a,b)* are the *(x,y)* coordinates of the first point on our grid of pixels and and *(c,d)* are the *(x,y)* coordinates of the second point on our grid.



See https://processing.org/reference/line_.html

Drawing a line

```
line(0,0,200,200);
```

In programming terms, `line` is a function and `0,0, 200 and 200` are the parameters.

In other words, anytime you want to draw a line, you can call the line function and spell out the (x,y) coordinates for the two points that define your line.

The processing environment offers many functions. You can also write your own!

See https://processing.org/reference/line_.html

Adding to your sketch

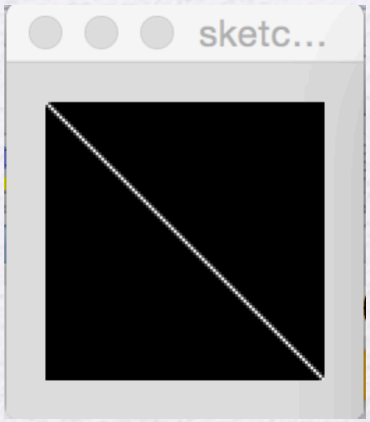
- When you create a sketch, it is saved in a folder along with a .pde file of the same name. For example, the above drawing is stored in a folder called `sketch_1_line` and named `sketch_1_line.pde` on your drive.
- Next let's add the following code and call it `sketch_2_line`:
  - The size of the drawing in pixels
  - More lines and other shapes
  - Color
  - Comments into your code

Let's render your line in white on a black background …

- If you add the first two lines above your code:

```
background(0);
stroke(255);
line(0,0,200,200);
```

- … here is your new sketch:



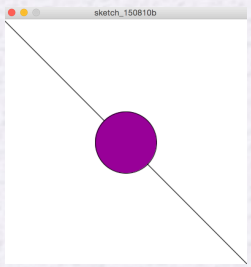
A Word About Color

- We will be using color today in RGB mode (*red*, *green*, and *blue*) and we will measure each of the three colors in a range of 0-255.
- Color can be defined for ***stroke*** (lines and/or outlines) and/or ***fill*** (contents of a shape)
- (Hint: use ***nofill()*** to leave a shape with only an outline!)
- Black can be expressed as (0) and white can be expressed as (255).
- Note there is a Processing tutorial on color posted to https://www.processing.org/tutorials/color/

We can also change the colors … and add another shape:

- The following code (saved on your drive as `sketch_3_circle`) renders:

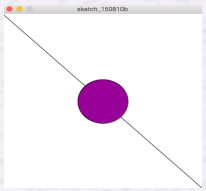
```
size(400,400);
background(255);
stroke(0);
fill(150,0,150);
line(0,0,400,400);
ellipse(200,200,100,100);
```



Notice that we can annotate the code using comments.

- Here is the same code … annotated:

```
size(400,400);     // set up the size of the window
background(255);  // the background is white
stroke(0); // lines and outlines are in black
fill(150,0,150);   // fill shapes in purple
line(0,0,400,400); // draw a diagonal in black
ellipse(200,200,100,100);  //draw a purple circle
```



- Comments are for human readers only … the computer ignores them!

We will now give you time to do a sketch of your own!

How do these shapes look … in different colors?

- Line: *line (a,b,c,d)* where *(a,b)* is the first point and *(c,d)* is the second point.
- Rectangle: *rect(a,b,c,d)* where *(a,b)* is the upper left corner; *c* is the width in pixels and *d* is the height in pixels.
- Ellipse: *ellipse(a,b,c,d)* where *(a,b)* is the center point; *c* is the width in pixels and *d* is the height in pixels. Note that if *c* is equal to *d*, *ellipse* draws a circle.
- Triangle: *triangle(a,b,c,d,e,f)* where *(a,b)*, *(c,d)* and *(e,f)* represent the three points that create the triangle.

Notes for your sketch:

Shapes:

Line: *line (a,b,c,d)* where *(a,b)* is the first point and *(c,d)* is the second point.

Rectangle: *rect(a,b,c,d)* where *(a,b)* is the upper left corner; *c* is the width in pixels and *d* is the height in pixels.

Ellipse: *ellipse(a,b,c,d)* where *(a,b)* is the center point; *c* is the width in pixels and *d* is the height in pixels. Note that if *c* is equal to *d*, *ellipse* draws a circle.

Triangle: *triangle(a,b,c,d,e,f)* where *(a,b)*, *(c,d)* and *(e,f)* represent the three points that create the triangle.

Colors:

Red (255,0,0)

Green (0,255,0)

Blue (0,0,255)

White (255)

Black (0)

Reddish Violet (255,0,255)

Olive green (100, 150, 0)

Navy blue (50, 75,150)

Online color picker: http://www.w3schools.com/tags/ref_colorpicker.asp

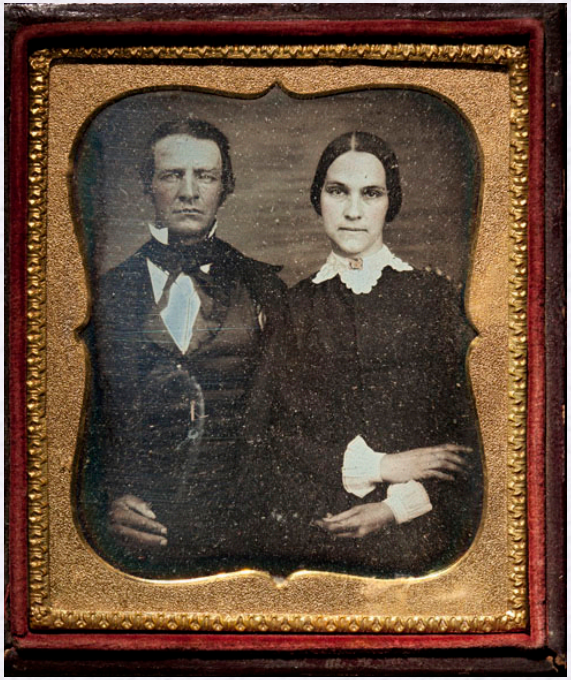
Introduction to Processing: Part 2

Tech Focus III: Caring for Software Based Art

September 26, 2015

Art and New Technologies

- Early photographs were reportedly used to capture images that might otherwise have been rendered in paintings or drawings.



- *Source: http://www.si.edu/MCI/EarlyPhotography/about.html*

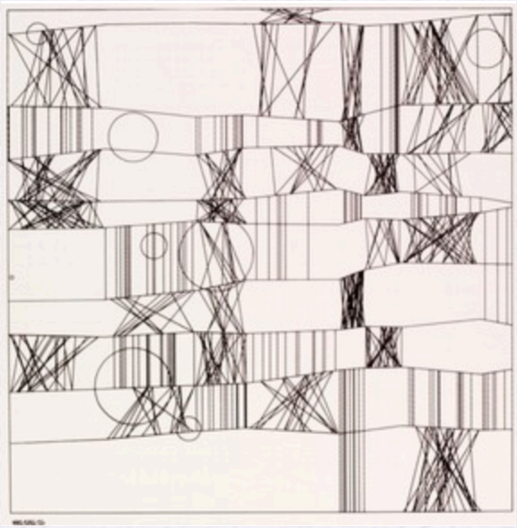
Early Photography

- With time and advances in technology, photography grew into a medium for art in its own right.

- Julia Margaret Cameron, *Mrs. Herbert Duckworth*, 1867, albumen silver print from glass negative (Metropolitan Museum of Art) https://www.khanacademy.org/humanities/becoming-modern/early-photography/a/julia-margaret-cameron-mrs-herbert-duckworth

Early software art

- So, too, software art and digital-born art have evolved as artists continue to work in new technologies.



- 'Hommage à Paul Klee 13/9/65 Nr.2', a screenprint of a plotter drawing created by Frieder Nake in 1965 (Victoria and Albert Museum: http://www.vam.ac.uk/content/articles/a/computer-art-history/)

Processing Class, Part 2

- In the first part of today's class, you experimented with drawings that you could have created with physical media such as crayons, paints, or ink.
- In the second part of today's class, I would like for you to take this opportunity to explore facets of drawing in *Processing* that you cannot replicate with familiar physical drawing materials.

Exploring Software Based Art

- What are some examples of software art that have gone beyond the realm of traditional media?
  - Algorithmic art: regular and repetitive drawing ideas that create or render a work which would be difficult or impossible or extremely time-consuming to create by hand.
  - Animation: drawings that shift and move
  - Interactive art: drawings that respond to the user (e.g. through a mouse)
  - And many more …

Algorithmic Art: using repetition with change

- Iteration (repetition) is widely used in computer programming. Computers are quite good at doing the same task over and over again.
- In algorithmic art, one can vary the parameters (either dramatically or subtly or anywhere in between) to render a drawing that results from such a “theme and variations” approach.

Simple Algorithmic Design: What does the following code produce?

```
size(400,400);
background(255);
stroke(0);
line(0,0,400,400);
fill(150,0,150);
ellipse(200,200,100,100);
/* setting up the ellipse parameters as integer variables: */
int x = 200;
int y = 200;
int w = 100;
int h = 100;
/* Using repetition and change: */
for (int i=1; i<6; i++) {
  x = x+25;
  y = y-25;
  w = w-20;
  h = h-20;
  ellipse(x,y,w,h);
}
```

This is stored on your drive as `sketch_4_repetition`.

Programming Interlude

- In programming terms, iteration is handled by loops. There are typically at least two kinds of loops in programming languages:
  - *for* loops for counted repetitions
  - *while* loops for conditional repetitions
- Conditions can be set using *if* statements or *if … else* statements so that the program is response to a condition.
- You will see these keywords used throughout programming and therefore throughout software-based art.

Drawing

- In order to move shapes and lines around a sketch, we will need to introduce two new functions to set the scene, so to speak:
  - `setup()` will allow us to define the starting condition for the work
  - `draw()` is used to render the work
- We will also use curly braces { … } to define blocks of code within each function*.
- Here is the previous drawing, rendered with these functions and saved as `sketch_4_usingFunctions`.

** Note that our `for` loop earlier also used braces to define that block of code.*

Using the `setup()` and `draw()` functions

```
void setup() {
  size(400,400);
  background(255);
}
```

```
void draw() {
  stroke(0);
  line(0,0,400,400);
  fill(150,0,150);
  ellipse(200,200,100,100);
  /* setting up the ellipse parameters as integer
variables: */
  int x = 200;
  int y = 200;
  int w = 100;
  int h = 100;
  /* Using repetition and change: */
  for (int i=1; i<6; i++) {
    x = x+25;
    y = y-25;
    w = w-20;
    h = h-20;
    ellipse(x,y,w,h);
  }
}
```

The “draw()” function handles some of the iteration for us!

- Sketch 5 (`sketch_5_movingLine`) yields a moving line. Here are questions for you:
  - What color is the line?
  - Is it horizontal or vertical now?
  - How would you change its direction?
  - Why does the line appear to move as a single line (rather than producing a drawing with many lines?)
  - Does the line go all the way across the window or just partway? How do you know?

Sketch 5: A Moving Line

```
int i = 0;
void setup() {
  size(400,400);
  stroke(0,0,255);
}
void draw() {
  background(255);
  line(0,i,400,i);
  i = i+2;
  if (i>400) {
     i=0;
  }
}
```

Sketch 5: A Moving Line (Annotated view)

```
int i = 0;                  // set up integer i
void setup() {              // open setup()
  size(400,400);            // window size
  stroke(0,0,255);          // use blue
}
void draw() {               // open draw()
  background(255);          // white background
  line(0,i,400,i);          // draw a line
  i = i+2;                  // increment i by 2
  if (i>400) {              // if i has reached
                            // the bottom …
     i=0;                   // reset i to zero
  }
}
```

Still images and animation

- Look closely at this line of code and where it is placed in the animated sketch:

```
background(255);
```

- In animation, one can reset the background to give the illusion of a moving shape or object.
- Therefore the placement of the code to set the background color (whether in the `setup()` function or in the `draw()` function) is one way to render the illusion of animation (which is actually a rapidly changing drawing.)

Interactive Images

- In order to build an interactive image (i.e. an image that the user can render or influence in some way), the program must accept input from a user or the environment.
- This can be done in many ways such as through a user manipulating a mouse or typing on the keyboard; or various peripherals to detect sound, motion, or environmental information.

Processing and your mouse

- *Processing* offers a number of functions to detect whether your user is manipulating her mouse.
- For example, `mouseX` and `mouseY` will detect the current *(x,y)* coordinates of a mouse and `mousePressed` will indicate if the user is currently clicking or pressing the mouse.
- You, as the artist/programmer, can make things happen where the user "tells" you to!

Responding to a mouse:

- In this program (`sketch_6_mouse`), the user "draws" with a purple circle:

```
void setup() {
  size(400,400);
  background(255);
  fill(150,0,150);
}
void draw() {
  ellipse(mouseX,mouseY,50,50);
}
```

Responding to a mouse:

- Annotated version:

```
void setup() {          // setup
  size(400,400);        // window size
  background(255);      // white background
  fill(150,0,150);      // purple color
}
void draw() {
  // draw a circle at the (x,y)
  // coordinates designated by the mouse
  ellipse(mouseX,mouseY,50,50);
}
```

One can change colors as well using an “if” clause:

```
void setup() {
  size(400,400);
  background(255);
}
void draw() {
  // change color … depending on whether
  // the mouse is pressed or not!
    if  (mousePressed) {
        fill(150 ,0,150); }
    else {
        fill(0,0,255);    }
  ellipse(mouseX,mouseY,50,50);
}
```

Now its your turn!

- Render your earlier drawing (or start a new one) with the addition of animation or user interactivity.
- Be sure to ask me or any of our talented teaching assistants any questions that you might have.
- Above all, have fun!

Bibliography

The Processing Site

- Opening Page https://processing.org
- Tutorials https://processing.org/tutorials/
- Examples https://processing.org/examples/ or … *open Processing and navigate to FILE / EXAMPLES.*
- Exhibition *https://processing.org/exhibition/*

Bibliography

Exhibitions and Inspiration

- OpenProcessing
  http://www.openprocessing.org
- OpenProcessing Data Visualizations
  http://www.openprocessing.org/collection/1122
- Ben Fry's Project Page
  http://benfry.com/projects/

Bibliography

Processing and Programming Languages

- Processing in Java
  https://processing.org
- Python mode for Processing
  http://py.processing.org
- Processing and JavaScript
  http://processingjs.org

Prof. Deena Engel
Clinical Professor
Associate Director of Undergraduate Studies for the
Computer Science Minors programs
Department of Computer Science
Courant Institute of Mathematical Sciences
New York University
deena.engel@nyu.edu