Exploring Processing

Georgia Tech

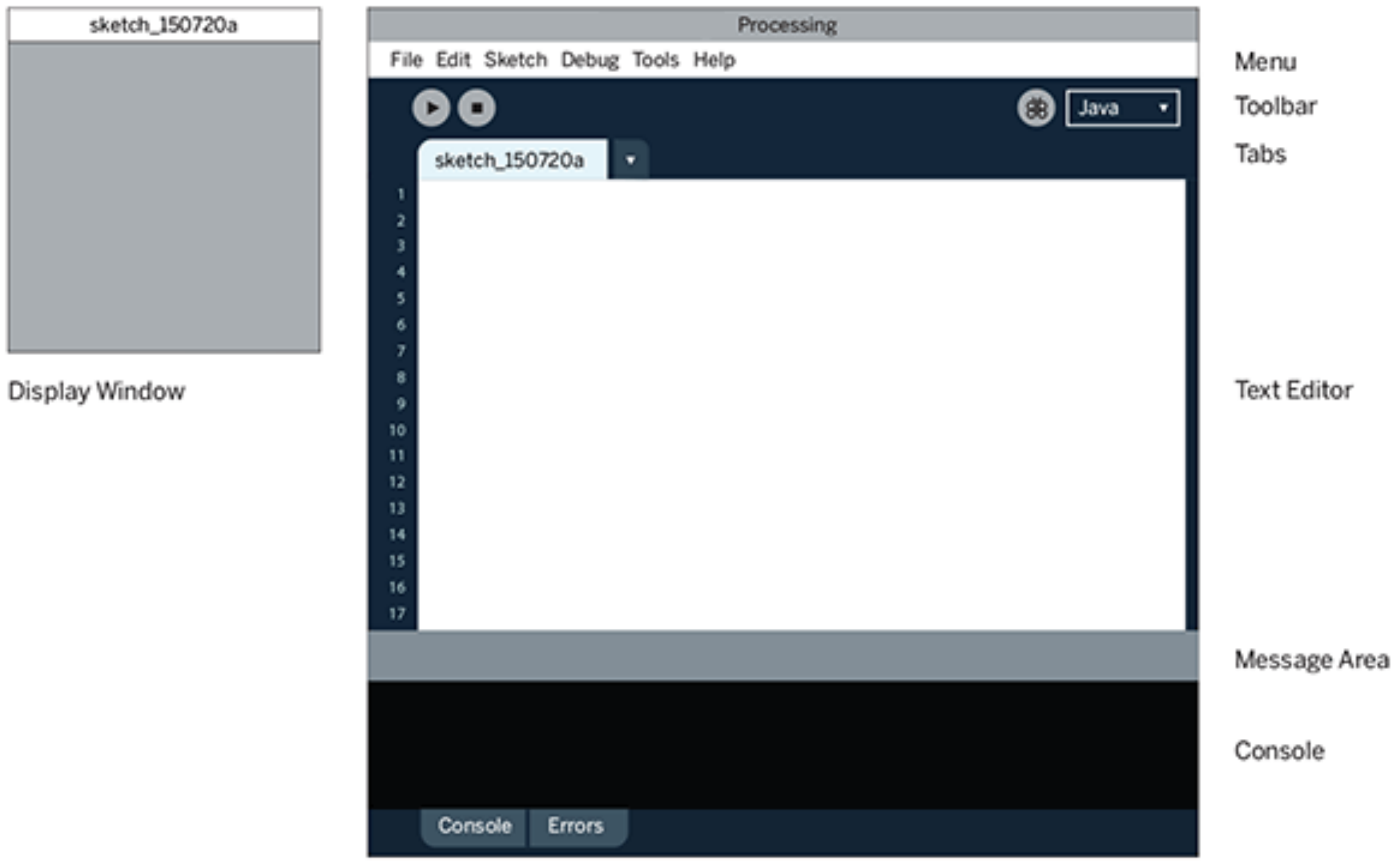
What is Processing?

- Easy-to-use programming environment
  - Let's you edit, run, save, share all in one application
- Designed to support interactive, **visual** applications
  - Something we've been missing so far in Python…
- Simplified Java-like syntax (in its default form)
  - Other languages available via plugins
- Useful for Arduino micro controller programming via special libraries ("Wiring")

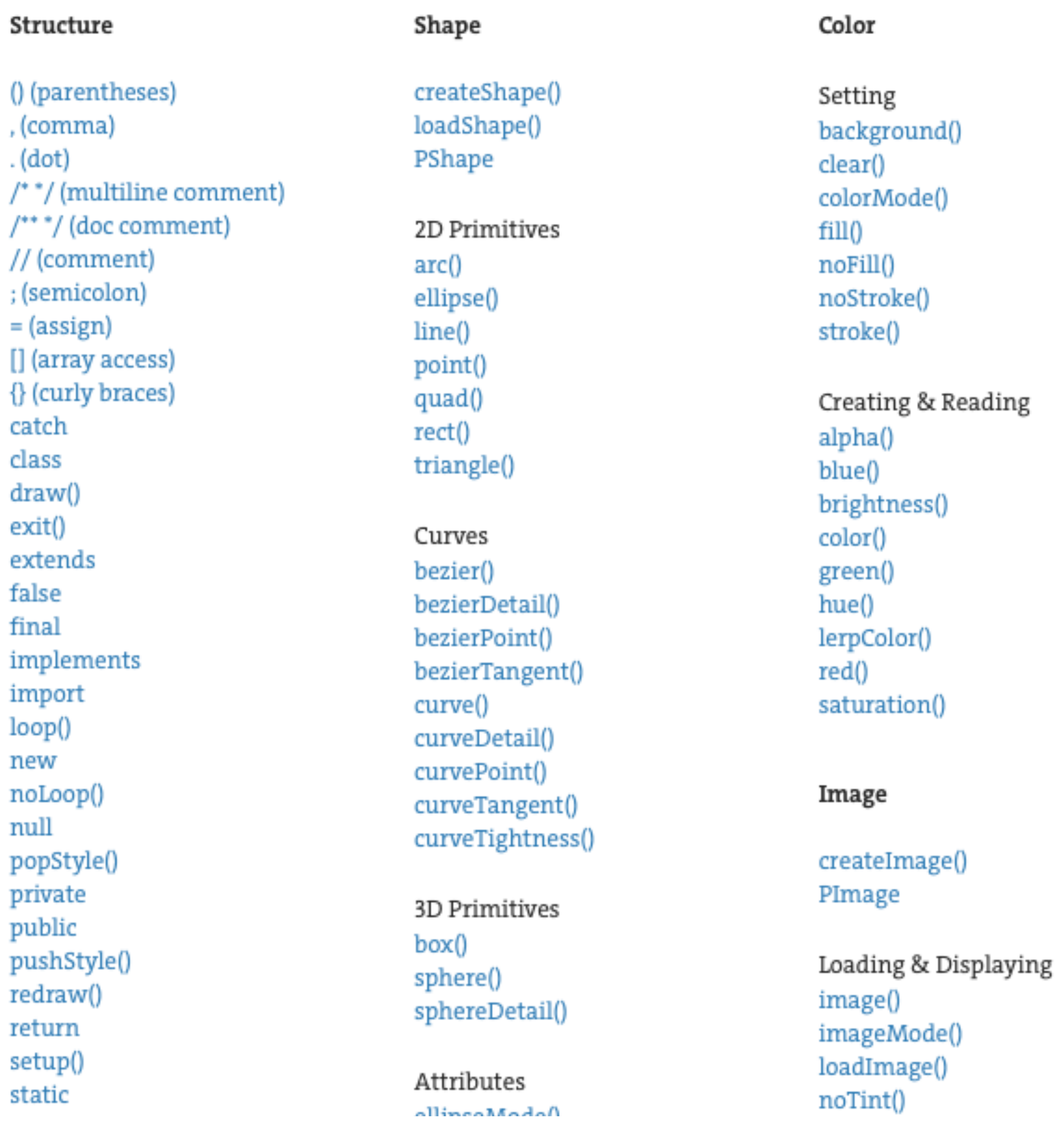
First stop…

PROCESSING.ORG

The Processing Development Environment



API for graphics, interactivity, etc.



Structure

() (parentheses)
, (comma)
. (dot)
/* */ (multiline comment)
/** */ (doc comment)
// (comment)
; (semicolon)
= (assign)
[] (array access)
{} (curly braces)
catch
class
draw()
exit()
extends
false
final
implements
import
loop()
new
noLoop()
null
popStyle()
private
public
pushStyle()
redraw()
return
setup()
static

Shape

createShape()
loadShape()
PShape

2D Primitives
arc()
ellipse()
line()
point()
quad()
rect()
triangle()

Curves
bezier()
bezierDetail()
bezierPoint()
bezierTangent()
curve()
curveDetail()
curvePoint()
curveTangent()
curveTightness()

3D Primitives
box()
sphere()
sphereDetail()

Attributes

Color

Setting
background()
clear()
colorMode()
fill()
noFill()
noStroke()
stroke()

Creating & Reading
alpha()
blue()
brightness()
color()
green()
hue()
lerpColor()
red()
saturation()

Image

createImage()
PImage

Loading & Displaying
image()
imageMode()
loadImage()
noTint()

Getting started with Processing

- Programs are called "sketches" in Processing's terminology
- Saved in the "sketchbook"

- Enter our first Processing program:

  - line(10, 10, 50, 50);

- NOTE the semicolon!!

Getting started with Processing

```
size(400, 400);              // set the window size
background(192, 64, 0);      // background color
stroke(255);                 // pen color to white
line(100, 25, 250, 350);     // X1, Y1,   X2, Y2
```

Colors in Processing

Lots of variants for controlling color:

```
stroke(255);                 // sets the stroke color to white
stroke(255, 255, 255);       // identical to the line above
stroke(255, 128, 0);         // bright orange (red 255, green 128, blue 0)
stroke(#FF8000);             // bright orange as a web color
stroke(255, 128, 0, 128);  // bright orange with 50% transparency
```

By default, colors are specified in the range 0-255 (8 bits for each of R, G, and B

Same variants work for fill(), background(), …

Functions that affect drawing properties affect all objects drawn to the screen until the next fill, stroke, etc.

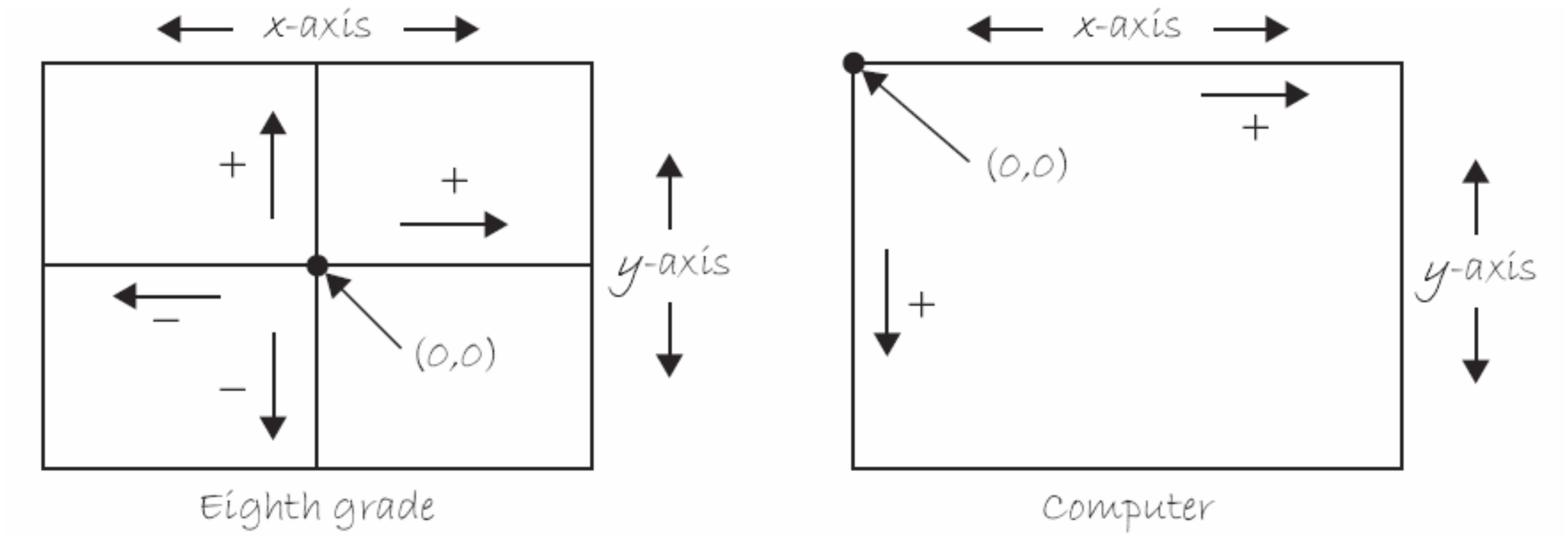
See Tools > Color Selector

More Simple Graphics

Drawing something a little more complicated…

```
background(173, 216, 230);
stroke(0);
fill(120,82,82);
size(300, 300);
rect(100, 200, 100, 80);
triangle(100, 200, 200, 200, 150, 100);
fill(255);
textSize(32);
textAlign(CENTER);
text("TECH", 150, 200);
```

A note on coordinates…



Moving Beyond Static Sketches

- Programs that are simple lists of statements are called **static sketches**
  - No animation, no interaction
- Interactive programs are drawn as a series of frames.
  - Add functions setup() and draw() - these will be called automatically

Example

```
void setup() {
  size(400, 400);
  stroke(255);
  background(192, 64, 0);
}

void draw() {
  line(150, 25, mouseX, mouseY);
}
```

Called once. size() should be the first line inside

Called repeatedly.

Note Java-style curly braces and declaration of return parameter (void) !

Example (cont'd)

How would you change this so that you don't have multiple lines drawn over the top of each other?

More complicated event handling

```
void setup() {
  size(400, 400);
  stroke(255);
  background(192, 64, 0);
}

void draw() {
  line(150, 25, mouseX, mouseY);
}

void mousePressed() {
  background(192, 64, 0);
}
```

More Simple Graphics: Text

```
PFont myFont;

void setup() {
  myFont = createFont("Georgia", 32);
}

void draw() {
  textFont(myFont);
  textAlign(CENTER, CENTER);
  text("Hello, World!", width/2, height/2);
}
```

NOTE special variables width, height
PFont is the type of a Processing font object

Interactivity in Processing

Special variables **mouseX** and **mouseY** contain the coordinates of the cursor relative to the origin

```
void setup() {
  size(100, 100);
  noStroke();
}

void draw() {
  background(126);
  ellipse(mouseX, 16, 33, 33);   // Top circle
  ellipse(mouseX/2, 50, 33, 33); // Middle circle
  ellipse(mouseX*2, 84, 33, 33); // Bottom circle
}
```

Values set to 0,0 until the pointer enters the window

Interactivity in Processing

pmouseX and **pmouseY** store the mouse values from the previous frame

Programming challenge: write a program that draws a stroke as the user moves the mouse around the screen

Programming challenge

Programming challenge: write a program that draws a stroke as the user moves the mouse around the screen

How do you stop the program from drawing the first (bogus) segment from 0,0? Hint: maybe a conditional?

How would you change the program so that it only draws when the mouse button is held down? Hint: special variable **mousePressed** will be true when button is pressed.

Event variables

mousePressed — will be true or false
mouseButton — will be LEFT, RIGHT, CENTER
keyPressed — true while key is actively being held down
key — holds a single alphanumeric character, the most recently pressed key (can draw to the screen using text()). Can also be used as a numeric ASCII value (A=65, etc.). Special values BACKSPACE, TAB, ENTER, RETURN, …
keyCode — if key == CODED, then keyCode contains special key info: ALT, CONTROL, SHIFT, UP, DOWN, LEFT, RIGHT

Events

- An **event** is a type of function that's called automatically by Processing when a user input occurs. These functions "handle" the user input.
  - Sometimes called callbacks, event handlers, listeners, … in other programming languages
- Called **asynchronously**: may happen at any time, may never happen at all, outside the normal flow of control of your program
  - More detailed answer: user inputs are queued until draw() finishes, then the event functions are called to handle any user inputs that occurred in the meantime
- The code inside the event function is run once, each time the corresponding user input occurs

Mouse Events

- mousePressed()
- mouseReleased()
- mouseMoved()
- mouseDragged()

- (mouseMoved() and mouseDragged() not called if the pointer stays in the same place on the screen)

- How do these relate to the variables mousePressed, etc?
- Value of mousePressed is true until the button is released… can be used within draw().
- mousePressed() function only runs once when a button is pressed… useful for triggering actions.

Dealing with Asynchrony

- In general:
- It's not a good idea to draw inside an event function: keep that code inside draw()
- Why? Because any drawing you do inside an event handler will get clobbered whenever draw() is called next (unless you have an empty draw() function).

Dealing with Asynchrony (cont’d)

- So how would you draw something in response to mouse events?
- Need to think about structuring your program a little differently…

- Event handler functions record details about the new thing that should be drawn…
- … draw() function then draws it the next time it is called.

- Commonly: event functions will set some variables indicating what to draw, and your code in the draw function checks these the next time through.

Key Events

- Similar setup as mouse events:
- keyPressed()
- keyReleased()
- Can check value of **key** variable inside these.

Under the Hood…

- If your program has a draw() function, it'll be called 60 times/second
  - Use frameRate() to change
- noLoop() pauses the draw loop; loop() restarts it
  - Event functions still get called when noLoop() is in effect
  - You rarely have to use these unless you're doing something weird
- Use redraw() to cause the code in draw() to be run one time. Often called from within an event function

More Processing: Strings

- String msg = “This is my string. There are many like it but this one is mine.”
  - (Remember variables have types that must be declared)
- msg.length();
- String upper = msg.toUpperCase(); println(upper);
  - (Strings are immutable, as in Python)
- Comparison: safest way is str1.equals(str2)

More Processing: Strings

- Concatenation:
  - String hw = “Hello” + “World”;
  - int x = 10;
    String msg = “The value of x is” + x;
- Printing to the console (for debugging):
  - println(msg);

More Processing: Arrays

- Similar to Python lists, with a few important exceptions…
  - Can (generally) only store homogenous data
  - After declaring it, create it with the keyword **new**
  - Fixed size

- `int[] data;`
- `data = new int[3];`
- `data[0] = 19;`
- `data[1] = 42;`
- `data[2] = 101;`
- `OR, just int[] data = {19, 42, 101};`

More Processing: Arrays

- length, square-bracket notation, and iteration

```
println(data.length);
data[0] = data[1] + data[2];

for (int i=0 ; i<data.length ; i++) {
    println(data[i]);
}
```

More Processing: Arrays

- append() - creates and returns a new array with the parameter date added

```
String[] trees = {“ash”, “oak”};
// INCORRECT! Doesn’t change the array
append(trees, “maple”);

// Create a new array, re-use trees to refer to it
trees = append(trees, “maple”);

printArray(trees);
```