

# **Introduction to Processing**

Copyright (c) Pearson 2013.  
All rights reserved.

# Processing

- Processing started by Ben Fry and Casey Reas while both were graduate students at MIT Media Lab in 2001.
- Processing is Java.
- Designed for visual artists with limited programming experience who want to create art without knowing complicated Java syntax.
- In its current version, hundred of libraries have been written for computer vision, data visualization, music composition, networking, 3D drawings and programming electronics.

# Structure of a Sketch

A sketch or program in processing only needs two methods: setup and draw.

`setup()` is similar to a constructor, it initializes variables. `draw()` is similar to the main method.

```
void setup() {  
...  
}
```

```
void draw() {  
...  
}
```

# setup()

//declare **global** variables on top

//setup() is called **automatically** to initialize variables, **runs ONCE**.

// it is good practice to initialize variables here, **calling it explicitly** later for  
// example can reset your game.

```
void setup() {
```

```
...
```

```
}
```

```
void draw() {
```

```
...
```

```
}
```

# draw()

```
void setup() {  
...  
}
```

// draw is called **automatically 60 times a second**(similar to an infinite // while loop). If you draw an object then update it, draw will animate // the object.

```
void draw() {  
...  
}
```

# File System

When a sketch is saved, Processing will create new folder whose name matches the name of the sketch(without the .pde extension). For example, if I save the sketch below as Game.pde, then Game.pde is in a folder called Game.

```
void setup() {
```

```
...
```

```
}
```

```
void draw() {
```

```
...}
```

# Example

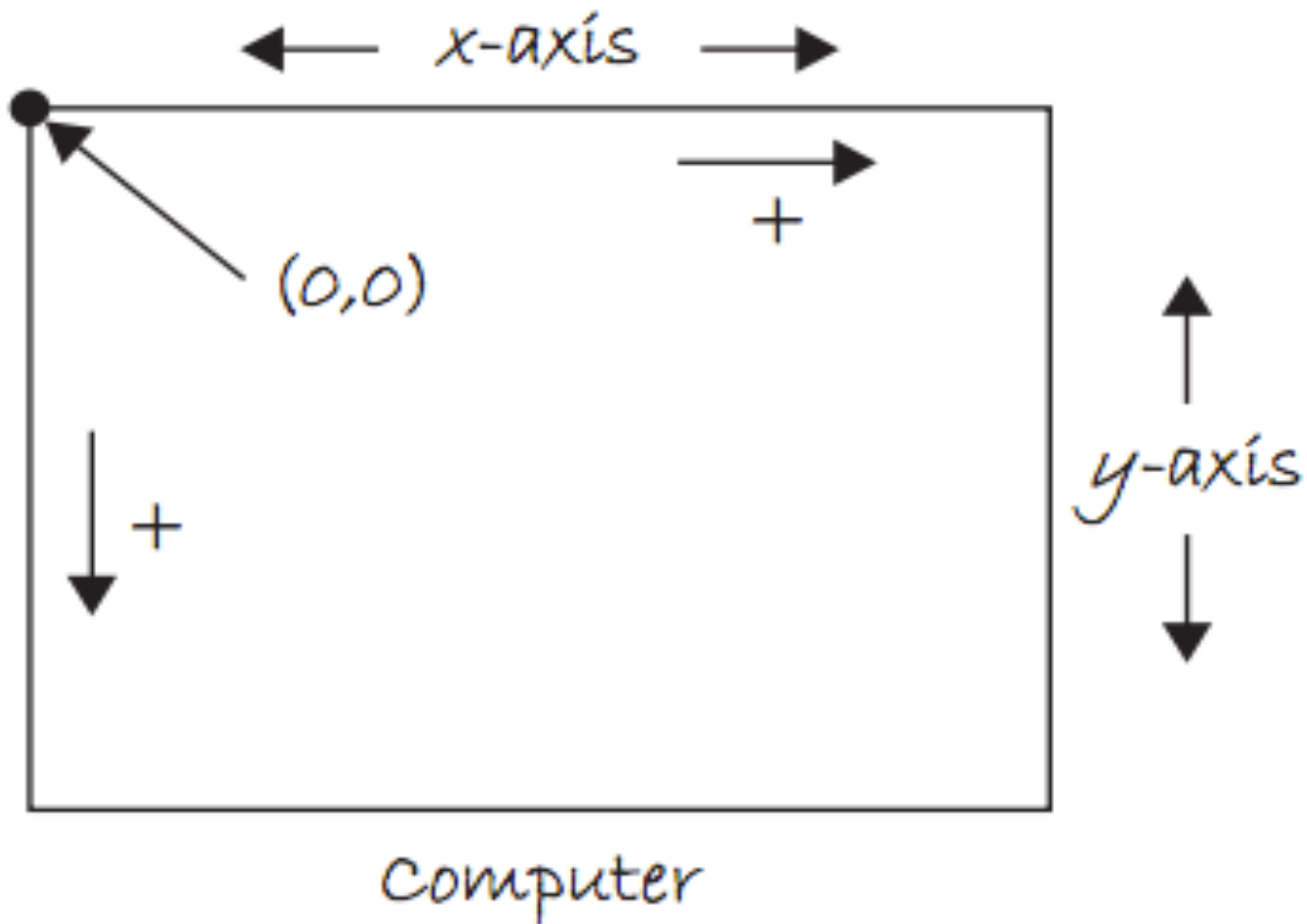
```
int x, y; //declare global variables

//initialize them in setup().
void setup(){
    x = 5;
    y = 10;
}

// modify and update them in draw().
void draw(){

    x += 1; //x increases by 1, 60 times per second.
}
```

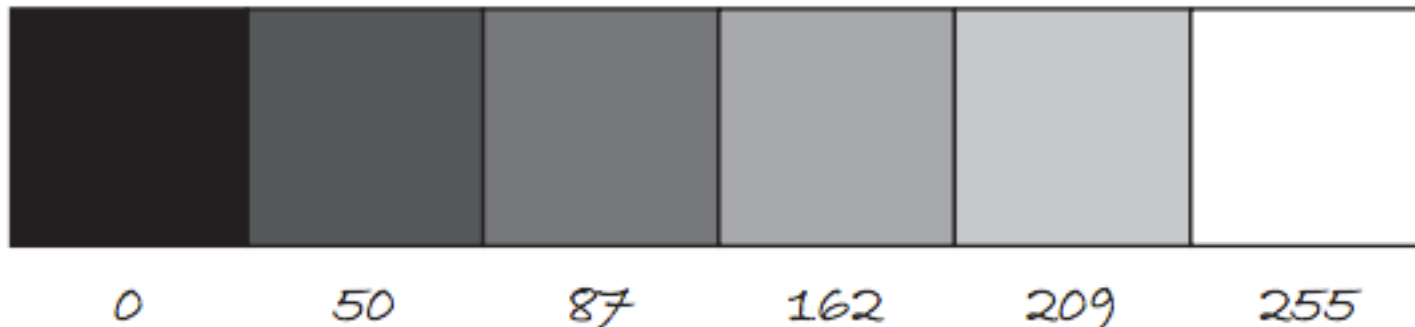
# Coordinate System





# Color

- Color is defined by a range of numbers.
- In grayscale, 0 is black, 255 is white and any color in between is a shade of gray ranging from black to white.



# Color

- RGB Color is determined by three parameters. Each parameter is in the range 0-255. The first indicates the degree of red(R), the second the degree of green(G) and the last the degree of blue(B).

- Red + green = yellow
- Red + blue = purple
- Green + blue = cyan (blue-green)
- Red + green + blue = white
- No colors = black

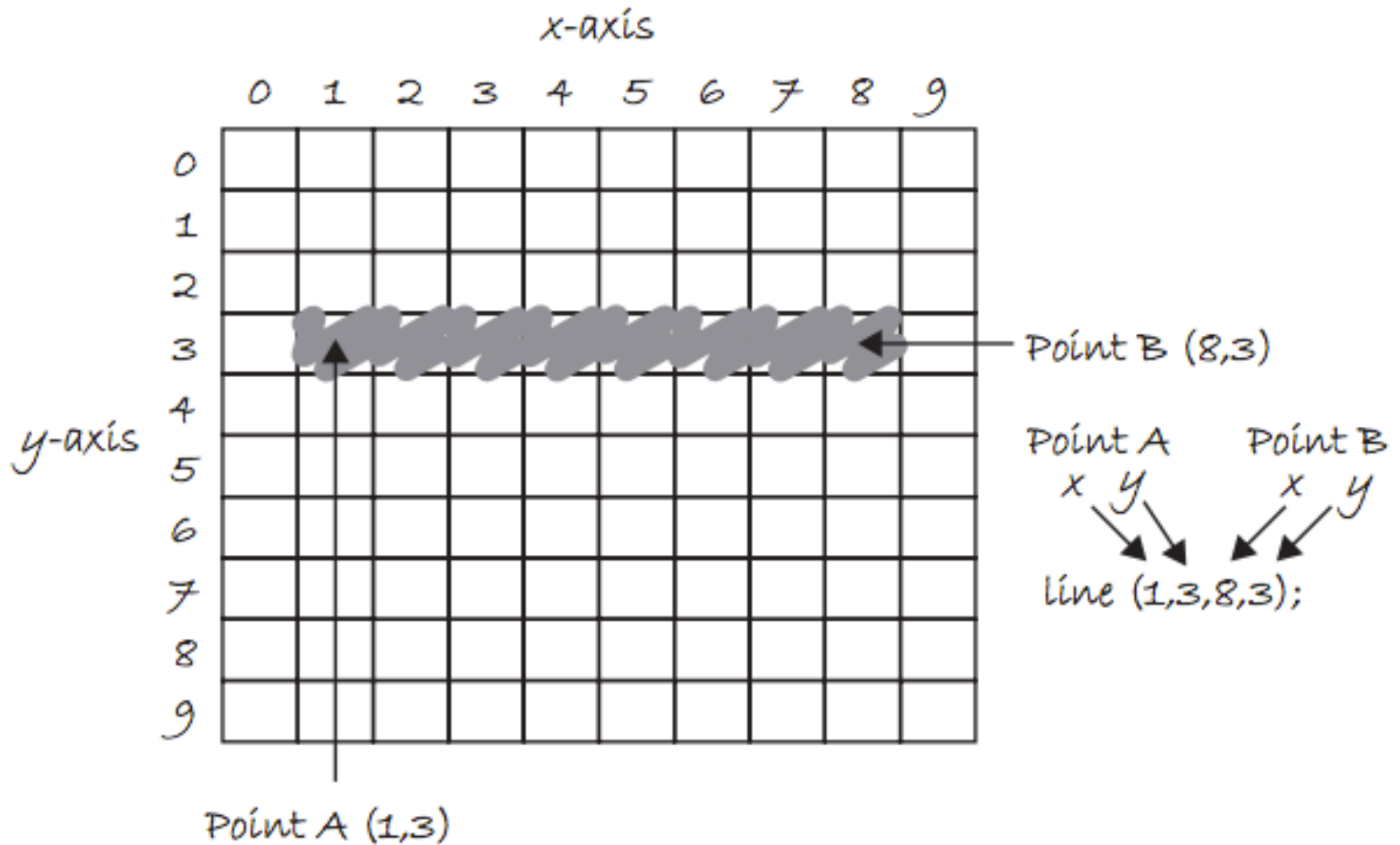
# Some Methods

- **size(int width, int height)**: Set the size of the window. This should go in `setup()`. `width` and `height` are reserved words referring to the size of the screen.
- **background(int range)**: `range` can be any integer from 0=black to 255=white. This should go in `draw()`.
- **background(int r, int g, int b)**: Background with RGB values. This should go in `draw()`.

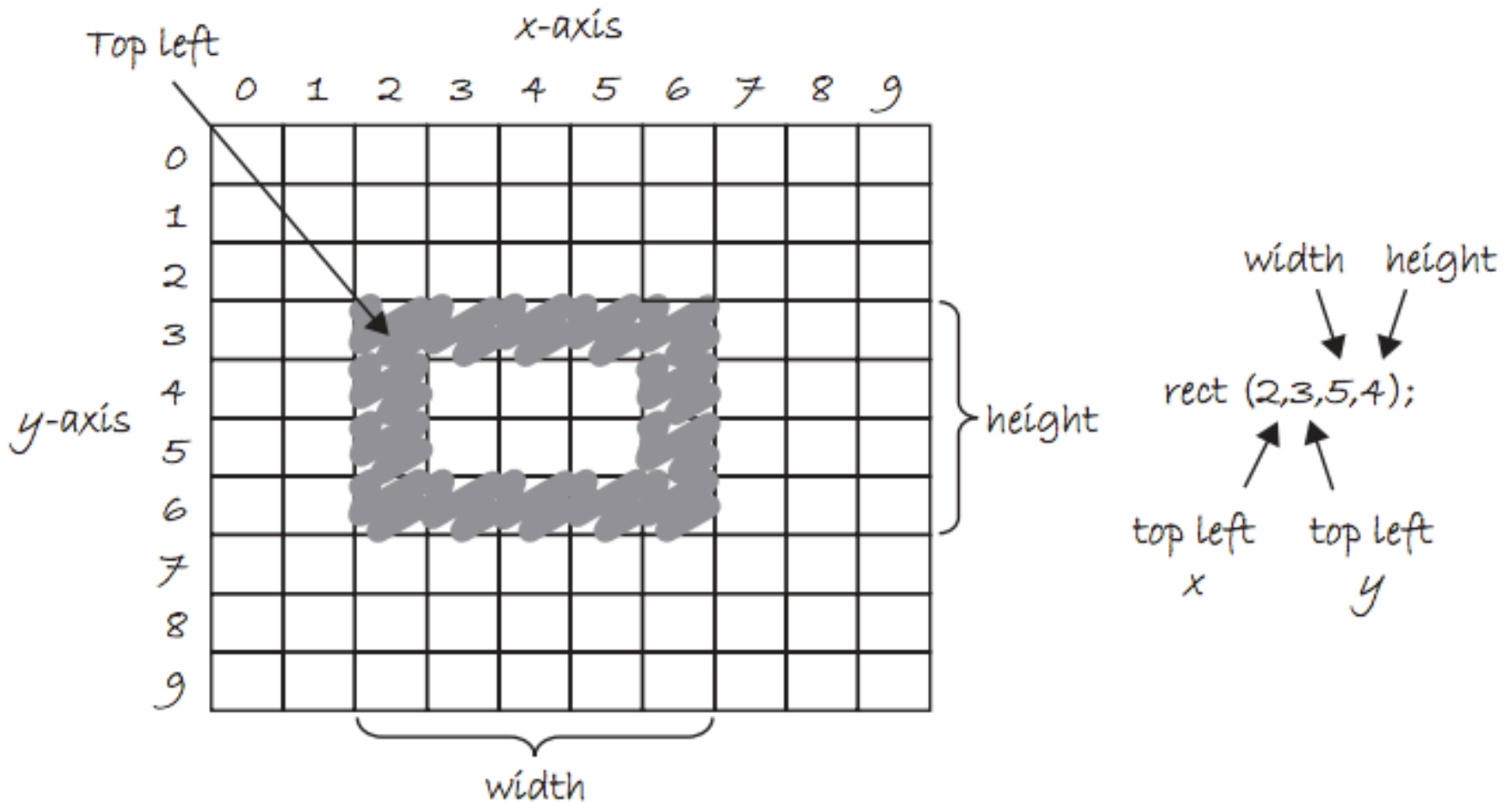
# Some Methods

- `fill(int r, int g, int b)` : By calling fill BEFORE a shape will set the color of the shape. Call it again before drawing another shape to change color.
- `strokeWeight(int pixel)` : Call BEFORE drawing a shape will set thickness of the boundary.
- `line(int x1, int y1, int x2, int y2)` :
- `ellipse(int x, int y, int width, int height)` : center of ellipse is (x, y); width and height are the lengths of the axes.
- `rect(int x, int y, int width, int height)` : position is top left corner by default.

# Draw a Line



# Draw a Rectangle



# Example 1

```
int x,y;
void setup() {
    size(800,600); //800 pixels by 600 pixels
    x = width / 2; //width=800, height=600
    y = height / 2; }

void draw() {
    background(255); //white background
    fill(255, 0, 0); //red fill
    ellipse(x, y, 80, 80); //red-filled circle radius 40
    //continued on the next slide...
```

# Example 1 Continued..

```
fill(0, 255, 0); //green fill
ellipse(x, y, 80, 80); //green-filled circle radius 40
fill(0, 0, 255); //blue fill
//blue filled rectangle at (40,50) width=60,height=70
rect(40, 50, 60, 70);
}
```



# Example 2: Make it move.

```
int x, y, xspeed = 5;
void setup() {
    size(600, 400);
    x = width/2;
    y = height/2;}
// make a red circle moves horizontally
// across the screen
void draw() {
    background(255); //white background
    fill(255, 0, 0); //fill red
    ellipse(x, y, 80, 80);
    x += xspeed; //moves 5*60=300 pixels per sec.
}
```

# Processing (keyPressed)

The `draw()` loop runs continuously until it is interrupted by an event, for example, a keyboard or mouse event.

If a key is pressed, `draw()` temporarily halts, Processing then jump execution to the `keyPressed()` function, runs the function's code then return control to the `draw()` loop. The key that is pressed is store in the `key` variable.(or `keyCode` if it is a special character, e.g. UP, DOWN, LEFT, RIGHT).

Similarly, if a key is released, `keyReleased()` is called.

# Processing (keyPressed)

```
void setup() {...}
```

```
void draw() {...}
```

```
// call automatically IF a key is pressed.
```

```
void keyPressed() {
```

```
    if(key == 'a') { //for letter keys
```

```
        //code to run if 'a' is pressed.
```

```
    }
```

```
    if(keyCode == UP) { //for arrow keys: UP, DOWN, LEFT, RIGHT
```

```
        //code to run if UP is pressed.
```

```
    }
```

```
}
```

```
void keyReleased()
```

```
}
```

# Processing: Mouse Events

Processing keeps track of the position of the mouse at any given time through the variables `mouseX`, `mouseY`.

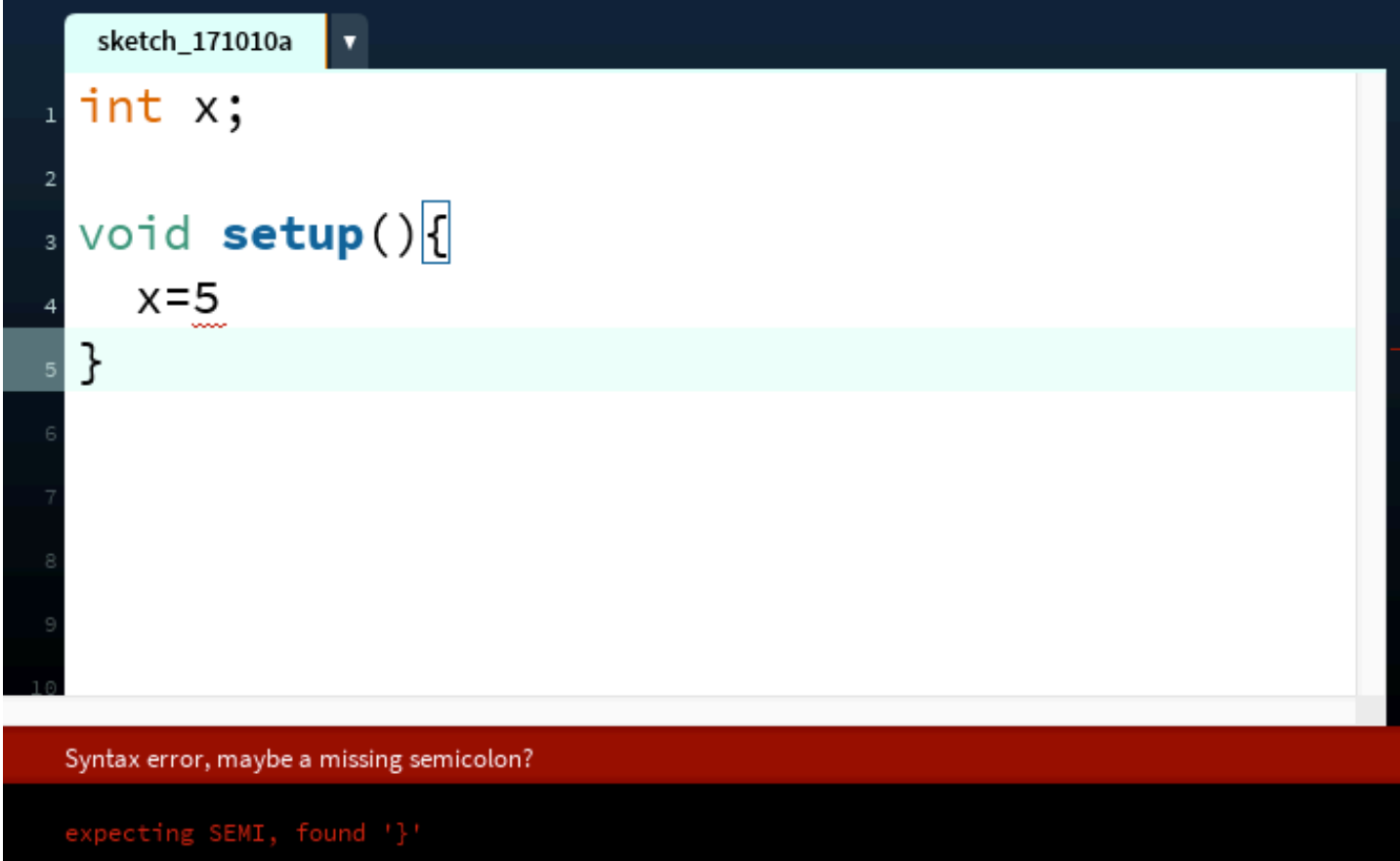
Similar to `keyPressed`, which responds to keyboard inputs, `mouseClicked` is a function that can be implemented to respond to the mouse being clicked. IF the mouse is clicked, `mouseClicked` is called and code can be implemented to respond to the click.

# What does it do?

```
int blue = 0;
void setup(){
    size(600, 600);
}
void draw(){
    background(255);
    fill(255, 0, blue);
    ellipse(mouseX, mouseY, 100, 100);
}
void mouseClicked(){
    if (blue == 0)
        blue = 255;
    else
        blue = 0;
}
```

# The Console

- The **console** refers to the box at the bottom of Processing. It is generally used to print error messages in the code.



The screenshot shows the Processing IDE interface. At the top, a tab is labeled 'sketch\_171010a'. The main area is a code editor with the following code:

```
1 int x;  
2  
3 void setup(){  
4   x=5  
5 }  
6  
7  
8  
9  
10
```

The code is syntactically incorrect because the assignment 'x=5' on line 4 is missing a semicolon. The IDE's console at the bottom displays the error message: 'Syntax error, maybe a missing semicolon?' followed by 'expecting SEMI, found '}''.

# The Console

Messages can be printed on the console(for error-checking purposes, etc..) by using the command `println()` or `print()`.

```
println(4);  
println(4 + 3/2);  
println("Hello, world");
```

# float vs double

- **float:** 32-bit data type representing real numbers to about 7 decimal places.(Not on AP Exam)
- **double:** 64-bit data type representing real numbers to about 16 decimal places.

**Since Processing is graphics intensive, it is recommended that floats are used instead of doubles.** In fact, float is the default type for decimal numbers in Processing. All of the built-in math functions in Processing returns floats. The math functions from the standard Math library in Java returns doubles.

*//sqrt() is Processing's version of Math.sqrt(). Both available in Processing*

```
double a = Math.sqrt(9); // a = 3.0 but is a double
```

```
float b = sqrt(9); // b = 3.0 but is a float
```



# Download Processing

Download Processing in your computer at home!

Your computer is probably a 64-bit computer if it's fairly recently. Use the 64-bit version.

<http://www.processing.org>

# Things to Try

Lab 1: Make the ball move horizontally back and forth in the middle of the screen, bouncing each time it hits the left or right side of the screen.

Modify your code so that it is moving in a diagonal direction and bouncing when it hits any side of the screen.

# Things to Try

Lab 2: Use `mouseX` and `mouseY` to write a program that draw a red ball which follows your mouse.

# Things to Try

Lab 3: Create a window of size 800p by 600p. Use for loops to draw horizontal and vertical lines every 100 pixels.

# Things to Try

Lab 3: Draw a rectangle at the center of the screen. Let the user move the rectangle up, down, left, right using the keyboard arrow keys. Hint: It is better (more smooth) if you use both `keyPressed` and `keyReleased`.