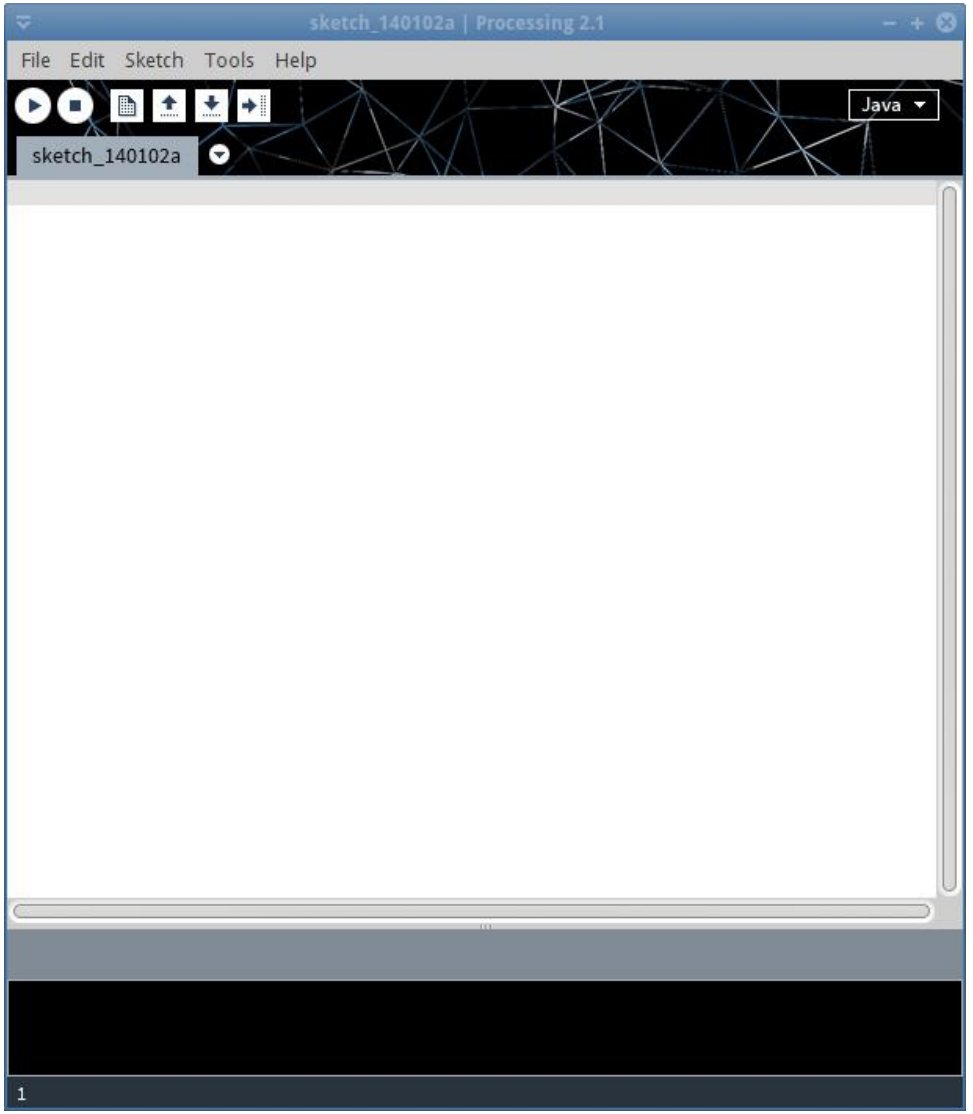
Processing

Introduction to Processing

- Processing is a programming environment that makes writing programs easier.
- It contains libraries and functions that make interacting with the program simple.

- The development software to be used is called Processing.
- Processing Is a programming language and development environment using Java.
- Processing makes developing software easier.
- More information can be found at processing.org. The software is free to download and use.
- The development environment is similar to the Arduino development environment.



First Program

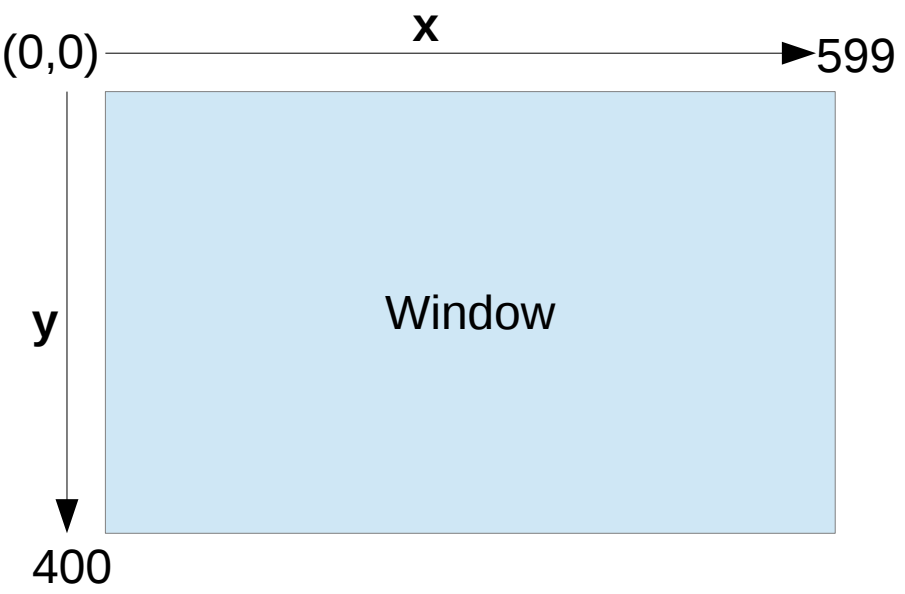
- Try the program to the right.
- It's structure is the same as the Arduino software.
- The one small difference is that **`loop()`** is now **`draw()`**.
- **`Draw()`** behaves the same as **`loop()`** and is executed repeatedly.

```
void setup()
{
  size(600,400);
}

void draw()
{
  ellipse(300,200,80,80);
}
```

First Program

- In the **`setup()`** function, there is a function called **`size()`**.
- Size creates a window with the size of the window specified in pixels.
  - The first number is the width.
  - The second number is the height.
- The window can be thought of as a graph with the 0,0 coordinate in the upper left hand corner. The value of X increases from left to right. The value of Y increases from top to bottom.

```
void setup()
{
  size(600,400);
}

void draw()
{
  ellipse(300,200,80,80);
}
```



First Program

- In the draw() function is a function called ellipse(). This function draws a circle or ellipse.
  - The first number is the x position.
  - The second number is the y position.
  - The third number is the width of the ellipse.
  - The fourth number is the height of the ellipse.

```
void setup()
{
  size(600,400);
}

void draw()
{
  ellipse(300,200,80,80);
}
```

Adding to the First Program

- The color of the ellipse can be changed with the function **`fill()`**.
- Add the function **`fill()`** before the **`ellipse()`** function.
- The ellipse should now be nearly black.
- Change the number to any number in the range of 0 to 255 and run again.
- The number sets the gray intensity level of the ellipse from black to white.

```
void setup()
{
  size(600,400);
}

void draw()
{
  fill(10);
  ellipse(300,200,80,80);
}
```

Adding to the First Program

- Change the **`fill()`** function parameters with the numbers shown to the right and run again.
- When the **`fill()`** function gets three numbers, it knows to set the ellipse to the specified color.
  - The first number is the intensity of red.
  - The second number is the intensity of green.
  - The third number is the intensity of blue.

```
void setup()
{
  size(600,400);
}

void draw()
{
  fill(200,50,200);
  ellipse(300,200,80,80);
}
```

Interacting with the Program

- The mouse can be used to interact with the program. The position of the mouse can be used by using two variables.
  - **mouseX**
  - **mouseY**
- The variables are case sensitive. Replace the X and Y coordinates in ellipse with **mouseX** and **mouseY**.
- Run the program and see what happens.

```
void setup()
{
  size(600,400);
}

void draw()
{
  fill(200,50,200);
  ellipse(mouseX,mouseY,80,80);
}
```

Interacting with the Program

- Now at the beginning of the **`draw()`** function, add the function **`background()`**.
- Run the program again.
- This time there is only one ellipse that is moved around.
- When **`background()`** is used, it clears the screen of anything displayed previously for a new start.

```
void setup()
{
  size(600,400);
}

void draw()
{
  background(50);
  fill(200,50,200);
  ellipse(mouseX,mouseY,80,80);
}
```

Keyboard Interaction

- This code will use the keyboard to change the color of the ellipse.
- Processing has a variable that indicates what key has been pressed.
- The comparison is made against a character and has to be in single quotes.
- Try running the program.
- Add more key selections and other colors.

```
void setup()
{
  size(600,400);
}

void draw()
{
  background(50);
  if(key == 'a')
    fill(200,50,200);
  else if(key == 'b')
    fill(0,255,0);
  ellipse(mouseX,mouseY,80,80);
}
```

Event Based Programming

- Processing is an event based programming environment. This means certain types of functions execute when specific events occur.
- This example shows the function **`keyPressed()`** which is executed when a key is pressed on the keyboard. It is not called from any other part of the program.
- This program uses the arrow keys. This requires the use of another processing variable called **`keyCode`**.
- Run this program.

```
void setup()
{
  size(600,400);
}

void draw()
{
  background(50);
  ellipse(mouseX,mouseY,80,80);
}

void keyPressed() {
  switch(keyCode) {
    case UP : fill(0,255,0);
              break;
    case DOWN : fill(255,0,0);
              break;
    case LEFT : fill(0,0,255);
              break;
    case RIGHT : fill(255,0,255);
                 break;
  }
}
```

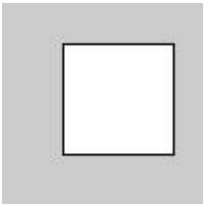
Event Based Programming

- Another event function is **`keyReleased()`**. This is executed when the key on the keyboard is let go.
- This program turns the ellipse black when a key is not pressed.
- Run this program.
- Save this program. It will be used again.

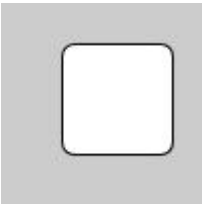
```
void setup()
{
  size(600,400);
}

void draw()
{
  background(50);
  ellipse(mouseX,mouseY,80,80);
}

void keyPressed() {
  switch(keyCode) {
    case UP : fill(0,255,0);
              break;
    case DOWN : fill(255,0,0);
              break;
    case LEFT : fill(0,0,255);
              break;
    case RIGHT : fill(255,0,255);
                 break;
  }
}

void keyReleased() {
  fill(0,0,0);
}
```

Other 2D Primitives

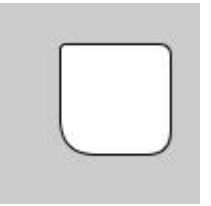
- Rectangle
  - rect(x,y,width,height);



  - rect(x,y,width,height,curve);



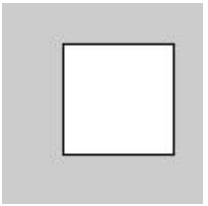
  - rect(x,y,width,height,topleft,topright,bottomright,bottomleft);



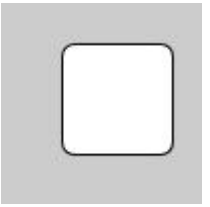
  - The x,y coordinates are the top left corner of the rectangle. The width specifies how large the rectangle is to the right. The height specifies how large the rectangle is in the downward direction.

Other 2D Primitives

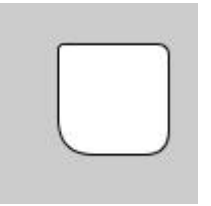
- Rectangle
  - rect(x,y,width,height);



  - rect(x,y,width,height,curve);



  - rect(x,y,width,height,topleft,topright,bottomright,bottomleft);



  - The curve specifies how many pixels the corners of the rectangle are rounded. The last form lets you specify how much each corner is curved.

Other 2D Primitives

- Triangle
  - triangle(x1,y1,x2,y2,x3,y3);
  - Each coordinate specifies a corner of the triangle. There is no required order for the coordinates.
  - Write a program to generate a triangle and set the first two coordinates to a fixed location.
  - Set the third coordinates to **mouseX** and **mouseY**.
  - Run the program.
  - Insert background(0) in the beginning of **draw()** and run again.

Other 2D Primitives

- Quadrilateral
  - quad(x1,y2,x2,y2,x3,y3,x4,y4);
  - The coordinates specify the four corners of the quadrilateral.

Displaying Text

- The function **`text()`** is used to display text in the window.
- The program to the right displays the mouse coordinates at the top left corner of the window.
- The values 20,20 are the x and y coordinates. The coordinates is the top left corner of the text.

```
void setup()
{
  size(600,400);
}

void draw()
{
  background(50);
  text("Mouse: " + mouseX + " " + mouseY,20,20);
  fill(200,50,200);
  ellipse(mouseX,mouseY,80,80);
}
```

Displaying Text

- The first arqument in the **text()** function is the text to be displayed.
- The word **Mouse:** is displayed.
- The values of the mouse position is added to the end of **Mouse:**.
- Try it out.

```
void setup()
{
  size(600,400);
}

void draw()
{
  background(50);
  text("Mouse: " + mouseX + " " + mouseY,20,20);
  fill(200,50,200);
  ellipse(mouseX,mouseY,80,80);
}
```

Displaying Text

- Need the text to be larger? Use the function **textSize()**.
- Try out different sizes.

```
void setup()
{
  size(600,400);
}

void draw()
{
  background(50);
  textSize(32);
  text("Mouse: " + mouseX + " " + mouseY,20,20);
  fill(200,50,200);
  ellipse(mouseX,mouseY,80,80);
}
```

Displaying Text

- The text can be set to different colors or shades of gray using the **`fill()`** function.
- To set a shade of gray, use one value in the function with a range of 0 to 255.

```
void setup()
{
  size(600,400);
}

void draw()
{
  background(50);
  textSize(32);
  fill(180);              // set to gray
  text("Mouse: " + mouseX + " " + mouseY,20,20);
  fill(200,50,200);
  ellipse(mouseX,mouseY,80,80);
}
```

Displaying Text

- To set the text to a color, insert three numbers in the **`fill()`** function. The numbers represent the primary colors red, green, blue.
- Values range from 0 to 255.

```
void setup()
{
  size(600,400);
}

void draw()
{
  background(50);
  textSize(32);
  fill(30,100,250);
  text("Mouse: " + mouseX + " " + mouseY,20,20);
  fill(200,50,200);
  ellipse(mouseX,mouseY,80,80);
}
```

Animating

- Next is to make some animation.
- This next example will show a rectangle grow in size.
- It starts on the left and grows to the right across the screen then starts over.
- The variable a sets the length of the rectangle. It is incremented each time the **`draw()`** function repeats.
- Try the program out and then change it to start at the bottom of the screen and grow toward the top. It will require a little math.

```
int a;
void setup()
{
  size(600,400);
  a = 1;
}

void draw()
{
  background(0);
  fill(0,150,200);
  rect(20,200,a,50);
  a = a + 1;
  if(a == 500) a = 1;
  }
}
```

Animating

- Notice that the variable **`a`** is checked with the **`if()`** statement to determine if it reached 500. If it increment to 500, a is set back to 1.

```
int a;
void setup()
{
  size(600,400);
  a = 1;
}

void draw()
{
  background(0);
  fill(0,150,200);
  rect(20,200,a,50);
  a = a + 1;
  if(a == 500) a = 1;
  }
}
```

More Animation

- Now, let's try some movement.
- Set the size of the rectangle to 50,50 pixels. Replace the x coordinate with the variable **a**.
- Run the program. The rectangle should move left to right and then start over on the left side.
- Add to the program to have the rectangle move back to the original position. It's just another for loop with the value going from 500 to 1.

```
int a;
void setup()
{
  size(600,400);
  a = 1;
}

void draw()
{
  background(0);
  fill(0,150,200);
  rect(a,200,50,50);
  a = a + 1;
  if(a == 500) a = 1;
  }
}
```

More Animation

- The color can also be animated.
- Replaec one of the values in fill with the variable.
- The value of a has to be reset after it reaches 255.

```
int a;

void setup()
{
  Size(600,400);
  a = 0;
}

void draw()
{
  background(0);
  fill(a,150,200);
  rect(200,200,50,50);
  a++;
  if(a >255) a = 0;
}
```

End

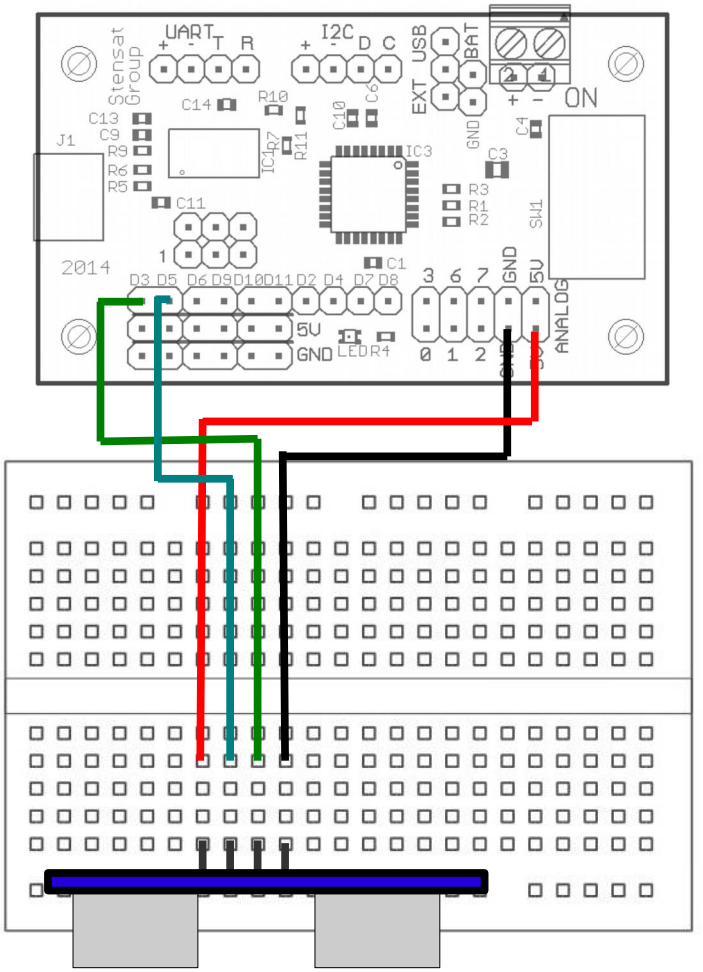
- At this point, you have a basic idea of creating shapes, detecting keys pressed and simple animations. This will be used with the rover for the remote operations.

Animating Sensor Data

- In this next section, sensor data will be used to control the position of a rectangle.
- The first sensor to use will be the ultrasonic range sensor.
- The animation will show the rectangle move away and toward the left edge of the screen based on the data from the sensor.

The Set Up

- Insert the ultrasonic ranger as shown. It should be mounted close to the center of the robot. The pins are inserted at the end of the rows.
- Connect jumpers from the sensor to the processor
  - GND to Analog GND
  - ECHO to pin D3
  - TRIG to pin D5
  - VCC to Analog 5V
- Look on the processor board for the word ANALOG. The power connections are done there to isolate the sensor from the motor power to reduce electrical noise.



The Code

- Next is to reuse the code from the rover lesson that was on page 109.
- If it wasn't saved, enter the code to the right.

```
void setup()
{
    Serial.begin(9600);
    pinMode(3,INPUT);
    pinMode(5,OUTPUT);
}

void loop()
{
    digitalWrite(5,LOW);
    delayMicroseconds(2);
    digitalWrite(5,HIGH);
    delayMicroseconds(10);
    digitalWrite(5,LOW);
    long distance = pulseIn(3,HIGH);
    distance = distance/58;
    Serial.println(distance);
    delay(500);
}
```

Animating

- Create a new program in Processing.
- Import the library **`Serial`**. The text to the right will appear. This will allow processing to talk to the rover processor board.

```
import processing.serial.*;
```

Animating

- Next is to create an instance of the object **Serial**. It will be called **port**.
- This is part of object oriented programming. The object **Serial** has several functions. Creating an instance of the **Serial** object lets you have multiple unique copies. This would be useful if you needed to communicate through more than one serial interface.

```
import processing.serial.*;

Serial port;
```

Animating

- With the Serial object port created, it is time to configure it.
- port is assigned a Serial interface. Change **x** to the **COM** port number used in the Arduino software. This is similar to **Serial.setup().**

```
import processing.serial.*;

Serial port;

void setup() {
  port = new Serial(this,"COMx",9600);
```

Animating

- We want to create a window for our animation and to have it take up the whole screen.
- Processing has two variables automatically set to the size of the computer screen. They are called **displayWidth** and **displayHeight**.
- These variables allow you to know the display size on the computer the program is running. This is useful to know since different computers have different display sizes.

```
import processing.serial.*;

Serial port;

void setup() {
  port = new Serial(this,"COMx",9600);
  size(displayWidth,displayHeight);
}
```

Animating

- Now that things are initialized, next is to create the animation.
- In the draw() function, the port is checked to see if any data has been received from the rover processor board. The function **port.available()** checks and returns the number of bytes received.

```
import processing.serial.*;

Serial port;

void setup() {
  port = new Serial(this,"COMx",9600);
  size(displayWidth,displayHeight);
}

void draw() {
  background(50);
  if(port.available() > 0) {
```

Animating

- If data is available, the **if()** statement is true.
- The next statement reads the data into a **String** variable.
- A **String** is a variable for containing text.

```
import processing.serial.*;

Serial port;

void setup() {
  port = new Serial(this,"COMx",9600);
  size(displayWidth,displayHeight);
}

void draw() {
  background(50);
  if(port.available() > 0) {
    String a = port.readString();
```

Animating

- The next thing that is done is the String variable a is checked to make sure there is something in the variable.
- If there is something in the variable, then the results are displayed in the window.

```
import processing.serial.*;

Serial port;

void setup() {
  port = new Serial(this,"COMx",9600);
  size(displayWidth,displayHeight);
}

void draw() {
  background(50);
  if(port.available() > 0) {
    String a = port.readString();
    if(a != null) {
      text(a,50,50);
    }
  }
}
```

Animating

- Enter the program and run it. Turn on the rover processor board.
- Text should appear with the distance measurement from the ultrasonic range sensor.

```
import processing.serial.*;

Serial port;

void setup() {
  port = new Serial(this,"COMx",9600);
  size(displayWidth,displayHeight);
}

void draw() {
  background(50);
  if(port.available() > 0) {
    String a = port.readString();
    if(a != null) {
      text(a,50,50);
    }
  }
}
```

Animating

- The next line is used to remove the carriage return and line feed characters that the rover processor includes when it sends the numbers over the USB port.
- This needs to be done so that the next function can operate properly.

```
import processing.serial.*;

Serial port;

void setup() {
  port = new Serial(this,"COMx",9600);
  size(displayWidth,displayHeight);
}

void draw() {
  background(50);
  if(port.available() > 0) {
    String a = port.readString();
    if(a != null) {
      text(a,50,50);
      a = a.replaceAll("(\\r|\\n)","");
    }
  }
}
```

Animating

- Next, the String variable is converted to an integer value so that the number can be used.

```
import processing.serial.*;

Serial port;

void setup() {
  port = new Serial(this,"COMx",9600);
  size(displayWidth,displayHeight);
}

void draw() {
  background(50);
  if(port.available() > 0) {
    String a = port.readString();
    if(a != null) {
      text(a,50,50);
      a = a.replaceAll("(\\r|\\n)","");
      int b = Integer.parseInt(a);
    }
  }
}
```

Animating

- Now it is time to change the code to animate the distance using a rectangle.
- First, the string variable needs to be converted to an integer.
- Then the rectangle can be drawn with the **rect()** function.
- Try running this program and see the results.

```
import processing.serial.*;

Serial port;

void setup() {
  port = new Serial(this,"COMx",9600);
  size(displayWidth,displayHeight);
}

void draw() {
  background(50);
  if(port.available() > 0) {
    String a = port.readString();
    if(a != null) {
      text(a,50,50);
      a = a.replaceAll("(\\r|\\n)","");
      b = Integer.parseInt(a);
      rect(b,100,10,300);
    }
  }
}
```

Animating

- If you want to make the rectange move further to the right, this would require scaling.
- Go to the rect() function and double b as shown.
- Rerun the program.
- Change the 2 to any other number and see what happens.

```
import processing.serial.*;

Serial port;

void setup() {
  port = new Serial(this,"COMx",9600);
  size(displayWidth,displayHeight);
}

void draw() {
  background(50);
  if(port.available() > 0) {
    String a = port.readString();
    if(a != null) {
      text(a,50,50);
      a = a.replaceAll("(\\r|\\n)","");
      b = Integer.parseInt(a);
      rect(b*2,100,10,300);
    }
  }
}
```

Modifications

- Instead of moving the rectange, modify the program to change the size of the rectangle based on the sensor data.
- Use the fill() function to change the color based on the sensor data. You may need to scale it so none of the colors exceeds 255.