Processing

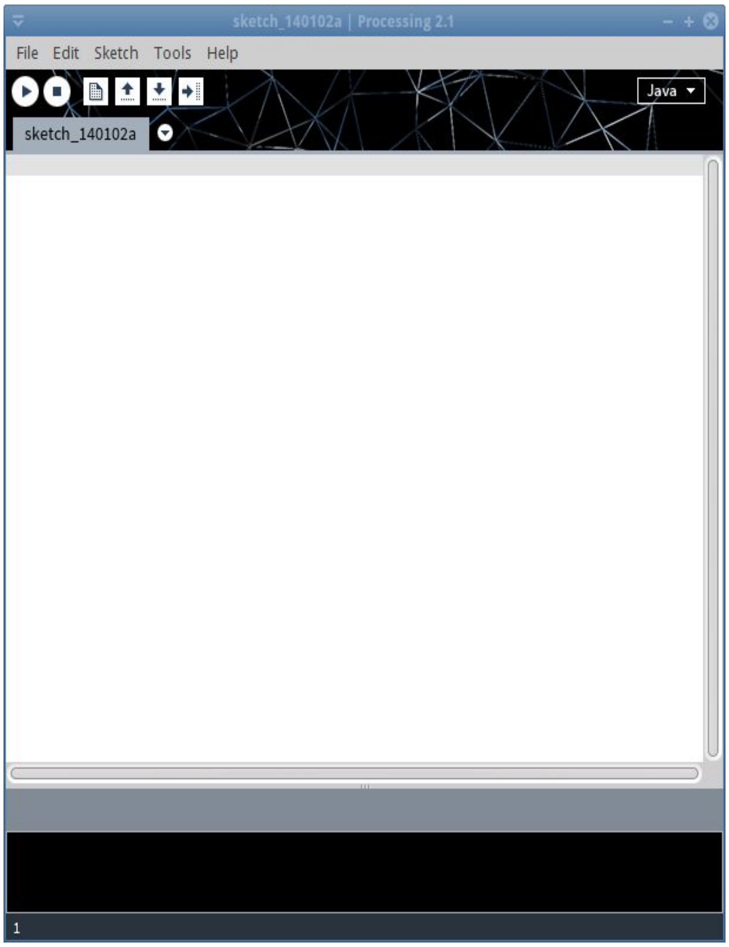
Introduction to Processing

Processing is a programming environment that makes writing programs easier.

It contains libraries and functions that make interacting with the program simple.

Processing IDE

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.

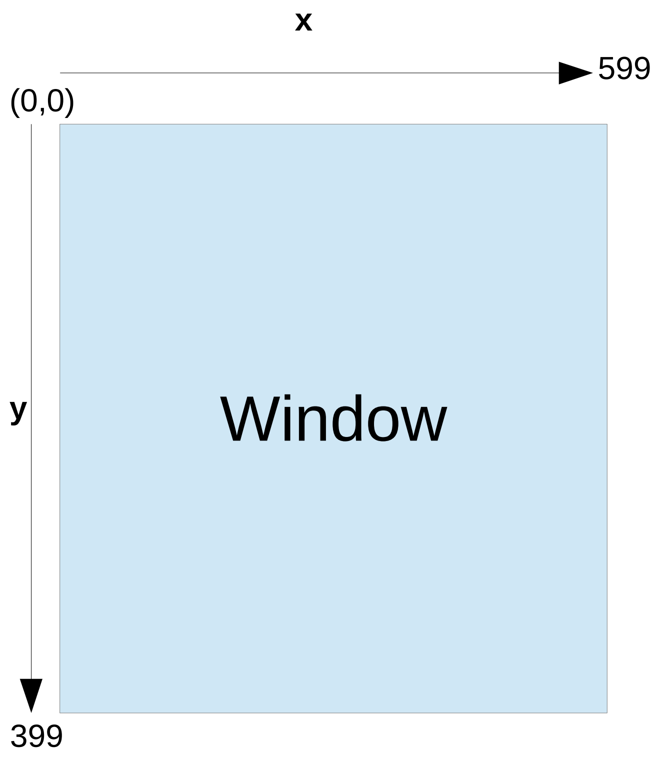


First Program

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. When the program starts, it first executes the **setup()** function once and then repeatedly executes the **draw()** function.

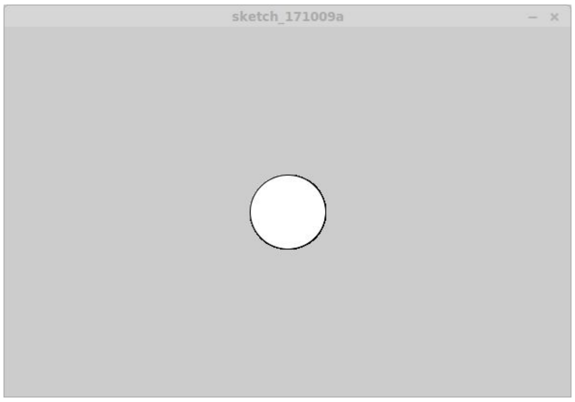
```
void setup()
{
  size(600,400);
}

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

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.

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. If the third and fourth numbers are the same, the ellipse is a circle.



Result of Program

Adding to the First Program

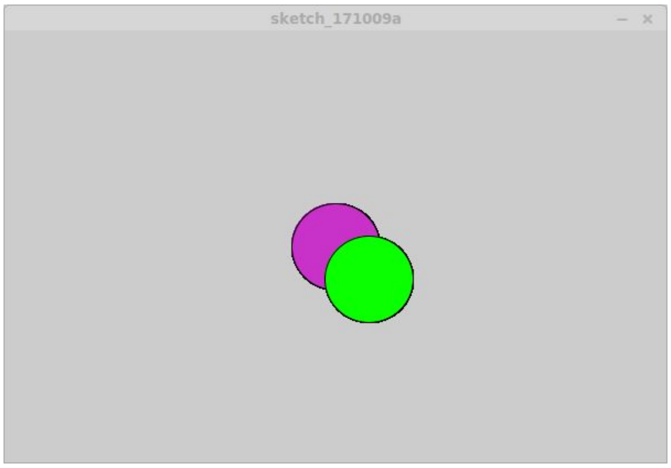
The color of the ellipse can be changed with the function **`fill()`**. Add the function **`fill()`** before the **`ellipse()`** function. The ellipse should now be nearly black. Change the number to any number in the range of 0 to 255 and run again. The number sets the gray intensity level of the ellipse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

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

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

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. The intensity ranges from 0 which is black and 255 which is the brightest.

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

void draw()
{
  fill(200,50,200);
  ellipse(300,200,80,80);
  fill(10,255,0);
  ellipse(330,230,80,80);
}
```

Notice the second ellipse added to the program. It is drawn on top of the first one. Objects are always drawn on top of previous objects. Add two more ellipses of different sizes and colors and see how they are drawn on top of each other.



Ellipse drawn over ellipse

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**. Make sure the X and Y are capital letters. Run the program and see what happens. As ellipses are generated, they are drawn on top of previous ellipses. Remember that newer objects are drawn on top of older objects.

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

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

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.

By default, the draw() function is executed 60 time a second which is the display refresh rate. This is useful for making animations and interactive graphics like this program does.

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

void draw()
{
  background(0);
  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);
}
```

The **draw()** function includes some conditional programming. They are the **if()** statements. The **if()** statement is used to make comparisons. This one is comparing the variable key to different letters. Notice that two equal symbols are used. This is because a single equal symbol indicates an assignment. If you use a single equal in the **if()** statement, then the result is always true.

If the comparison is true, then the code in the braces after the **if()** statement will be executed. If the result is not true, then the program will skip the code between the braces.

Multiple comparisons can be made. The **else** statement lets another **if()** statement to be executed if the previous one resulted in not being true. Also notice there are no curly braces after the second **if()** statement. This can be done if only a single instruction is needed for the true condition of the **if()** statement.

Types of comparisons:

== equal
> greater than
< less than
>= equal or greater than
<= equal or less than
!= not equal

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**. Try this program.

In the **keyPressed()** function, the **switch()** statement is used. This is used in place of the **if()** else statements. It operates similarly except the comparisons are always equal.

The variable to be compared in specified by the **switch()** statement. Then each line has a **case** statement with a value. This value is compared to the specified variable. If the comparison is true, all the code after the colon is executed. More than one line of code can exist after the colon.

At the end of the code for each **case** statement is a **break** statement. The **break** statement forces the program to exit the **switch()** statement. If the **break** was not included, the program would keep executing code for all the other **case** statements below.

```
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

The rectangle primitive has four parameters. The first two are for the position of the rectangle. The coordinates given are the top left corner of the rectangle. The Third parameter is the width and the fourth is the height.

rect(x,y,width,height);

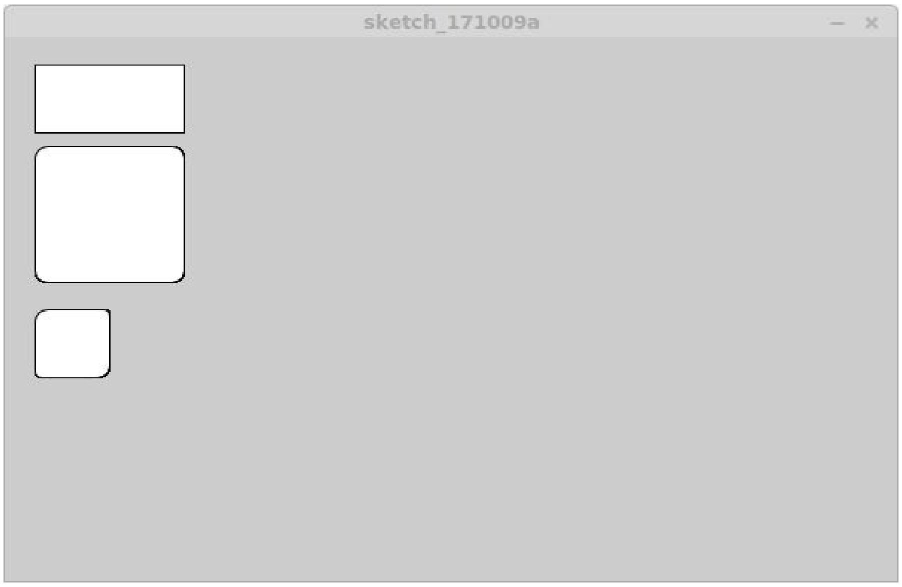
The rectangle primitive can have more parameters. Add a fifth one and it controls how the corners of the rectangle can be curved.

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

Replacing the curve parameter with four numbers allows each corner of the rectangle to have different size curves.

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

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

void draw() {
  rect(20,20,100,50);
  rect(20,80,100,100,10);
  rect(20,200,50,50,10,5,10,5);
}
```



The three rectangles

Other 2D Primitives

The triangle primitive has three coordinates for each corner.

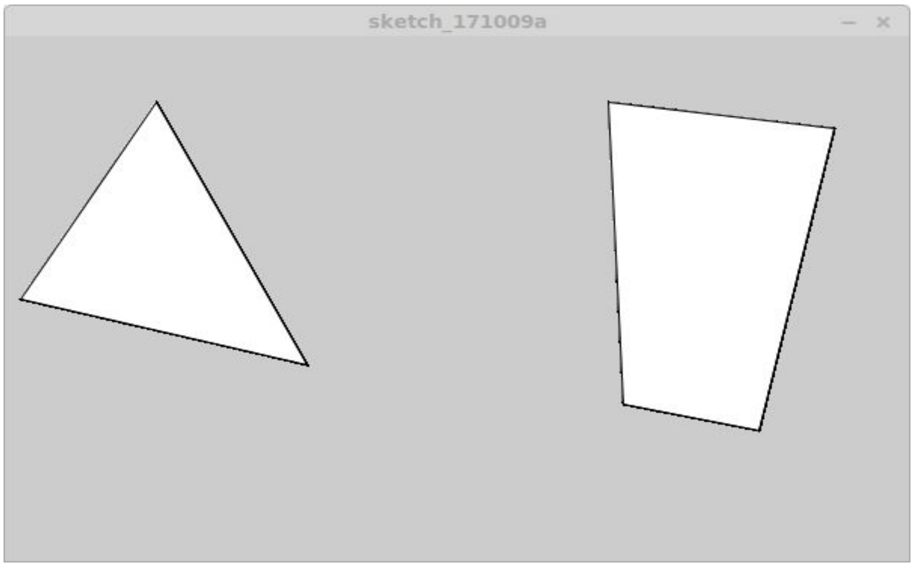
triangle(x1,y1,x2,y2,x3,y3);

The quadrilateral has four coordinates for each of its corners.

quad(x1,y2,x2,y2,x3,y3,x4,y4);

Try each of the primitives out in a program.

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

void draw() {
  triangle(10,200,100,50,200,250);
  quad(400,50,550,70,500,300,410,280);
}
```



Triangle and quad

Displaying Text

The function **text()** is used to display text in the window. The program below 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. 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);
}
```

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);
}
```

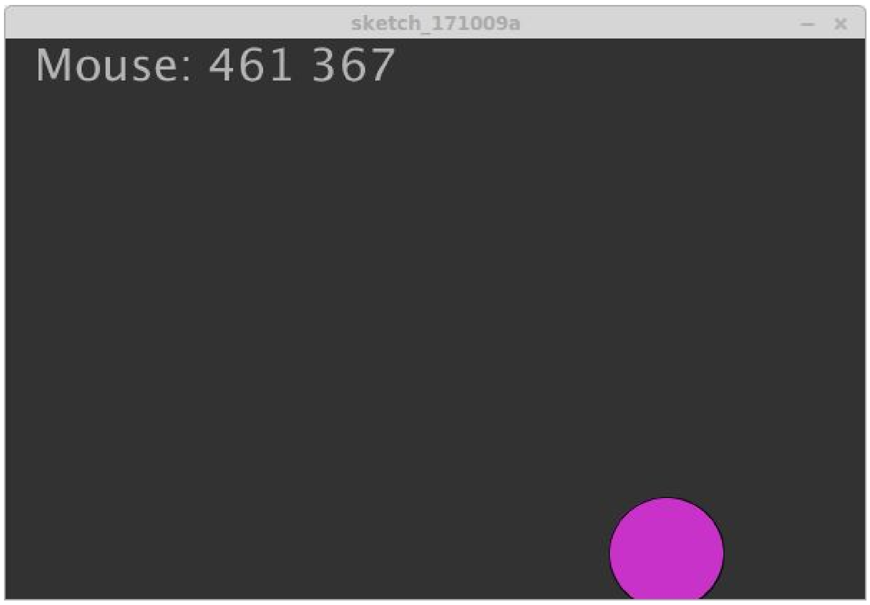
Displaying Text

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. After each **fill()**, all things drawn on the screen will be that color. You need to add the **fill()** to change colors. As seen in the program below, the text is set to gray and the ellipse is set to purple.

Change the text to a color instead of gray and try it out.

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

void draw()
{
  background(50);
  textSize(32);
  fill(180);               // set to gray
  text(“Mouse: “ + mouseX + “ “ + mouseY,20,30);
  fill(200,50,200);
  ellipse(mouseX,mouseY,80,80);
}
```

Notice the // in the program. This is a comment indicator. Anything to the right of // is ignored by the computer. This is useful for putting notes in the program to provide a description of what is happening.



```
Mouse: 461 367
```

Animating

Next is to make an 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.

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.

```
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;
}
```

Add to the program to have the rectangle move back to the original position. To do this, a second variable needs to be added to indicate the direction. Variable **dir** is set to 1 to indicate left to right movement. In the **draw()** function, **dir** is checked for the direction. If 1, the position is incremented. The check is also included. If **dir** is 0, then the rectangle is moving right to left. The position is decremented. The position is also checked to make sure it does not go left out of the window.

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

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

More Animation

The color can also be animated. Replace 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;
}
```

Interacting With the Robotic Arm

In this section, you will learn how to get the robotic arm kit talking with Processing. You will get potentiometer positions from the robotic arm into Processing and make a simple graphical user interface to control the Robotic arm.

The robotic arm needs to be connected to the computer running Processing with the USB cable. All communications will occur over USB. The interface is called a COM port. Processing will use the same COM port number that the Arduino software uses.

You will need to load the Firmata software into the robotic arm. It is included with the Arduino software. Firmata provides a communications interface that allows the user to access all the devices on the robotic arm. Processing has a library available to interact with Arduino based devices using the Firmata software.

Start Processing. Select the **Sketch** menu and select **Import Library** then select **Add Library**. A new window will pop up listing a variety of libraries that can be added. Look for **Arduino (Firmata)** and install it.

Connect the robotic arm to the computer and start the Arduino software. Under the **File** menu, select **Examples**. Under Examples locate **Firmata** and select **StandardFirmata**. A new window will open with the Firmata program. Make sure the robotic arm is on and click on the **Upload Code** button. This will compile and upload the Firmata software. Once complete, you are ready to continue.

Getting Potentiometer Data

In Processing, import the Arduino Firmata Library. Select the **Sketch** menu then select **Import Library**. Locate **Arduino Firmata** and select it. Then go back to the **Import Library** menu and select **Serial**.

Enter the code after the import statements. The first line is declaring a variable of type **Arduino**. In the **setup()** function, the robotic arm is accessed with the **Arduino()** function. In the draw function, the three potentiometers are read using the port.readAnalog() functions. Analog port 0 is the base potentiometer. The arm potentiometer is the port 1 and port 2 is the elbow potentiometer.

The data is displayed with the **println()** function. Notice how the data is displayed. To combine different information to be displayed on the console, the **+** is used connect them together. The **" "** inserts a space between the numbers.

```
import cc.arduino.*;
import org.firmata.*;
import processing.serial.*;

Arduino port;

void setup()
{
  port = new Arduino(this,”COM3”,57600);
}

void draw() {
  int b = port.readAnalog(0);
  int a = port.readAnalog(1);
  int e = port.readAnalog(2);
  println(b + “ “ + a + “ “ + e);
}
```

Changing Colors

Let's do something with the potentiometer values. We'll use them to change the color of the program window. Using the same code as before, instead of printing the data, the data is used to change the color of the window.

After the values of the potentiometers are read, the values need to be scaled to fit the color range values. The potentiometers has a range of 0 to 1023 while each color has a range of 0 to 255. After the scaling, the window color is updated.

Make sure the robotic arm kit is on and the USB cable is plugged into the computer. Start the Processing program and adjust the potentiometers.

Notice the // in various parts of the program. These are called comments. These are notes you can use to remind you what parts of the program do. The computer does not try to execute them. You can use comments anywhere. // indicates the start of the the comment to the right. Anything to the left is part of the program instructions.

```
import cc.arduino.*;
import org.firmata.*;
import processing.serial.*;

Arduino port;       // declare a serial port

void setup()
{
  port = new Arduino(this,”COM3”,57600); // open and configure
  size(1100,600);                        // open a window
}

void draw() {
  float r = port.analogRead(0);    // get potentiometers
  float g = port.analogRead(1);
  float b = port.analogRead(2);
  r = map(r,0,1023,0,255);         // scale to 0-255 range
  g = map(g,0,1023,0,255);
  b = map(b,0,1023,0,255);
  background(r,g,b);               // set window to new color
}
```

Moving an Object

In this new program, we will use the potentiometer to move an object in Processing. We will keep most of the previous program up until the map functions. The background is set to black which clears anything drawn before. Next, the rectangle is drawn with the X position set to the first potentiometer value. With the robotic arm on and the Firmata program still running, start this program and adjust the left potentiometer.

As an exercise, add two more rectangles, each controlled by the other two potentiometers. Make each rectangle a different color.

And to make it more interesting, go back to one rectangle but use the potentiometer to control X and Y. Use the third potentiometer to control the size of the rectangle.

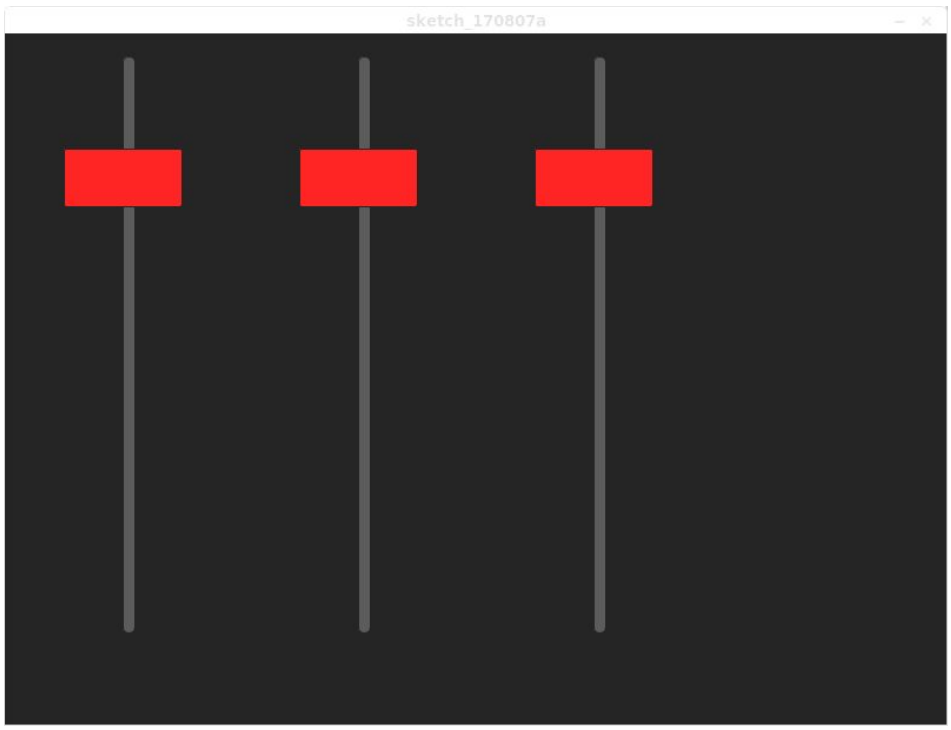
```
import cc.arduino.*;
import org.firmata.*;
import processing.serial.*;

Arduino port;       // declare a serial port

void setup()
{
  port = new Arduino(this,”COM3”,57600);  // open and configure
  size(1100,600);                        // open a window
}

void draw() {
  float r = port.analogRead(0);    // get potentiometers
  float g = port.analogRead(1);
  float b = port.analogRead(2);
  background(0);              // set window to black
  rect(r,550,100,20);
}
```

Building a GUI

GUI stands for Graphical User Interface. We will make one to control the robotic arm. It will consist of three sliders to control the three servos. Two programs need to be written, one in Processing to provide the GUI and one in the Arduino software to get the robotic arm to receive commands and move the robotic arm.

For Processing, the flow of the program is get inputs, update the graphics and send command to the robotic arm. For the robotic arm, it is listen for data input, get values, update servos.

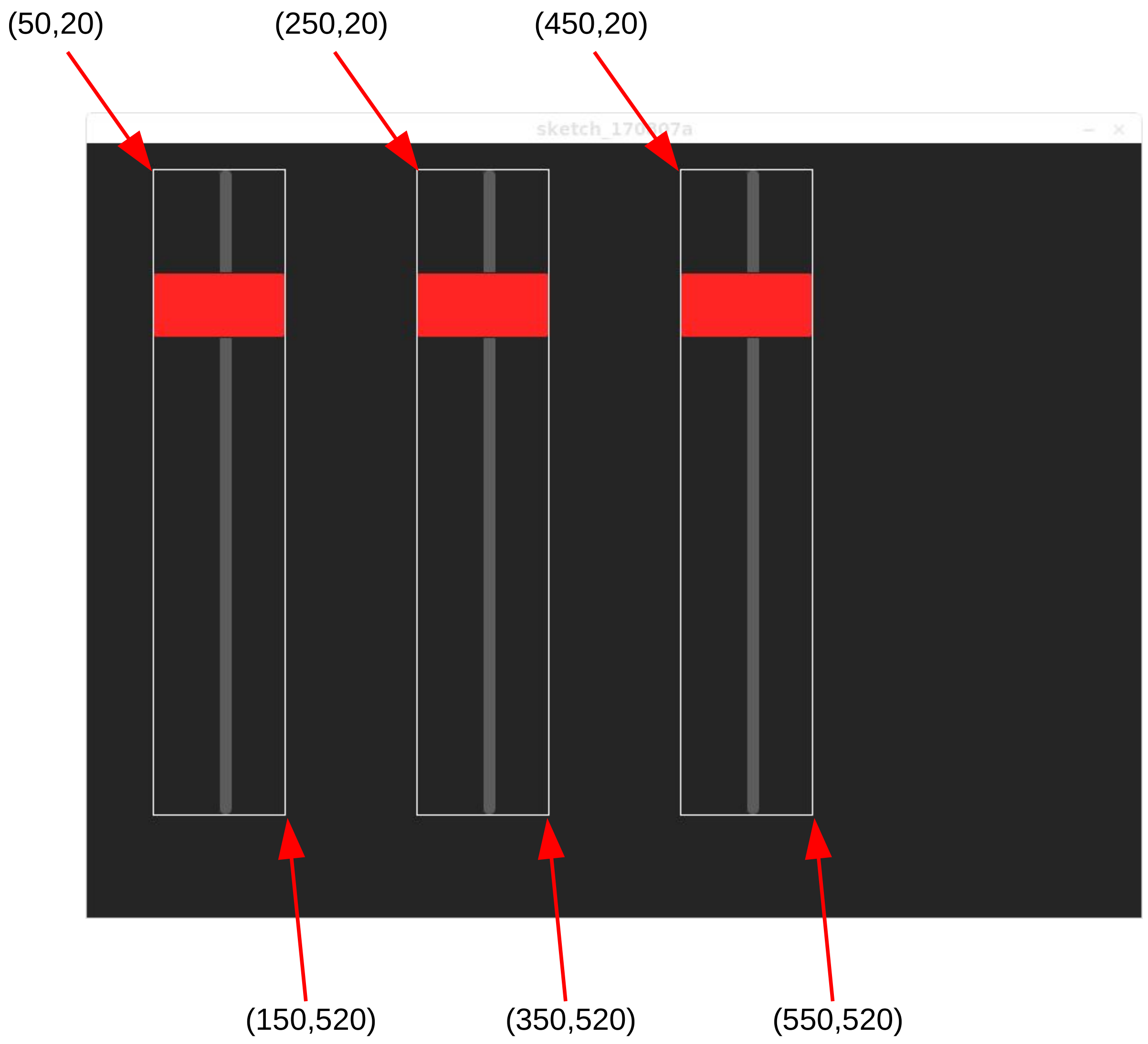
The GUI is shown below. Thee are three sliders. They are made with two rectangles.



GUI Construction

Coordinates of the sliders are important. The image below shows the outline of the sliders. The sliders are 100 pixels wide and 500 pixels tall. The top left coordinate for the slider on the left is (50,20). The bottom right corner is (50+100,20+500) or (150,520). The second slider top left corner is (250,20). The third slider top left corner is (450,20). Calculate the bottom right corner for the second and third slider.

To know when a slider is being controlled, the program has to determine if the mouse is inside the slider as shown by the rectangles. Two comparisons need to be made, one in the x direction and the other in the y direction. The mouse position needs to be between the X and Y ranges. Outside the ranges, no slider should be controlled.



Making a Slider

We now move on to build up a program to control the robotic arm using a graphical user interface. The interface will work with a mouse or touch screen. To keep it simple, we will use two rectangles. One rectangle will be long and skinny and show the length of the slider. The other rectangle will be the knob of the slider.

A variable **p1** is declared first. This makes is a global variable so that all functions can use it. In **setup()**, the window is sized and **p1** is initialized to a start value.

In the draw() function, the first check is to see if the mouse button is pressed. If so, the mouse location is checked. The top left corner of the knob has an X value of 50 and is 100 pixels long so the the mouseX is checked if it is within 50 and 150. If so, the Y value is checked to see if it is between 20 and 520 pixel. The length of the slider is 500 pixels starting at pixel Y value of 20. If the mouse is within the X and Y ranges of the slider, the slider knob is updated to the Y position of the mouse. Notice when you run the program that if you click the mouse anywhere within the ranges, the knob will be positioned there. You don't have to click on the knob to move it. If you hold the mouse button down and move the mouse, the slider knob will move with the mouse as long as the mouse is within the ranges.

In the if() statement, notice the **&&** symbols. This stands for AND. This allows you to compare more than one thing and both comparisons must be true for the if() statement result to be true.

```
int p1;        // hold current position of slider knob

void setup() {
  size(800,600);
  p1 = 100;    // set initial position
}

void draw() {
  if(mousePressed) {     // check if in range
    if((mouseX > 50) && (mouseX < 150)) {
      if((mouseY > 20) && (mouseY < 520)) {
        p1 = mouseY;     // update knob position
      }
    }
  }
  background(0);               // clear screen
  fill(80);                    // draw slider
  rect(100,20,10,500,5);
  fill(200,0,0);               // draw knob
  rect(50,p1,100,50,2);
}
```

Making Three Sliders

The robotic arm has three servos to control. We need to add two more sliders.

First, create two more variables. Then in the draw() function, we need to check the mouse position in three areas to determine which of the three sliders to modify.

Notice the code is the duplicated for each slider. The mouseX position is compared to three different ranges of numbers. The first slider is positioned at 100, the second at 300 and the third at 500.

At the bottom of the code, the rectangles are drawn. First, all three slider lengths are drawn then the knobs are drawn.

Each knob has its own variable.

Try the program out and see how the sliders behave.

```
int p1,p2,p3;

void setup() {
  size(800,600);
  p1 = p2 = p3 = 100;
}

void draw() {
  if(mousePressed) {
    if((mouseX > 50) && (mouseX < 150)) {
      if((mouseY > 20) && (mouseY < 520)) {
        p1 = mouseY;
      }
    }
    if((mouseX > 250) && (mouseX < 350)) {
      if((mouseY > 20) && (mouseY < 520)) {
        p2 = mouseY;
      }
    }
    if((mouseX > 450) && (mouseX < 550)) {
      if((mouseY > 20) && (mouseY < 520)) {
        p3 = mouseY;
      }
    }
  }
  background(0);
  fill(80);
  rect(100,20,10,500,5);
  rect(300,20,10,500,5);
  rect(400,20,10,500,5);
  fill(200,0,0);
  rect(50,p1,100,50,2);
  rect(250,p2,100,50,2);
  rect(450,p3,100,50,2);
}
```

Making Three Sliders

Let's add the Firmata library. In the **setup()** function, notice three lines after configuring the port. The three functions configure the digital pins 3, 5, 6 to generate servo signals. All port configurations are performed in the **setup()** function.

Also, the servos are positioned to start starting point.

This only covers the **setup()** function. The **draw()** function is needed and is on the next page. It must be added to the program.

```
import cc.arduino.*;
import org.firmata.*;
import processing.serial.*;

Arduino port;
int p1,p2,p3;

void setup() {
  size(800,600);
  p1 = p2 = p3 = 100;
  port = new Arduino(this,”COM3”,57600);
  port.pinMode(3,Arduino.SERVO);
  port.pinMode(5,Arduino.SERVO);
  port.pinMode(6,Arduino.SERVO);
  port.servoWrite(3,90);
  port.servoWrite(5,80);
  port.servoWrite(6,90);
}
```

Making Three Sliders

In the draw function, everything stays the same. At the bottom are three **map()** functions that convert the slider position to the range of the servos. The base servo can move from 1 to 179. Notice that it is entered as 179 to 1. This is to make the robotic arm move left to right as the slider is moved from top to bottom.

The arm servo has a shorter range of 1 to 80. The elbow has a range of 1 to 179. Remember, the sliders have a range of 20 to 520. That range is larger than the servos can handle so the **map()** function is used to scale to the proper range.

The **(int)** in front of the **map()** function is to convert the results of the **map()** function from floating point to integer. The Arduino **map()** function works with integers while Processing is only floating point so the number has to be converted to the proper type. The **servoWrite()** function uses integers only.

```
void draw() {
  if(mousePressed) {
    if((mouseX > 50) && (mouseX < 150)) {
      if((mouseY > 20) && (mouseY < 520)) {
        p1 = mouseY;
      }
    }
    if((mouseX > 250) && (mouseX < 350)) {
      if((mouseY > 20) && (mouseY < 520)) {
        p2 = mouseY;
      }
    }
    if((mouseX > 450) && (mouseX < 550)) {
      if((mouseY > 20) && (mouseY < 520)) {
        p3 = mouseY;
      }
    }
  }
  background(0);
  fill(80);
  rect(100,20,10,500,5);
  rect(300,20,10,500,5);
  rect(400,20,10,500,5);
  fill(200,0,0);
  rect(50,p1,100,50,2);
  rect(250,p2,100,50,2);
  rect(450,p3,100,50,2);
  int base = (int)map(p1,20,520,179,1);
  int arm = (int)map(p2,20,520,1,80);
  int elb = (int)map(p3,20,520,1,179);
  port.servoWrite(3,base);
  port.servoWrite(5,arm);
  port.servoWrite(6,elb);
}
```

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.