Arduino Etch A Sketch Workshop

What is an Etch A Sketch?

An Etch A Sketch is a drawing toy that was sold by the Ohio Art Company. It was first sold in the 1960s. The toy draws a line using two controllers: a knob that changes the horizontal position and a second knob for the vertical position. To erase what has been drawn, the toy is flipped upside down and is shaken.

We are going to mimic this interaction using Arduino and Processing. We will use Processing to draw on a computer, and we will control that drawing using two potentiomenters and a tactile button.

1. Processing Sketch

Setting Up the Sketch

First we set up the sketch as we would any Processing sketch with `setup()` and `draw()`

functions.

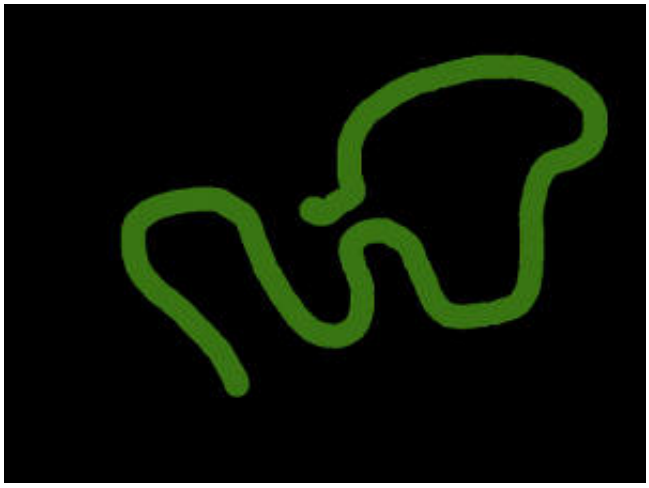
```
void setup() {
    size( 800, 600 );
}

void draw() {

}
```

Drawing with the Mouse

Now we are ready to add some interaction.



Exercise 1

Create a new Processing sketch. Use the `ellipse()` shape to draw a trail of circles wherever the mouse goes.

Exercise 2

If you haven't already, use variables to control the colour and diameter of the circles.

Clearing with a Key Press

Now that we have a trail of circles wherever the mouse moved, we need to add functionality to clear the screen. We will use the `keyPressed()` function.

```
void keyPressed() {

}
```

The `keyPressed()` function is called whenever a key on the keyboard is pressed.

Exercise 3

Using the `background()` function, clear the screen every time a key is pressed.

Exercise 4

If you haven't done so already, use a variable or multiple variables to set the background color.

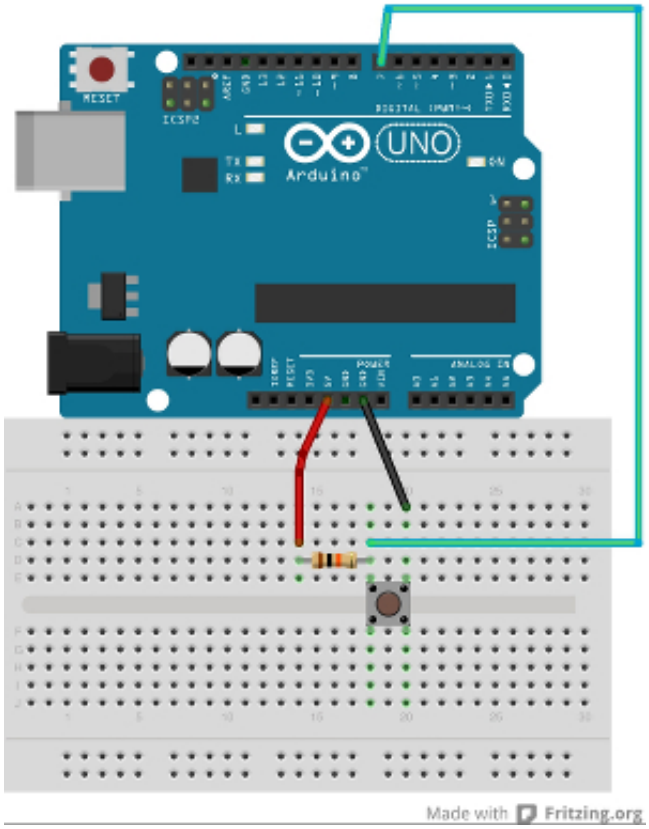
2. Adding a Button

We will introduce control from our Arduino by adding a tactile button to clear the screen.

Circuit

Exercise 1

Using a breadboard, Arduino, tactile button and 10k resistor, set up a circuit to read in the state of the button on Digital Pin 7.



Arduino Code

Exercise 2

Write and upload an Arduino sketch that reads in the state of the button on Digital Pin 7 and prints it on the Serial Monitor. Check that a 1 is printed when the button is not pressed, and a 0 is shown when it is pressed.

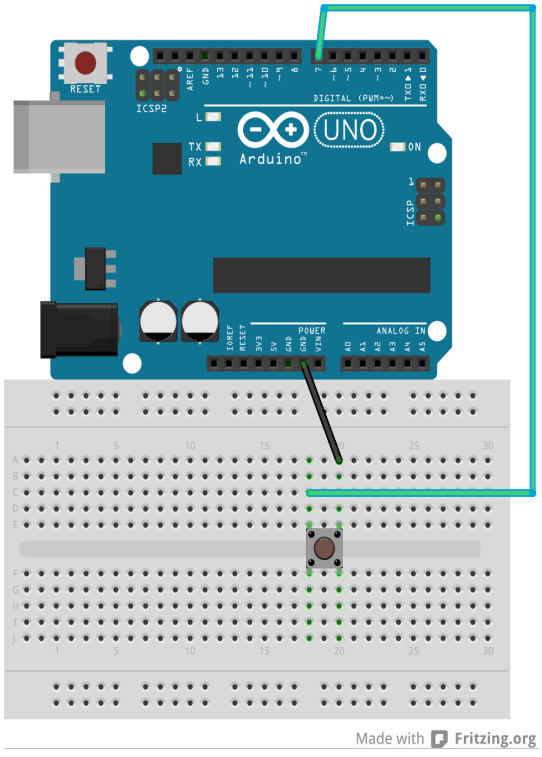
Using Internal Pull Up Resistors

The resistor in the button circuit is a called a pull up resistor. This is because if the switch is open, the signal on the pin on the Arduino reading in the state of the switch is pulled up to 5V. If this pull up resistor hooked up to 5V wasn't there and the switch was open, the pin would be reading in random values. We wouldn't know if the pin is reading in 0 because it is actually connected to ground or if it's just a random number.

The Arduino has resistors in the microcontoller that can be used instead of external resistors. We activate these resistors in code and then have a simpler circuit with fewer components.

Exercise 3

Build the below circuit, removing the external pull up resistor from your button circuit.



Exercise 4

To activate the internal pull up resistor, we need to add a line to our `setup()` function in our Arduino code. pinMode(7, INPUT); // this sets up our pin to be an input digitalWrite(7, HIGH); //this turns on our internal pull up on 7

Confirm that your button still works as before.

Processing Code

When we use `Serial.print()` in our Arduino code, we are sending information over the Arduino's serial port. When we open the Serial Monitor in the Arduino IDE, we are reading that data from the Arduino through the computer's serial port. Processing can access that same serial port and read that data into Processing.

It is important to note that only one device or application can read from a serial port at a time. This means if Serial Monitor is open and reading in data, then a Processing sketch also reading in that same data can't be running. Furthermore, new sketches are loaded onto the Arduino using the serial port, so if you have a Processing sketch running that is accessing the serial port, then you can't upload a new Arduino sketch.

Exercise 5

Processing has a built-in library to handle reading from and writing to a serial port. Import it by adding

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

to the top of your Processing sketch.

Also declare a new `Serial` object as a global variable.

```
Serial myPort; // our serial port
```

Inside of `setup()` we instantiate our object. First we print a list of all serial ports available.

```
println(Serial.list()); // print list of all ports
```

Run the sketch and find the number in the list that corresponds to the serial port that your Arduino is plugged into. It is most likely 0, but it may be another number.

Once you know the number, add the code below to your `setup()` and change the number in the [ ].

```
// you may need to change the number in [ ] to match
// the correct port for your computer
myPort = new Serial(this, Serial.list()[0], 9600);
```

We will now use a new function that is called every time there is new data waiting to be read in from the serial port. This function is defined outside of any other function (it doesn't go inside `setup()` or `loop()`).

```
/* serialEvent
this function is called whenever there is data waiting
on the serial port
*/
void serialEvent(Serial p) {

}
```

Inside of `serialEvent()` we read in the data and save it in a `String`.

```
String inString = p.readString(); // read in the string
```

Now print out the data that is read in.

Exercise 6

We can remove any extra whitespace (that's things like new lines, tabs and spaces) using a built-in function, `trim`.

```
inString = trim( inString );  // remove any whitespace
```

Add this line to `serialEvent` and see if it changes what is printed. Try adding some whitespace to what Arduino prints to the serial port and see if it has any effect.

Exercise 7

We no longer want a key press to clear the screen, so let's change the `keyPressed()` function.

```
/* clearDrawing
 this function is called when button on Arduino is pressed
 */
void clearDrawing() {
    background( bgColor );
}
```

Now change the `serialEvent()` function so that the screen is cleared when the data is "0".

Note that since the data is a `String` you will need to do the following to test what the data is:

```
if ( inString.equals("0") ) {

}
```

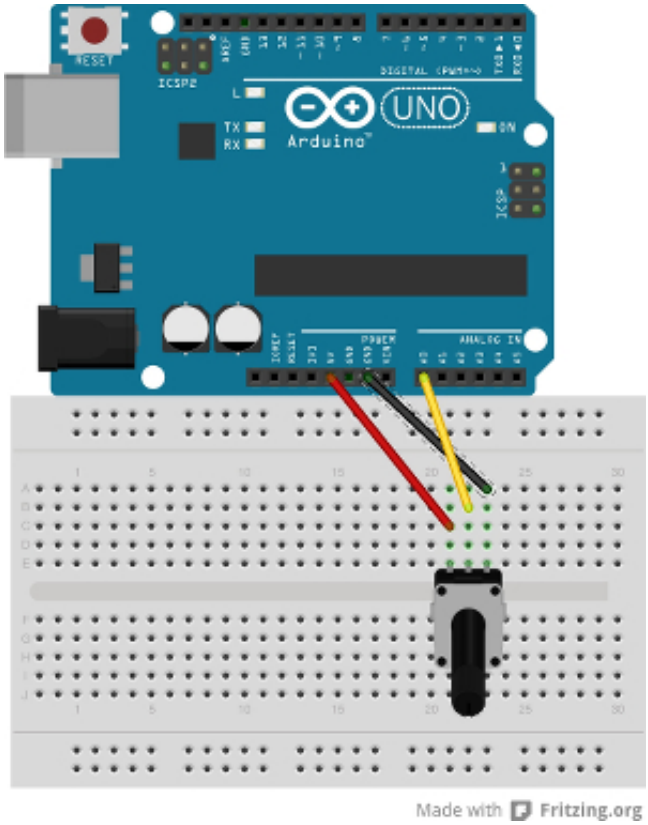
3. Adding the First Knob

We will now add controling the pen from a potentiometer. For now our button won't work with Processing, but will fix that at the end.

Circuit

Exercise 1

Build a circuit with the Arduino reading in the value from a potentiometer into Analogue Pin 0.



In order to have both the button and potentiometer on the same breadboard you will need to alter the layout in order for both components to access 5V.

Arduino Code

With the button we were only sending one character at a time - either a 0 or 1. When we read in an analogue value, we can have up to 4 characters at a time (since the analogue inputs can be up to 1023). We need to know how many digits are in number, so we add a

special character to separate consecutive values. We could use anything, but it's easy to use the newline character `'/n'`.

Exercise 2

Write a Arduino sketch that reads the potentiometer value and prints to the serial port with a newline after each value. Verify it's working by looking at the data stream in the Serial Monitor.

Processing Code

We have already set up our Processing sketch to read in data from the serial port, but we can have pay special attention to certain characters. We can tell it to only call the `serialEvent()` function when a certain character is received. This lets the data sit and wait for all the characters to arrive before reading in the `String` of our full value.

Exercise 3

In your `setup()` function, add the following line after you instantiate the `Serial` object:

```
myPort = new Serial(this, Serial.list()[10], 9600);
myPort.bufferUntil('\n');
```

Print out the value of `inString` in your `serialEvent()` function and check that the full number is being read in, not only single digits at a time.

Exercise 4

In order to use the value being read in as a `int` variable, we need to convert or cast the `String` into an `int`. The following line does that.

```
int v = int(inString); // convert from a string to int
```

Now that we have an integer that is the value from our potentiometer, change your Processing code so that the potentiometer moves the pen of your drawing left and right (the up and down is still controlled by the mouse).

4. Adding the Second Knob

To add a second potentiometer isn't much more work than adding the first one. We will use a second special character to separate the first potentiometer value from the second potentiometer value.

Circuit

Exercise 1

Add the second potentiometer to the breadboard. Connect it to Analogue Pin 1.

Arduino Code

Exercise 2

Change your Arduino code so that the data sent to the serial port looks like:

```
231; 0
230; 1
229; 1
```

Processing Code

We now have 2 values included each time we read in a `String` from the serial port. We know that they are separated by a `;`, so we can use that to split up the `String`.

Excercise 3

The following line of code takes a `String` and looks for the character `;`. It then returns an array of `String` types.

```
String data[] = split( inString, ';' );
```

We now can test and make sure there are two items in the array as we would expect. If there are, then we can clean up any whitespace from the `String`.

Add the below code to `serialEvent()` and then add in the missing lines of code so that the data from the potentiometers control both the x and y movement of the pen.

```
if ( data.length == 2 ) { // continue only if there are 2 things
    String xValue = trim( data[0] ); // remove extra whitespace
    String yValue = trim( data[1] ); // remove extra whitespace

// convert from string to int and then resize range to window
}
```

5. Putting It All Together

We now have all the elements in place and just have a few last steps to complete.

Circuit

Exercise 1

If you haven't done so already, make sure you have

- a button being read into Digital Pin 7 (should read 1 when not pressed, 0 when pressed)
- a potentiometer connected to A0
- a potentiometer connected to A1

Arduino Code

Exercise 2

We are now going to print the values from the button and both potentiometers. We will add labels so we know where each piece of data came from. Alter your Arduino code so that it prints the following:

```
x: 10; y: 87; button: 1
x: 11; y: 87; button: 1
x: 12; y: 86; button: 1
```

Processing Code

Exercise 2

We already know how to read in a `String` and look for a particular character, then split it up into multiple `String`s.

The following code will split up a `String` wherever there is a `;` and then each shorter `String` is processed one by one.

```
String pairs[] = split( inString, ';' ); // split up string into pairs

// go through each pair of label and value
// and assign it to its variable
for ( int i=0; i<pairs.length; i++) {
    // add code here to split Strings again
}
```

Add code in the `for` loop to further split the `String` where there is a `:`. Then compare the first item in the array from the split to see if the value is from `x`, `y`, or `button`. Remember you use `equals()` to compare `String`s.

You should now have an Etch A Sketch that is controlled by two knobs and a button. Now you can customise the drawing style!