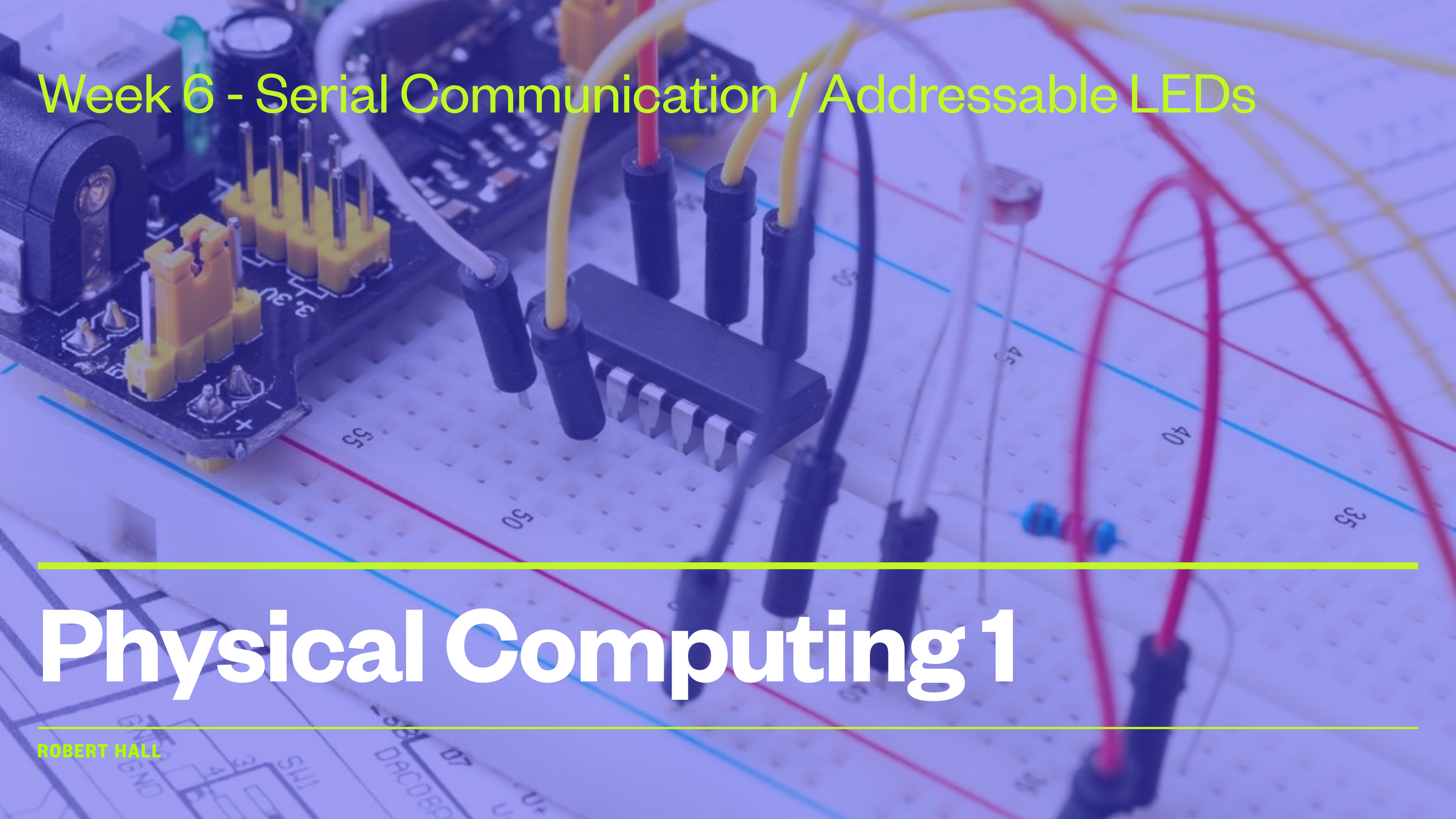


Week 6 - Serial Communication / Addressable LEDs

Physical Computing 1

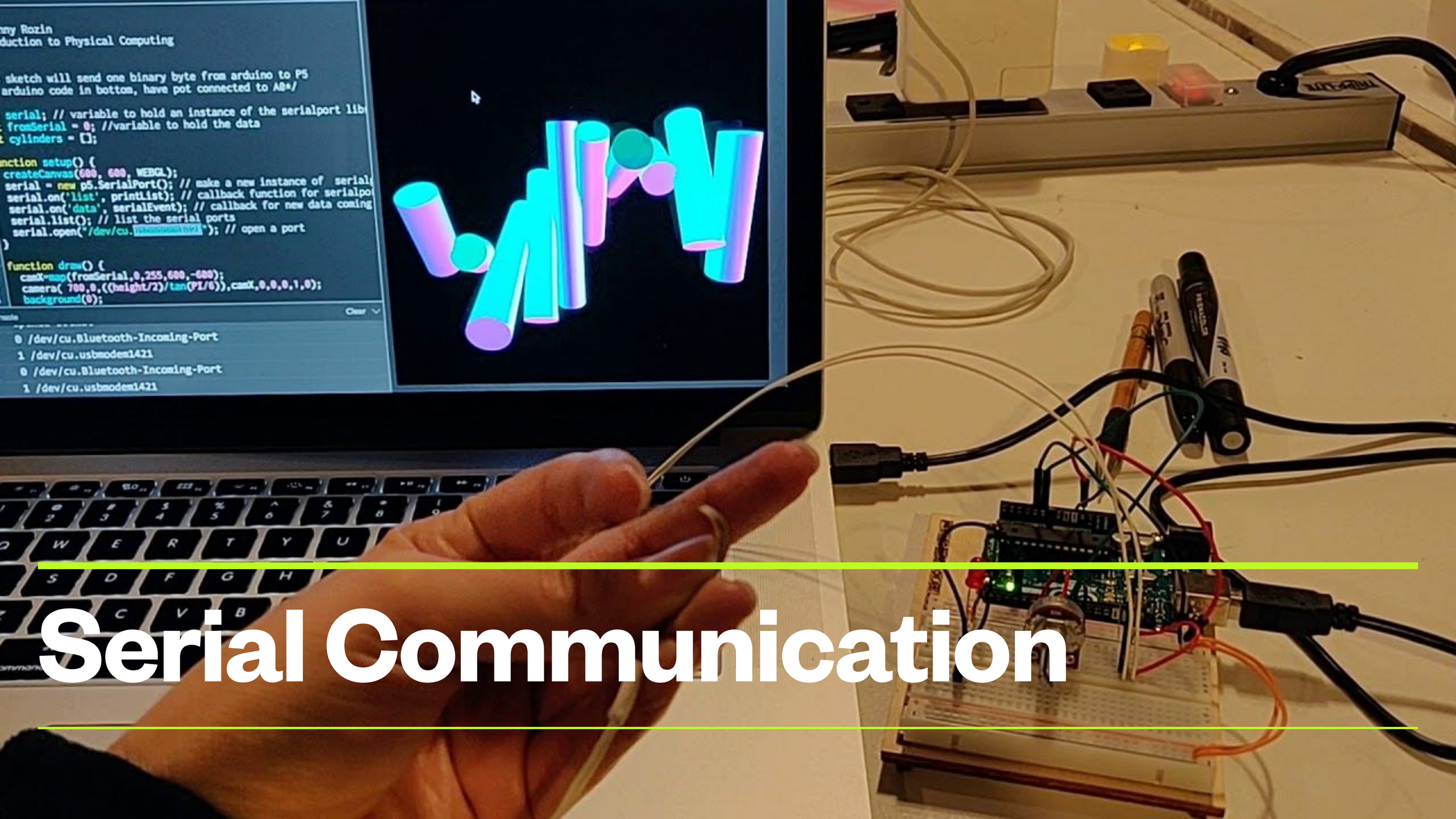
ROBERT HALL

Plan for Today

- Inspiration Time
- Homework review
- Serial Communication
- Addressable LEDs
- MVP Feedback
- Work on your projects!

Inspiration Time!

How was this done?



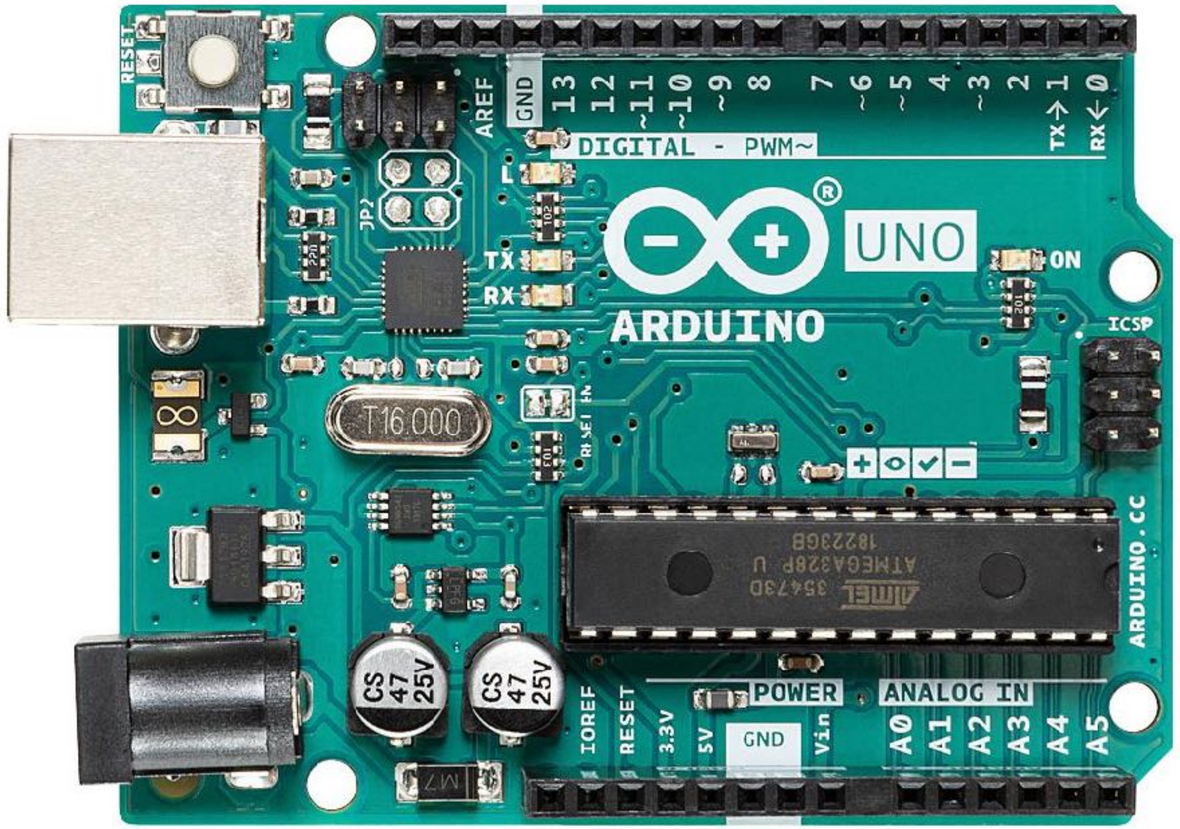
Serial Communication

Demo

Serial Monitor

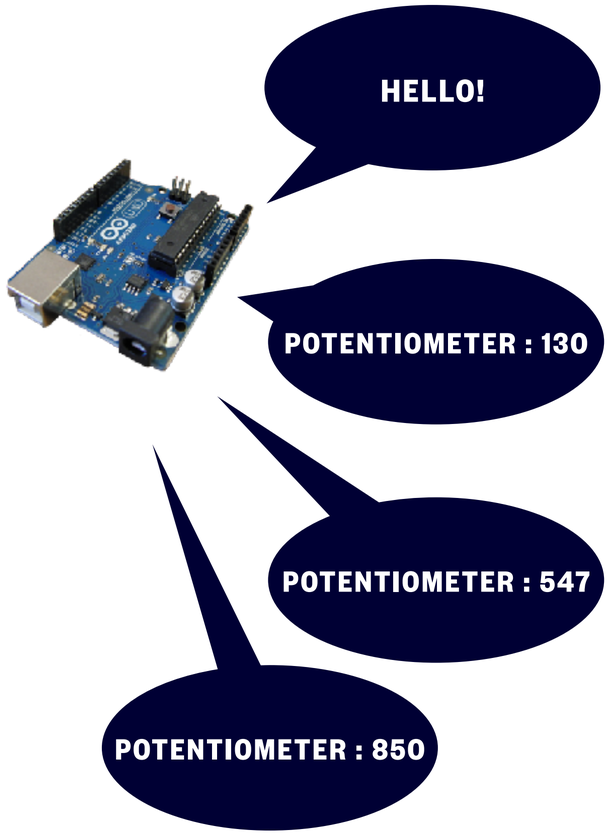
- We have used the serial monitor to listen to messages from our Arduino
- We will now start using other software to receive messages from our Arduino

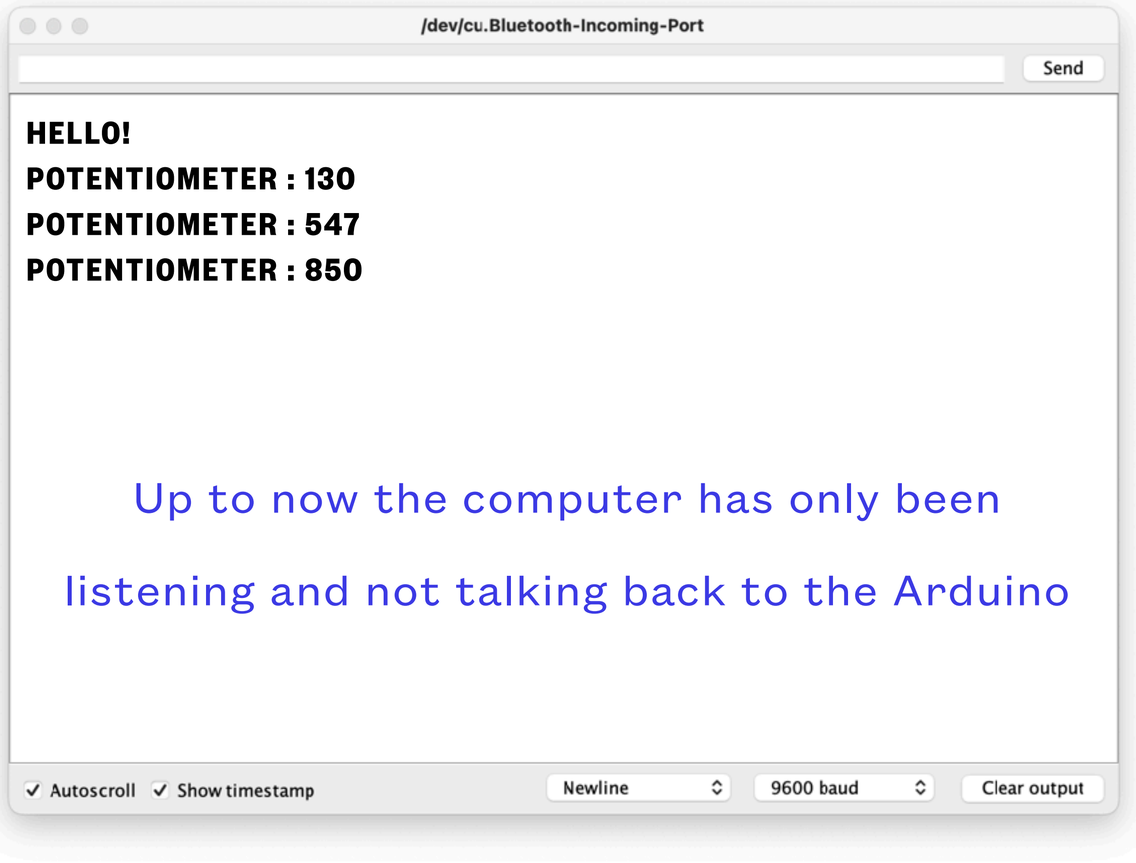
Arduino is running code



- It is important to remember that the Arduino is running code on it’s own
- The Arduino code is not running on the computer

Serial Communication is like a conversation!

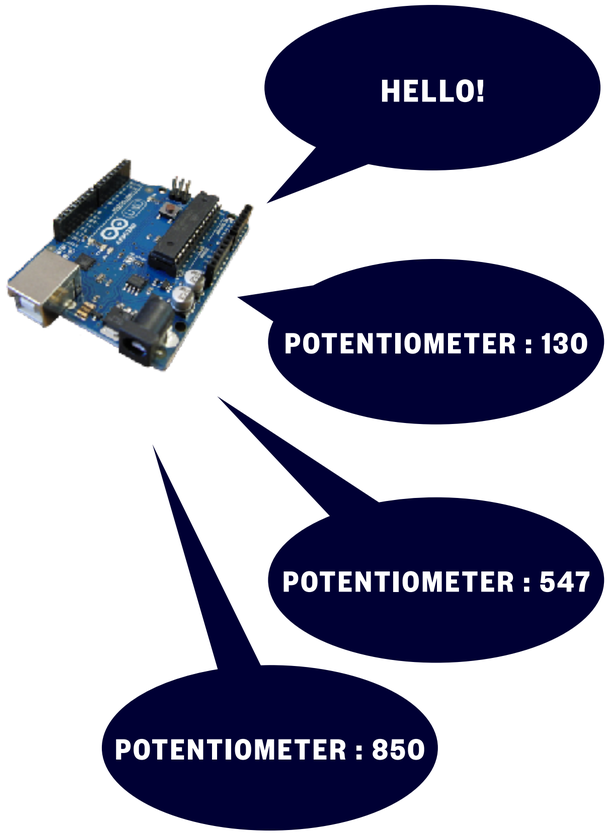


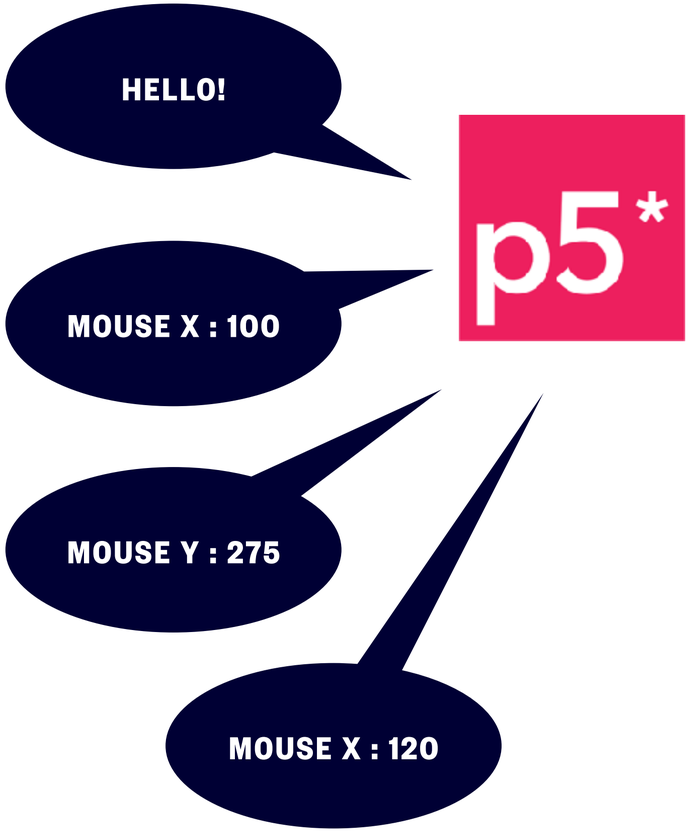


Up to now the computer has only been

listening and not talking back to the Arduino

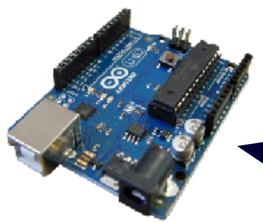
Serial Communication is like a conversation!





When we talk to something like p5.js we normally need messages going in both directions

Serial Communication is like a conversation!



HELLO!

POTENTIOMETER : 130

POTENTIOMETER : 547

POTENTIOMETER : 850

HELLO!

MOUSE X : 100

MOUSE Y : 275

MOUSE X : 120

p5*

Serial Communication is like a conversation!

- Unfortunately, because p5.js runs inside a browser, it can’t talk to the Arduino directly.
- In order to make this work we need a program to help us send messages between them.

Node.JS

Node.js is a framework that lets us write programs outside of a browser using JavaScript

It can speak to both the Arduino but also P5.js

You can download it here : https://nodejs.org/en/

Node.JS

- Node.JS is a very useful framework as it has lots of packages (like libraries) that it can use to talk to all sorts of different hardware and software.
- It is most commonly used as a web server, to host web pages.
- In Physical Computing 2, we use Node.JS as the glue to stitch our projects together.
- A project might have :
  - an openFrameworks Sketch
  - 3 ESP32s (Arduinos With Wifi)
  - a MIDI keyboard
  - a p5.js sketch that is accessible online.
- Node.js can talk to all of these things and coordinate them.

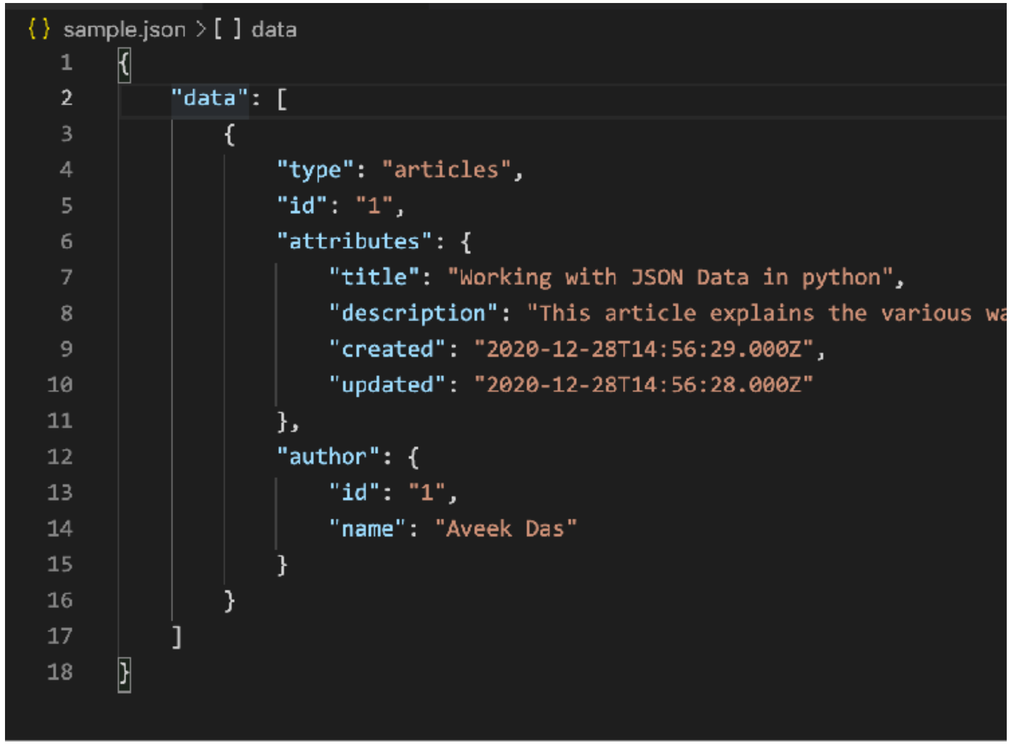
Node.JS

- In Physical Computing 1, we can keep it much more simple.
- I have written some Node.JS code that just sends anything it receives from Arduino to P5.JS or the other way around.
- This way it is like they are directly connected.

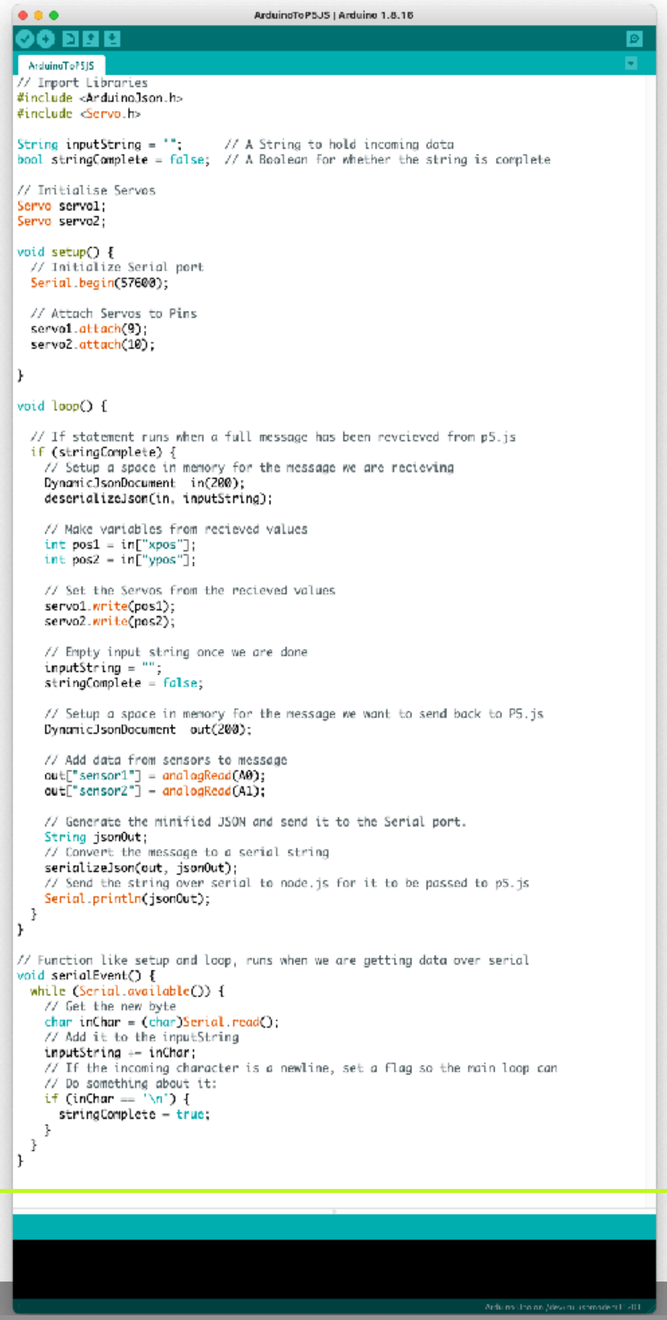
JSON

- As we are sending messages from Arduino (C++) to P5.JS (JavaScript) it can be difficult to format data in a way that both understand.
- To solve this problem we use a way of writing our data called JSON.
- This stands for JavaScript Object Notation.
- It is a structure we can use to send lots of variables in one message.

JSON



- JSON allows us to group elements together in objects just like you can in JavaScript.
- JSON is just a way of writing a JavaScript object out as text so we can read it.
- Luckily there is a way we can read and write this type of object on Arduino.



The Arduino Code

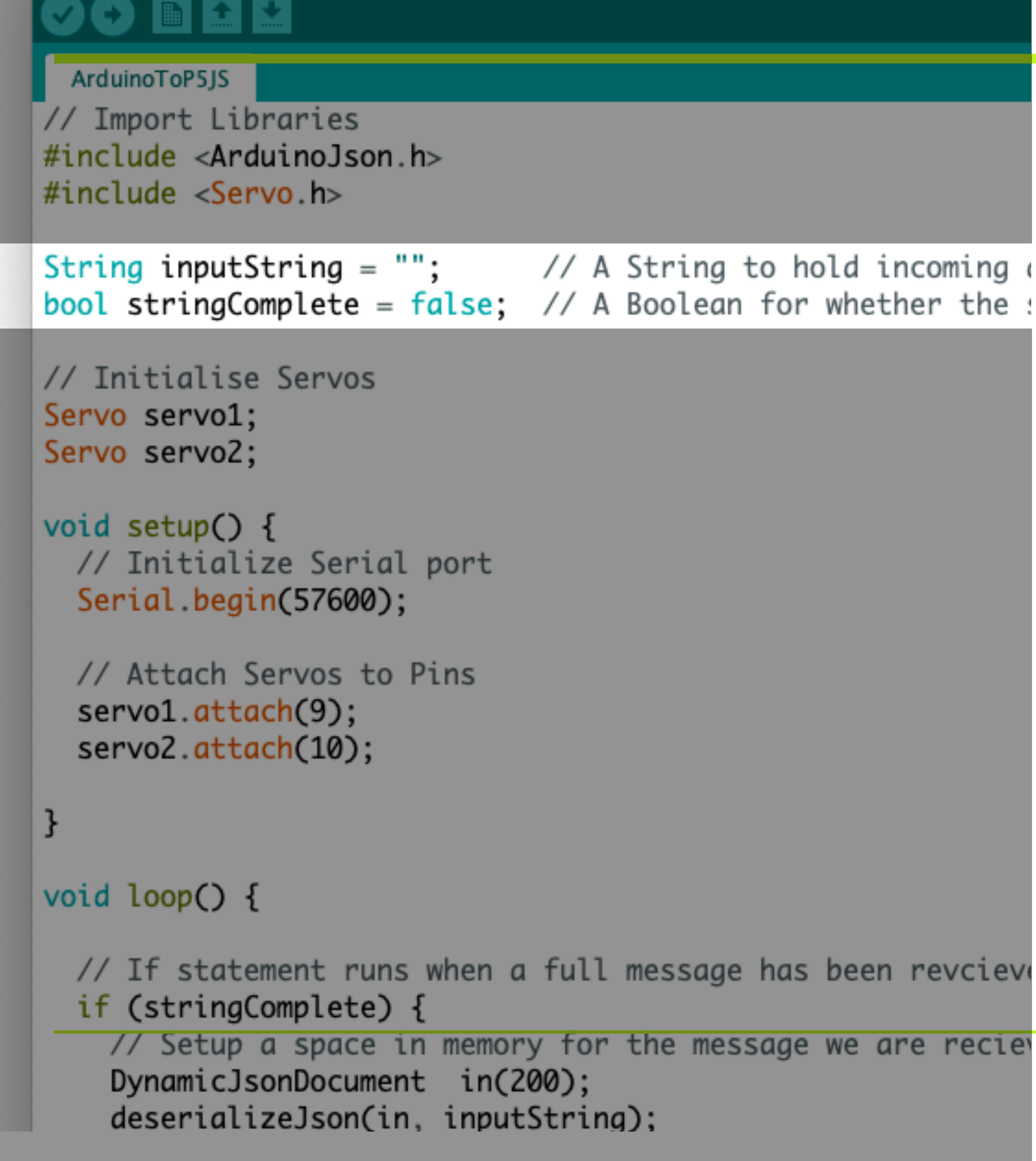
- This is all the code we need to send variables to and receive variables from P5.JS
- It looks like a lot but we can break it down step by step

ArduinoToP5JS

```
// Import Libraries
#include <ArduinoJson.h>
#include <Servo.h>

String inputString = "";      // A String to hold incoming
bool stringComplete = false;  // A Boolean for whether the

// Initialise Servos
Servo servo1;
Servo servo2;

void setup() {
  // Initialize Serial port
  Serial.begin(57600);

  // Attach Servos to Pins
  servo1.attach(9);
  servo2.attach(10);

}

void loop() {

  // If statement runs when a full message has been revcieve
  if (stringComplete) {
    // Setup a space in memory for the message we are recie
    DynamicJsonDocument  in(200);
    deserializeJson(in, inputString);
```

The Arduino Code

- Lets start with the libraries, we are using :
  - ArduinoJSON to construct JSON messages that JavaScript can understand
  - Servo to control a servo with the variables we receive from P5.JS



The Arduino Code

- Now we have a pair of variables.
- The first holds the message coming from P5.JS while it is being constructed, as we only receive one letter at a time.
- The second is a boolean that lets our program know when the message is complete and we can read it.

The Arduino Code

```
ArduinoToP5JS
// Import Libraries
#include <ArduinoJson.h>
#include <Servo.h>

String inputString = "";      // A String to hold incoming
bool stringComplete = false;  // A Boolean for whether the

// Initialise Servos
Servo servo1;
Servo servo2;

void setup() {
  // Initialize Serial port
  Serial.begin(57600);

  // Attach Servos to Pins
  servo1.attach(9);
  servo2.attach(10);

}

void loop() {

  // If statement runs when a full message has been revcieve
  if (stringComplete) {
    // Setup a space in memory for the message we are recie
    DynamicJsonDocument  in(200);
    deserializeJson(in, inputString);
```

- In setup() we initialise the serial port at 57600bps. This is so we can have a lot of data going quickly either way.
- Fun Fact, this is very close to the speed the internet used to be! (56kbps)
- We also assign the pins for our servos here.

```
    out["sensor2"] = analogRead(A1);

    // Generate the minified JSON and send it to the Serial
    String jsonOut;
    // Convert the message to a serial string
    serializeJson(out, jsonOut);
    // Send the string over serial to node.js for it to be p
    Serial.println(jsonOut);
  }
}

// Function like setup and loop, runs when we are getting da
void serialEvent() {
  while (Serial.available()) {
    // Get the new byte
    char inChar = (char)Serial.read();
    // Add it to the inputString
    inputString += inChar;
    // If the incoming character is a newline, set a flag so
    // Do something about it:
    if (inChar == '\n') {
      stringComplete = true;
    }
  }
}
```

The Arduino Code

- We are now going to look at the bottom of our code where we have a new function we haven't seen before.
- We remember that setup() runs once at the start of our code, and loop() runs continuously forever.
- serialEvent() runs once whenever there is new data from P5.JS

```
    out["sensor2"] = analogRead(A1);

    // Generate the minified JSON and send it to the Serial
    String jsonOut;
    // Convert the message to a serial string
    serializeJson(out, jsonOut);
    // Send the string over serial to node.js for it to be p
    Serial.println(jsonOut);
  }
}

// Function like setup and loop, runs when we are getting da
void serialEvent() {
  while (Serial.available()) {
    // Get the new byte
    char inChar = (char)Serial.read();
    // Add it to the inputString
    inputString += inChar;
    // If the incoming character is a newline, set a flag so
    // Do something about it:
    if (inChar == '\n') {
      stringComplete = true;
    }
  }
}
```

The Arduino Code

- Inside serialEvent() we have a while loop that loops while Serial.available() is true.
- This lets us run the same code over and over until we have read all the new data.
- The code adds the new data to our inputString variable which is where we are storing the message while it is being constructed.
- Lastly we check if the newest character is ‘\n’
  - If it is we know this is the end of the message as ‘\n’ means ‘New Line’
  - If this happens we set stringComplete to true, which lets the if statement in loop() run.

```
  servo1.attach(9);
  servo2.attach(10);

}

void loop() {

  // If statement runs when a full message has been revcieve
  if (stringComplete) {
    // Setup a space in memory for the message we are reciev
    DynamicJsonDocument  in(200);
    deserializeJson(in, inputString);

    // Make variables from recieved values
    int pos1 = in["xpos"];
    int pos2 = in["ypos"];

    // Set the Servos from the recieved values
    servo1.write(pos1);
    servo2.write(pos2);

    // Empty input string once we are done
    inputString = "";
    stringComplete = false;

    // Setup a space in memory for the message we want to se
    DynamicJsonDocument  out(200);

    // Add data from sensors to message
    out["sensor1"] = analogRead(A0);
    out["sensor2"] = analogRead(A1);
```

The Arduino Code

- Back to loop() now and we can see an if statement with stringComplete as the argument.
- This is set to true every time we have a new complete message, so this code inside only runs in this case.

```
  servo1.attach(9);
  servo2.attach(10);

}

void loop() {

  // If statement runs when a full message has been revcieve
  if (stringComplete) {
    // Setup a space in memory for the message we are reciev
    DynamicJsonDocument  in(200);
    deserializeJson(in, inputString);

    // Make variables from recieved values
    int pos1 = in["xpos"];
    int pos2 = in["ypos"];

    // Set the Servos from the recieved values
    servo1.write(pos1);
    servo2.write(pos2);

    // Empty input string once we are done
    inputString = "";
    stringComplete = false;

    // Setup a space in memory for the message we want to se
    DynamicJsonDocument  out(200);

    // Add data from sensors to message
    out["sensor1"] = analogRead(A0);
    out["sensor2"] = analogRead(A1);
```

The Arduino Code

- Now we have some new code.
- These two statements are from the ArduinoJSON library.
- DynamicJsonDocument in(200); makes something a bit like an array, that we can use to convert the message into lots of variables.
- in is the name of the ‘array’ and it has a size of (200) which is the maximum size of our message in memory.

```
  servo1.attach(9);
  servo2.attach(10);

}

void loop() {

  // If statement runs when a full message has been revcieve
  if (stringComplete) {
    // Setup a space in memory for the message we are reciev
    DynamicJsonDocument  in(200);
    deserializeJson(in, inputString);

    // Make variables from recieved values
    int pos1 = in["xpos"];
    int pos2 = in["ypos"];

    // Set the Servos from the recieved values
    servo1.write(pos1);
    servo2.write(pos2);

    // Empty input string once we are done
    inputString = "";
    stringComplete = false;

    // Setup a space in memory for the message we want to se
    DynamicJsonDocument  out(200);

    // Add data from sensors to message
    out["sensor1"] = analogRead(A0);
    out["sensor2"] = analogRead(A1);
```

The Arduino Code

- Now that we have our array ready we can use deserialiseJson() to convert the message into our array.
- We can then retrieve our variables using the names we give them in P5.JS (We will see this bit later!)

```
  servo1.attach(9);
  servo2.attach(10);

}

void loop() {

  // If statement runs when a full message has been revcieve
  if (stringComplete) {
    // Setup a space in memory for the message we are reciev
    DynamicJsonDocument  in(200);
    deserializeJson(in, inputString);

    // Make variables from recieved values
    int pos1 = in["xpos"];
    int pos2 = in["ypos"];

    // Set the Servos from the recieved values
    servo1.write(pos1);
    servo2.write(pos2);

    // Empty input string once we are done
    inputString = "";
    stringComplete = false;

    // Setup a space in memory for the message we want to se
    DynamicJsonDocument  out(200);

    // Add data from sensors to message
    out["sensor1"] = analogRead(A0);
    out["sensor2"] = analogRead(A1);
```

The Arduino Code

- Once we have these as variables we can use them however we like, in this case to control servos.

```
    // Make variables from recieved values
    int pos1 = in["xpos"];
    int pos2 = in["ypos"];

    // Set the Servos from the recieved values
    servo1.write(pos1);
    servo2.write(pos2);

    // Empty input string once we are done
    inputString = "";
    stringComplete = false;

    // Setup a space in memory for the message we want to se
    DynamicJsonDocument  out(200);

    // Add data from sensors to message
    out["sensor1"] = analogRead(A0);
    out["sensor2"] = analogRead(A1);

    // Generate the minified JSON and send it to the Serial
    String jsonOut;
    // Convert the message to a serial string
    serializeJson(out, jsonOut);
    // Send the string over serial to node.js for it to be p
    Serial.println(jsonOut);
  }
}

// Function like setup and loop, runs when we are getting da
```

The Arduino Code

- Now that have received our message and used it to do something, we can reset our variables for receiving by setting the input string to “” which empties it and setting stringComplete to false.

```
    // Make variables from recieved values
    int pos1 = in["xpos"];
    int pos2 = in["ypos"];

    // Set the Servos from the recieved values
    servo1.write(pos1);
    servo2.write(pos2);

    // Empty input string once we are done
    inputString = "";
    stringComplete = false;

    // Setup a space in memory for the message we want to se
    DynamicJsonDocument  out(200);

    // Add data from sensors to message
    out["sensor1"] = analogRead(A0);
    out["sensor2"] = analogRead(A1);

    // Generate the minified JSON and send it to the Serial
    String jsonOut;
    // Convert the message to a serial string
    serializeJson(out, jsonOut);
    // Send the string over serial to node.js for it to be p
    Serial.println(jsonOut);
  }
}

// Function like setup and loop, runs when we are getting da
```

The Arduino Code

- Now that we have finished receiving a message, we can send one back.
- We do this much the same way but in reverse.

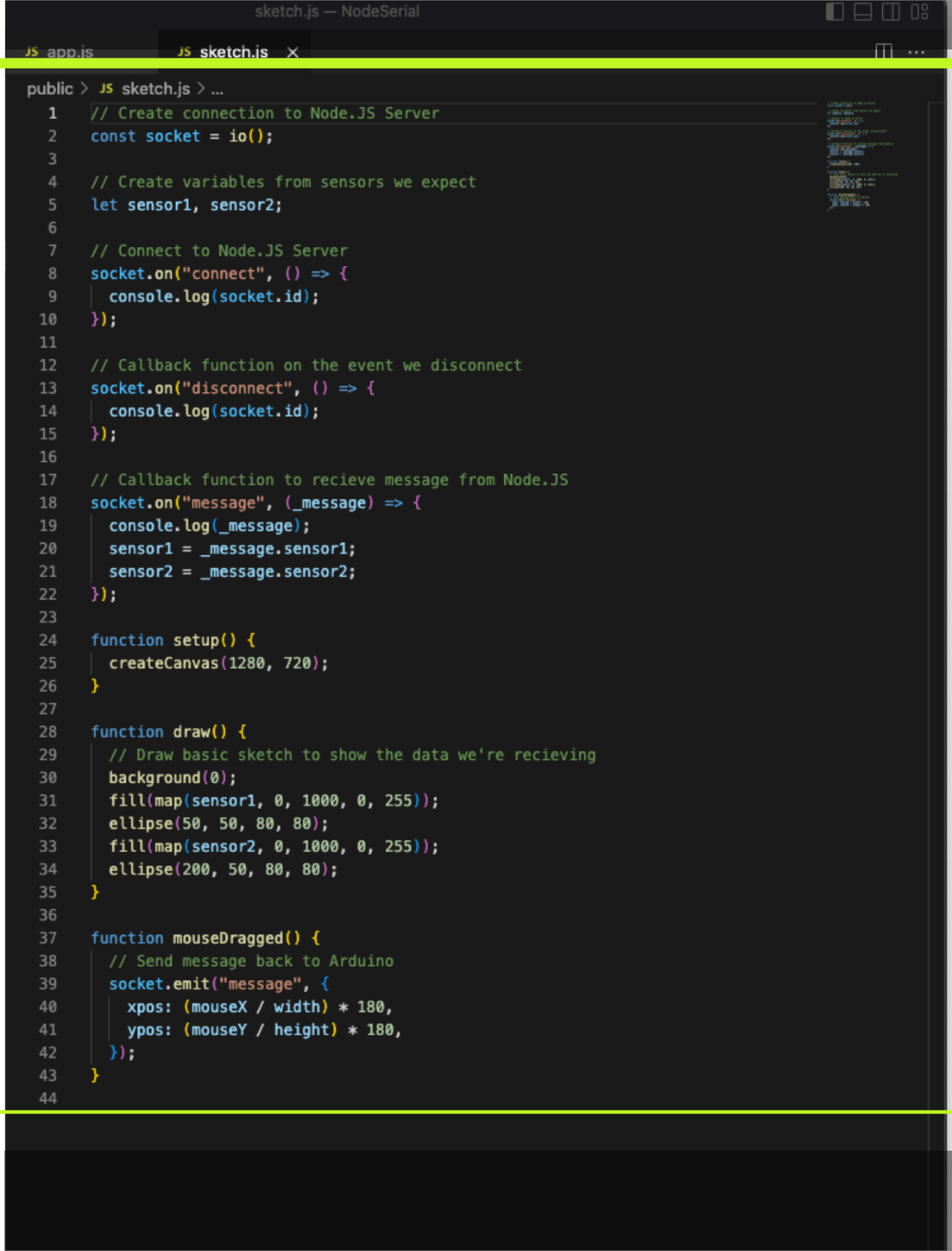
```
    // Make variables from recieved values
    int pos1 = in["xpos"];
    int pos2 = in["ypos"];

    // Set the Servos from the recieved values
    servo1.write(pos1);
    servo2.write(pos2);

    // Empty input string once we are done
    inputString = "";
    stringComplete = false;

    // Setup a space in memory for the message we want to se
    DynamicJsonDocument  out(200);

    // Add data from sensors to message
    out["sensor1"] = analogRead(A0);
    out["sensor2"] = analogRead(A1);

    // Generate the minified JSON and send it to the Serial
    String jsonOut;
    // Convert the message to a serial string
    serializeJson(out, jsonOut);
    // Send the string over serial to node.js for it to be p
    Serial.println(jsonOut);
  }
}

// Function like setup and loop, runs when we are getting da
```

The Arduino Code

- First we create a space for our JSON object
- Then we add data from sensors
- Then we make a string and use serializeJson() to make it into a message we can send over serial
- Then we send it with Serial.println()

The P5.JS Code

- Now we can take a look at the P5.JS Code
- We are now looking at Visual Studio Code, instead of the Arduino IDE, because we are now coding in JavaScript
- You can download Visual Studio Code here https://code.visualstudio.com/



The P5.JS Code

- Let’s start at the top
- Because we are using a Node.JS program to link Arduino and P5, we first need to connect to the Node.JS program.
- We do this using something called a socket.

app.js | sketch.js

public > sketch.js > ...

```
1   // Create connection to Node.JS Server
2   const socket = io();
3
4   // Create variables from sensors we expect
5   let sensor1, sensor2;
6
7   // Connect to Node.JS Server
8   socket.on("connect", () => {
9     console.log(socket.id);
10  });
11
12  // Callback function on the event we disconnect
13  socket.on("disconnect", () => {
14    console.log(socket.id);
15  });
16
17  // Callback function to recieve message from Node.JS
18  socket.on("message", (_message) => {
19    console.log(_message);
20    sensor1 = _message.sensor1;
21    sensor2 = _message.sensor2;
22  });
23
24  function setup() {
25    createCanvas(1280, 720);
26  }
```

The P5.JS Code

- Next up we make variables for the data we are sending from the Arduino

app.js | sketch.js

public > sketch.js > ...

```
1   // Create connection to Node.JS Server
2   const socket = io();
3
4   // Create variables from sensors we expect
5   let sensor1, sensor2;
6
7   // Connect to Node.JS Server
8   socket.on("connect", () => {
9     console.log(socket.id);
10  });
11
12  // Callback function on the event we disconnect
13  socket.on("disconnect", () => {
14    console.log(socket.id);
15  });
16
17  // Callback function to recieve message from Node.JS
18  socket.on("message", (_message) => {
19    console.log(_message);
20    sensor1 = _message.sensor1;
21    sensor2 = _message.sensor2;
22  });
23
24  function setup() {
25    createCanvas(1280, 720);
26  }
```

The P5.JS Code

- Now we encounter something called a callback function.
- This is a type of function we define to decide what to do when something happens.
- In this case we have callbacks for connecting and disconnecting from Node.JS

app.js | sketch.js

public > sketch.js > ...

```
1  // Create connection to Node.JS Server
2  const socket = io();
3
4  // Create variables from sensors we expect
5  let sensor1, sensor2;
6
7  // Connect to Node.JS Server
8  socket.on("connect", () => {
9    console.log(socket.id);
10 });
11
12 // Callback function on the event we disconnect
13 socket.on("disconnect", () => {
14   console.log(socket.id);
15 });
16
17 // Callback function to recieve message from Node.JS
18 socket.on("message", (_message) => {
19   console.log(_message);
20   sensor1 = _message.sensor1;
21   sensor2 = _message.sensor2;
22 });
23
24 function setup() {
25   createCanvas(1280, 720);
26 }
```

The P5.JS Code

- Now our important callback function, this one handles what happens when we get a message from the Arduino.
- First we console.log() the message so we can see everything we have received.
- Then we set our two sensor variables to the elements from the object.
- This is easier in JavaScript because JSON is native to JavaScript.

```
8   socket.on("connect", () => {
9     console.log(socket.id);
10  });
11
12  // Callback function on the event we disconnect
13  socket.on("disconnect", () => {
14    console.log(socket.id);
15  });
16
17  // Callback function to recieve message from Node.JS
18  socket.on("message", (_message) => {
19    console.log(_message);
20    sensor1 = _message.sensor1;
21    sensor2 = _message.sensor2;
22  });
23
24  function setup() {
25    createCanvas(1280, 720);
26  }
27
28  function draw() {
29    // Draw basic sketch to show the data we're recieving
30    background(0);
31    fill(map(sensor1, 0, 1000, 0, 255));
32    ellipse(50, 50, 80, 80);
33    fill(map(sensor2, 0, 1000, 0, 255));
34    ellipse(200, 50, 80, 80);
35  }
36
```

The P5.JS Code

- Now we can go on with the rest of the code.
- In this case we can use our sensor1 and sensor2 variables to control the colour of two ellipses on the screen.

```
19    console.log(_message);
20    sensor1 = _message.sensor1;
21    sensor2 = _message.sensor2;
22  });
23
24  function setup() {
25    createCanvas(1280, 720);
26  }
27
28  function draw() {
29    // Draw basic sketch to show the data we're recieving
30    background(0);
31    fill(map(sensor1, 0, 1000, 0, 255));
32    ellipse(50, 50, 80, 80);
33    fill(map(sensor2, 0, 1000, 0, 255));
34    ellipse(200, 50, 80, 80);
35  }
36
37  function mouseDragged() {
38    // Send message back to Arduino
39    socket.emit("message", {
40      xpos: (mouseX / width) * 180,
41      ypos: (mouseY / height) * 180,
42    });
43  }
44
```

The P5.JS Code

- Lastly use the mouseDragged() function.
- This runs when the mouse is moved while it is pressed.
- Here we use socket.emit() to send a message back to our Arduino.
- We have 2 arguments in socket.emit(), first the title of the type of message (for now this is always “message”) then the message itself in JSON.
- We send mouseX and mouseY but scaled to both be between 0 and 180 for our servo.
- The message we send is JSON, let’s look a little closer.

```
26    }
27
28    function draw() {
29      // Draw basic sketch to show the data we're recieving
30      background(0);
31      fill(map(sensor1, 0, 1000, 0, 255));
32      ellipse(50, 50, 80, 80);
33      fill(map(sensor2, 0, 1000, 0, 255));
34      ellipse(200, 50, 80, 80);
35    }
36
37    function mouseDragged() {
38      // Send message back to Arduino
39      socket.emit("message", {
40        xpos: (mouseX / width) * 180,
41        ypos: (mouseY / height) * 180,
42      });
43    }
44
```

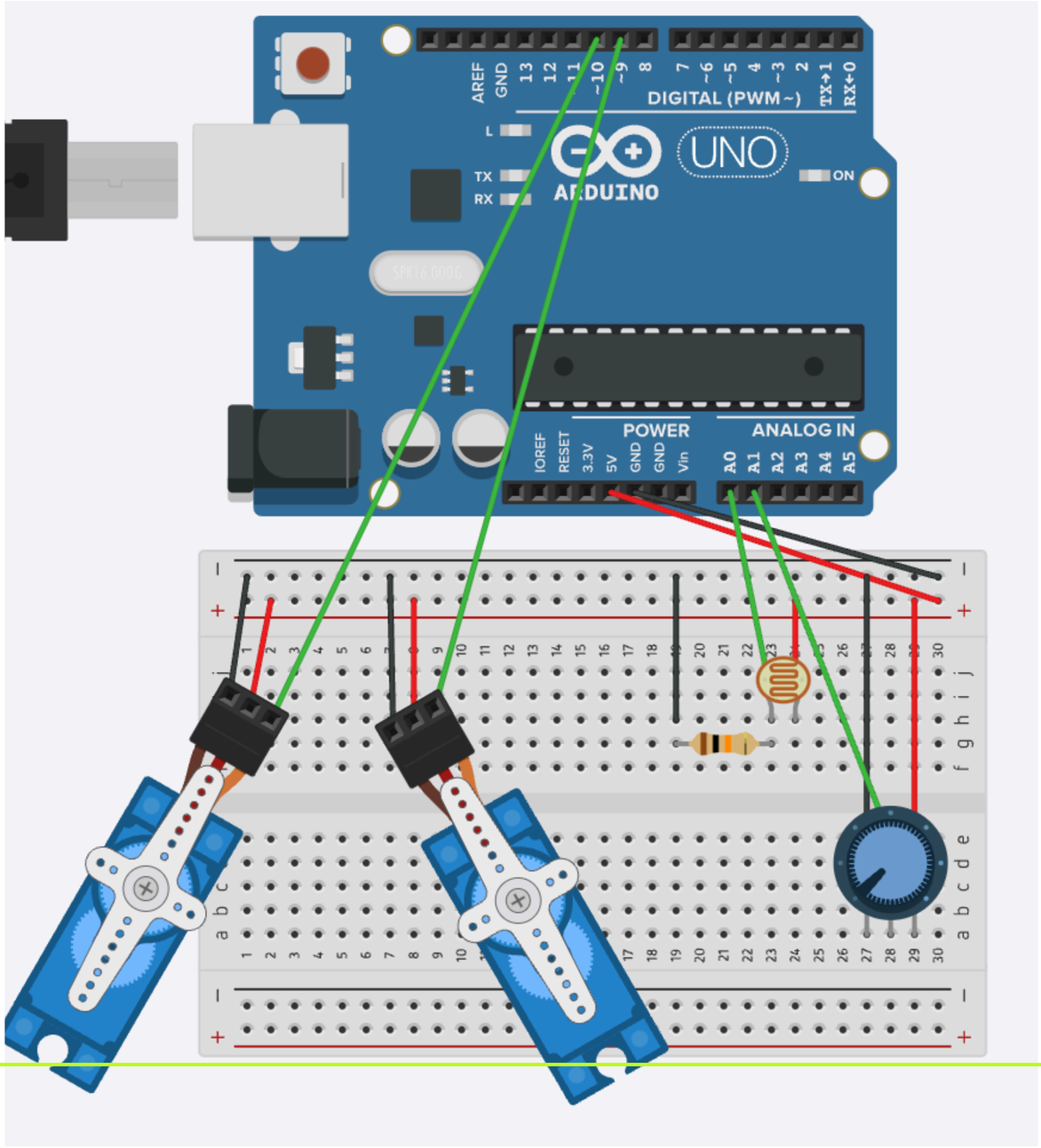
The P5.JS Code

- Rather than = we use : to set elements in a JSON object
- For each of xpos and ypos, we divide the mouse value by width or height to get a number between 0 and 1, then multiply that by 180 to get a number between 0 and 180.
- We use , after each variable we want to send.

```
{
    xpos: (mouseX / width) * 180,
    ypos: (mouseY / height) * 180,
}
```

The Arduino Circuit

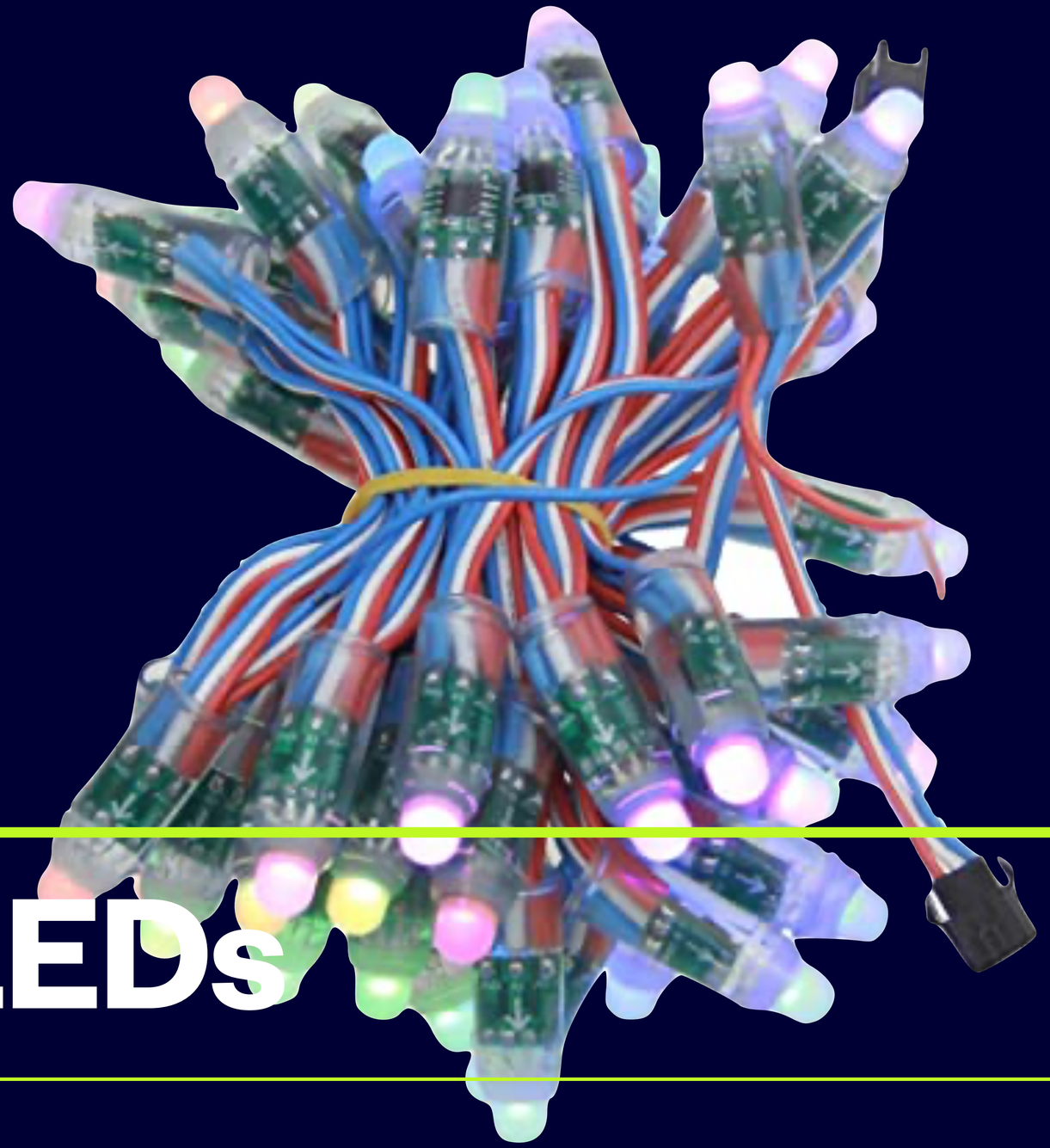
- If you want to have a go at getting this set up you can build the Arduino circuit here (it will work with only one servo but you can borrow one if you want to try both)



To Do List

- Download Code Folder from VLE
- Install Node.JS
- Install ArduinoJson Library
- Upload Arduino Code
- Change Arduino Port in app.js
  - Check port by looking at title of serial monitor
  - Close Serial Monitor
- Run app.js (node app.js)
- Remember to quit node.js using Ctrl + C on PC and Mac!

Demo



Addressable LEDs

Addressable LED Presentation

Work on your project!

- We’re here to help!
- Ask us anything!
- Idea or Tech Help!
- This is your moment to get help with anything you have been struggling with in your MVP.

This Weeks Assignment : MVP

- Try to get a feeling for what your project will look like
- Try out your idea
- Make a basic prototype quickly
- Fake the stuff you can’t make yet
- Try to test it on someone
- Build what you can with what you have
- Use sensors and actuators you have until you have the ones you need

This Weeks Assignment : MVP

- Taking into account the feedback you received today, make a working MVP of your final project
- Please post a video on the forum and the VLE submission page
- Please also submit the code and circuit diagram on the submission page
- Monday Group : 12th November
- Wednesday Group : 14th November