

UBC ATSC 303 - 2022W (March 2023)
Lab 11 – Arduino & Air Quality Lab (/38)

Guest lecturers: Chris Rodell & Reagan McKinney

Teaching assistant: Davi Monticelli

Email: dmonticelli@eoas.ubc.ca

Learning goals

1. Learn how to install the Arduino software and its dependencies.
2. Write sketches (small programs) and upload them to a microcontroller board ([CPE](#)).
3. Understand how particulate matter is measured using an optical particle counter.
4. Understand how data can be wirelessly transmitted between an instrument and data logger.
5. Conduct a calibration of the [adafruit built sensors](#) with the [Sensit RAMP AQ sensor](#).

Background and Resources

Quick videos on common electronic components:

<https://learn.adafruit.com/groups/circuit-playground>

Safety

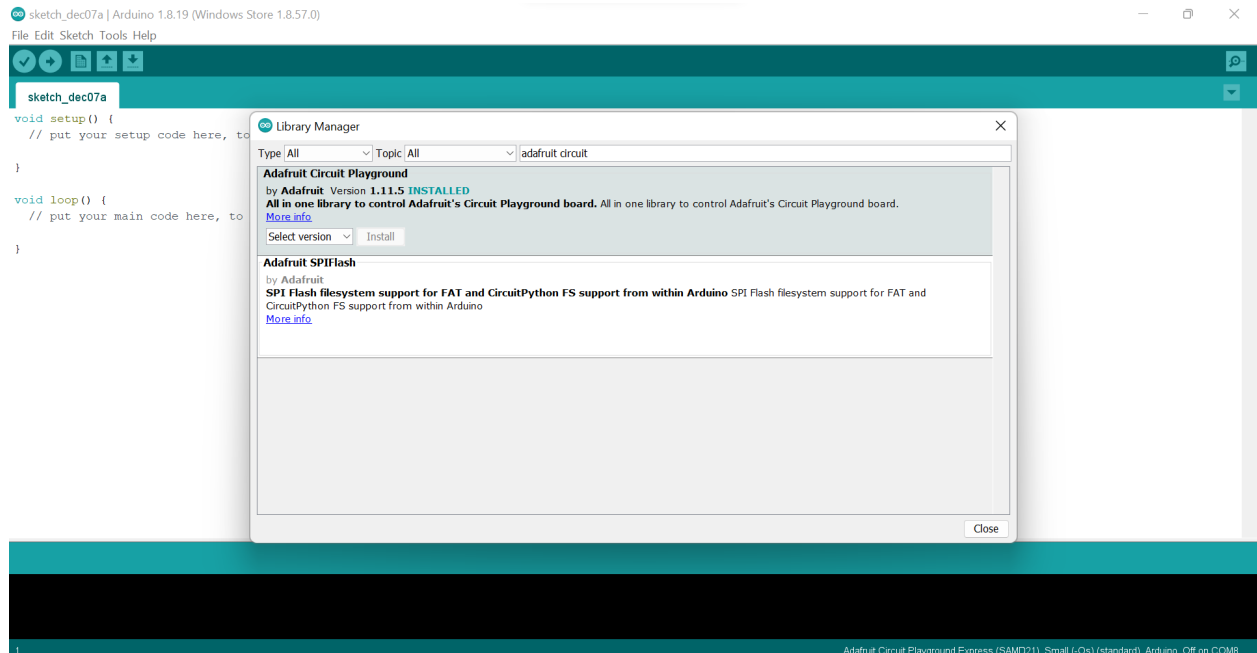
- We are working with electronics, do not put hands in sockets

Method

Part 0 – Install Arduino and Dependencies

Equipment:

- Personal laptop: One per group of 4.
 - Internet connection.
- 1) Download the [Arduino IDE software](#) for your specific machine's OS
<https://www.arduino.cc/en/software>
 - 2) **Launch** the Arduino IDE
 - 3) We will also need to download several libraries for this lab. To download libraries go to **Tools->Manage Libraries...** To install a package, simply hover over the box with the package name you want. A button to *install* will appear. Click it. This can take some time. Use the search bar to install the following:
 - a) Adafruit Circuit Playground



Part 1 – Circuit Playground Express

Equipment for each Group:

- 1 x circuit playground express
- micro-USB to USB cable
- Laptop (yours)

What are microcontroller boards?

Microcontrollers contain a processor that executes code and a set of circuits called 'peripherals' that provide additional functions: USB, serial interfaces, ADCs, timers, and so on. The biggest difference between microcontrollers is the relationship between the processor, the peripherals, and the physical pins that come out of the package.

The microcontrollers we will use in this lab both use 32-bit SAMD21G microcontrollers. You can read more about the specifications of these boards here:

<https://learn.adafruit.com/how-to-choose-a-microcontroller/the-microcontrollers-in-adafruit-products>.

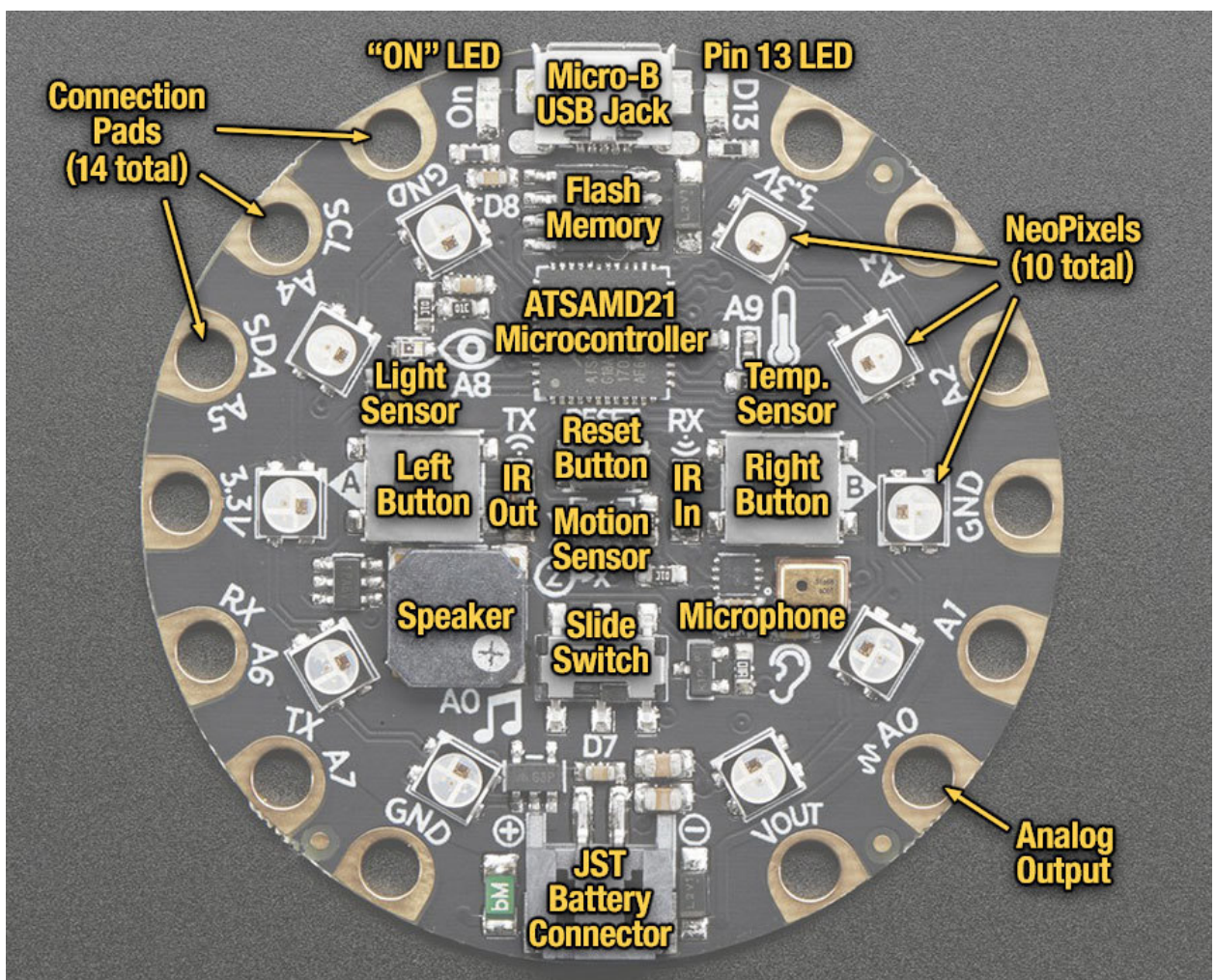
Specifically, we will use the Circuit Playground Express (CPE) and the Feather M0. The CPE schematic is shown below. This is a powerful board with a lot of built-in functionality. This allows us to do a lot of experimentation with scripting utilizing only a single piece of hardware. Other boards, like the feather M0, were built barer, with the aim to have different modules attached based on need. We will see this applied in Part 2 of the lab.

It should be noted that these boards alone do not keep time. Once they are powered up, they will start counting the milliseconds since power was introduced. This can make it challenging to work with time as the amount of milliseconds it starts to record becomes massive. This is something we must keep in mind in meteorology as time is a very important component of what we do.

Circuit Playground Express:

https://www.mouser.com/datasheet/2/737/introducing_circuit_playground-1392960.pdf

In this Part, we will program the CPE to change colors when input is provided to the light sensor. We will also look at how to get the board to communicate with other CPE boards using INFRARED light.

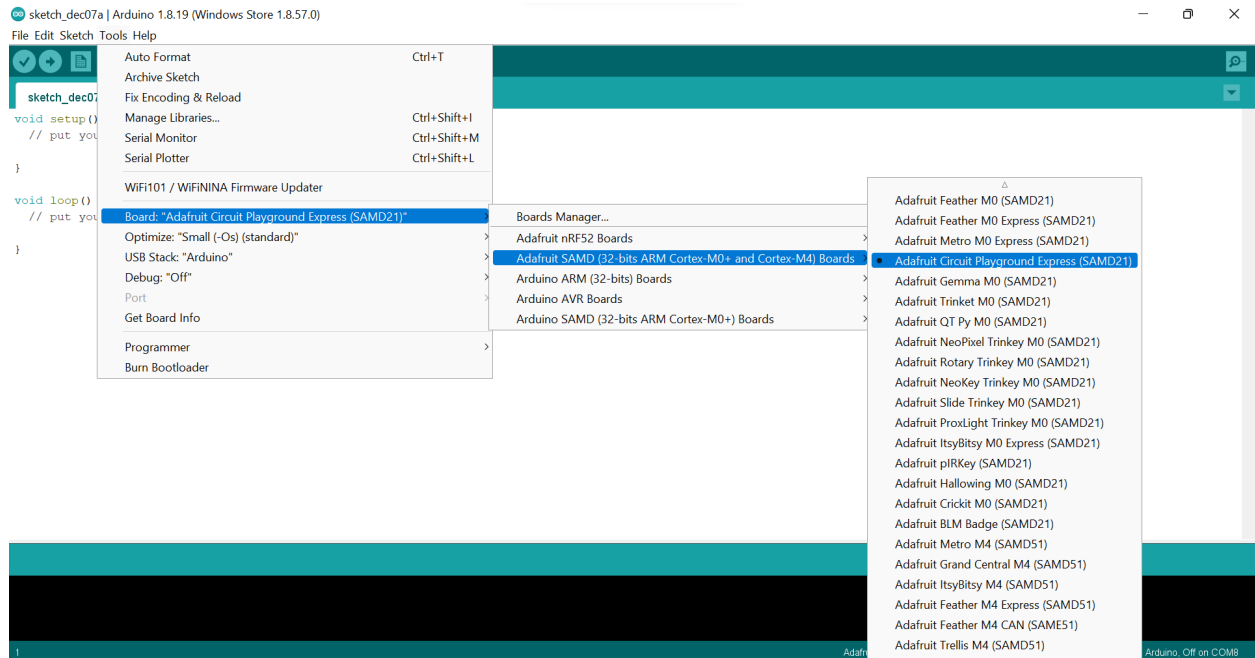


Part 1a. Arduino Sketches - Light Sensor

Since our board has everything built in already, we can get started with programming right away and do not have to worry about setting up the hardware first.

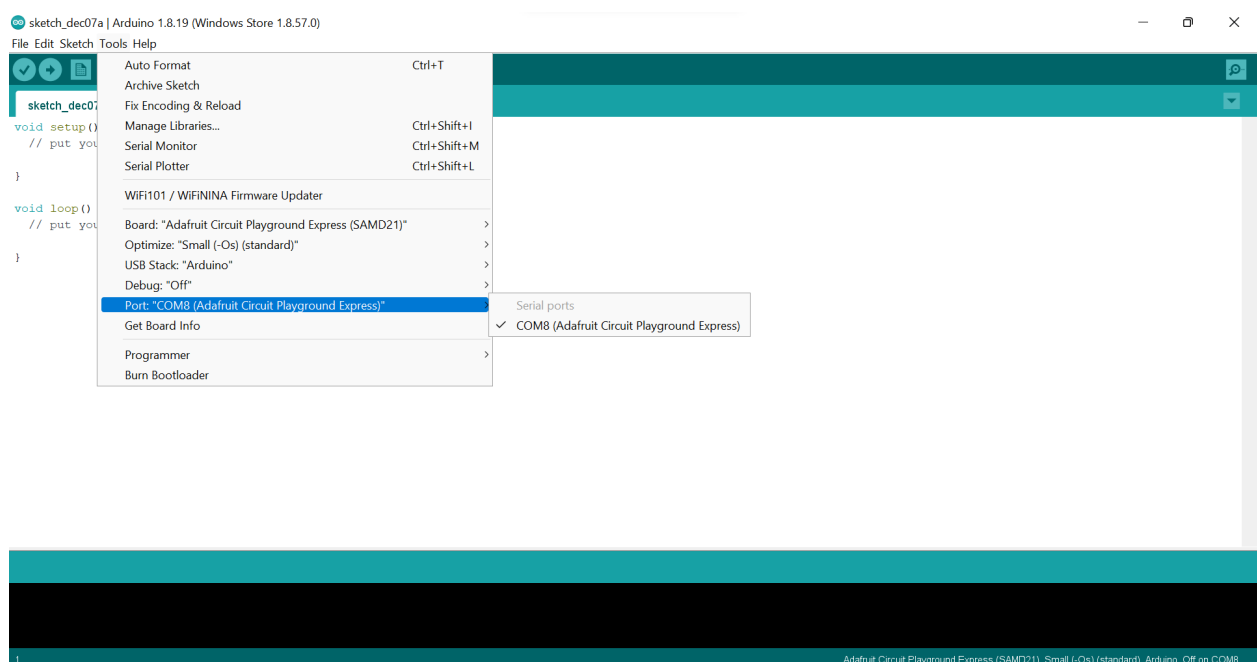
Before we start programming, though, it is best that we ensure our board is working properly. To do this, will we:

- a) Hook the CPE to our computer with the USB to micro-USB cable.
- b) Open up the Arduino IDE.
- c) Open up the **Tools->Board** menu to ensure we have the right board selected.



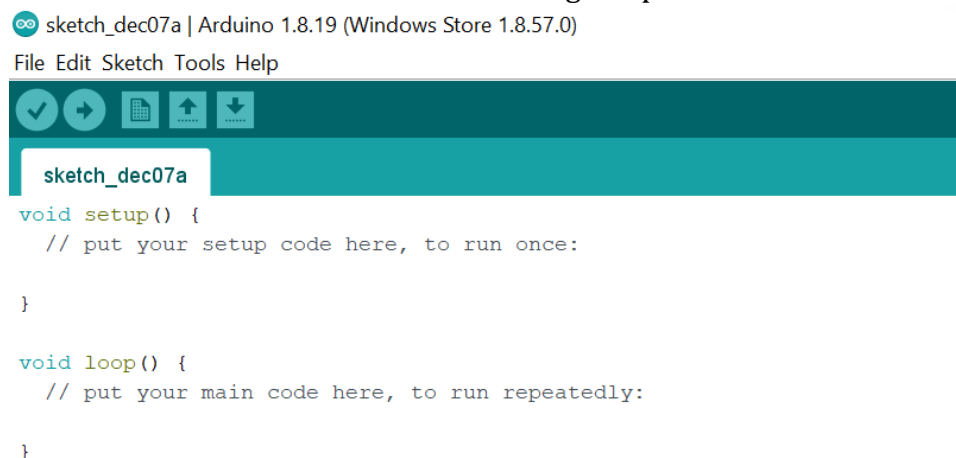
- d) Next, in **Tools->Port**. We need to make sure that the IDE is pointing to the port with our CPE. The CPE should have all green lights when it is connected. This indicates that it is ready to have code uploaded to it.

*If the lights are all red, the IDE or board is not set up right. Ask one of the TAs for help.



Now we can start working in the IDE. Arduino scripts (computer programs) use the C programming language and are called sketches.

Blank sketches start off with the following template:



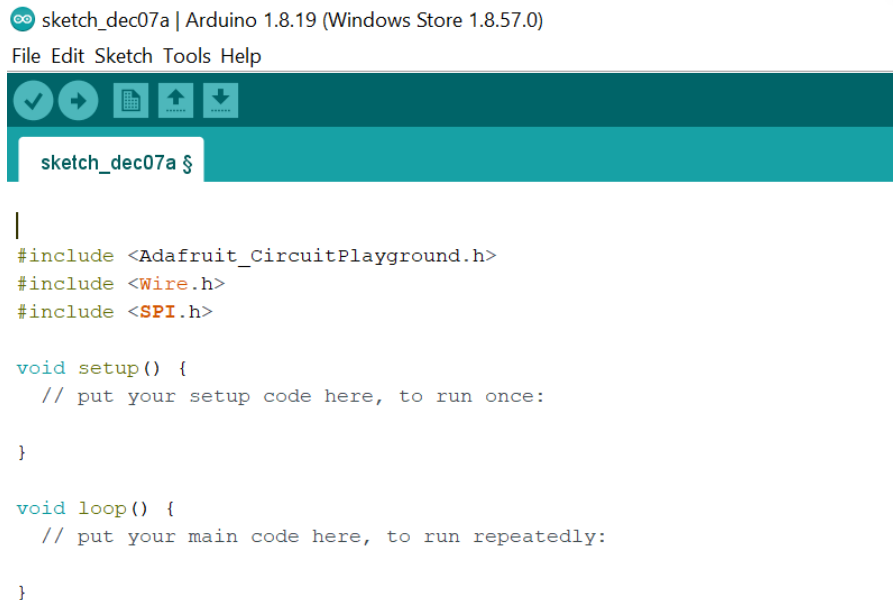
The first part is the setup code. These are processes that you want the board to start when power is introduced. Often this includes:

- A sleep period to allow the board to boot up properly;
- Starting the boards' initial clock;
- An indicator that the board is working properly (i.e, flash of an LED)

The second part of this template is loops. They run continuously and need to be properly defined to function correctly. There are lots of other void functions as well that we will see throughout this lab.

You probably also notice here the use of `//` this is used to indicate comments. Like any other coding language, comment early and often, and your future self will be much happier.

Before we start these code blocks though, we need to indicate the libraries we want to use. We will need three libraries, as shown below:



```
sketch_dec07a | Arduino 1.8.19 (Windows Store 1.8.57.0)
File Edit Sketch Tools Help

sketch_dec07a $

#include <Adafruit_CircuitPlayground.h>
#include <Wire.h>
#include <SPI.h>

void setup() {
  // put your setup code here, to run once:
}

void loop() {
  // put your main code here, to run repeatedly:
}
```

Next, we define the variables used in our code. This identifies the ports that the program will pertain to. Recalling from the diagram above, the CPE has 7 analog ports we can call upon.


A0 = Thermistor (temperature sensor)
A4 = Microphone (sound sensor)
A5 = Light sensor

And 4 ports for external connections:
A7 = pin #6
A9 = pin #9
A10 = pin #10
A11 = pin #12

These analog ports can send any number of values for communication.

We will focus on the **A5** (the light sensor) for now and call it **ANALOG_INPUT**. A5 can take on any value between 0-1023. Higher values are brighter light levels. We are more

interested in how it interacts with darkness, though, so we will stick to a lower range (0-200).



The screenshot shows the Arduino IDE interface. At the top, it says "sketch_dec07a | Arduino 1.8.19 (Windows Store 1.8.57.0)". Below that is a menu bar with "File", "Edit", "Sketch", "Tools", and "Help". A toolbar with icons for check, run, serial monitor, upload, and download is visible. The main text area contains the following code:

```
// Circuit Playground Analog Sensor Demo
// Shows how to read an analog sensor
// and convert the analog value to color and sound.

// Identify libraries needed for this sketch
#include <Adafruit_CircuitPlayground.h>
#include <Wire.h>
#include <SPI.h>
```

Now we will pick the colors we want the board to light-up when the light sensor is affected. The boards use an RGB (Red-Green-Blue) scheme for coloring, so we need to define how much of each of the three colors we want for different analog inputs. We will start with the following settings:

For **minimum** input: only **red** turned on.
For **maximum** input: only **blue** turned on.

```
#define COLOR_RED_MIN 255
#define COLOR_GREEN_MIN 0
#define COLOR_BLUE_MIN 0

#define COLOR_RED_MAX 0
#define COLOR_GREEN_MAX 0
#define COLOR_BLUE_MAX 255
```

Note that all these definitions start with a #. We will now move onto the **void setup()**. The first part of this is a sleep for the serial monitor, as mentioned before, this allows the board to startup properly. An arbitrary amount of seconds is chosen for this. We have chosen 9600ms or 9.6s.

After this startup, we will print some text to ensure that the board is responding appropriately. You can say anything from "Hi World!" to "This is CPE, I am running :)".

We also need to start up our library.


```

void setup( ) {
    Serial.begin(9600);    // Startup serial monitor after 9600ms
    Serial.println("Welcome to your CPE!");

    CircuitPlayground.begin( ); // Startup circuit playground
library
}

```

Note that all code that does not begin with a # needs to end with a semicolon ; . We also need to ensure that the void setup code is all contained within the curly brackets {}.

And finally the meat of our code: **void loop{ }**. The following code should all be contained within the curly brackets. Here we will:

- a) Get a value from the A5 port based on the amount of light it is receiving and print it out in the serial monitor.

```

void loop( ) {
    uint16_t value = analogRead(ANALOG_INPUT); //Get value from A5
input
    Serial.println(value, DEC); //Print value received from A5

```

The code above gives a good template of how the sketches work. The lines of code start with defining the variable type (**uint_16**), the variable name (**value**) and the function that produces the variable (**analogRead**). Here the variable value is a 16 bit integer that contains the info from port A5. Note here we are not saving the data anywhere. We are just printing it to the serial monitor using the **Serial.println()** function.

- b) Map the colors to the values we are reading in. We will also use a gamma function here to make the colors look better.

```

// Map the A5 value to a color using the range we provided
if(value < VALUE_MIN) value = VALUE_MIN;
else if(value > VALUE_MAX) value = VALUE_MAX;

int red = map(value, VALUE_MIN, VALUE_MAX, COLOR_RED_MIN, COLOR_RED_MAX);
int green = map(value,VALUE_MIN, VALUE_MAX, COLOR_GREEN_MIN, COLOR_GREEN,
MAX);
int blue = map(value, VALUE_MIN, VALUE_MAX, COLOR_BLUE_MIN, COLOR_BLUE_MAX);

// Smooth out color appearance with gamma correction
red = CircuitPlayground.gamma8(red);

```



```
green = CircuitPlayground.gamma8(green);  
blue = CircuitPlayground.gamma8(blue);
```

- c) Light up the pixels of our choosing with our newly defined color maps. The clear function is given first so that the colors can be produced on a clean slate.

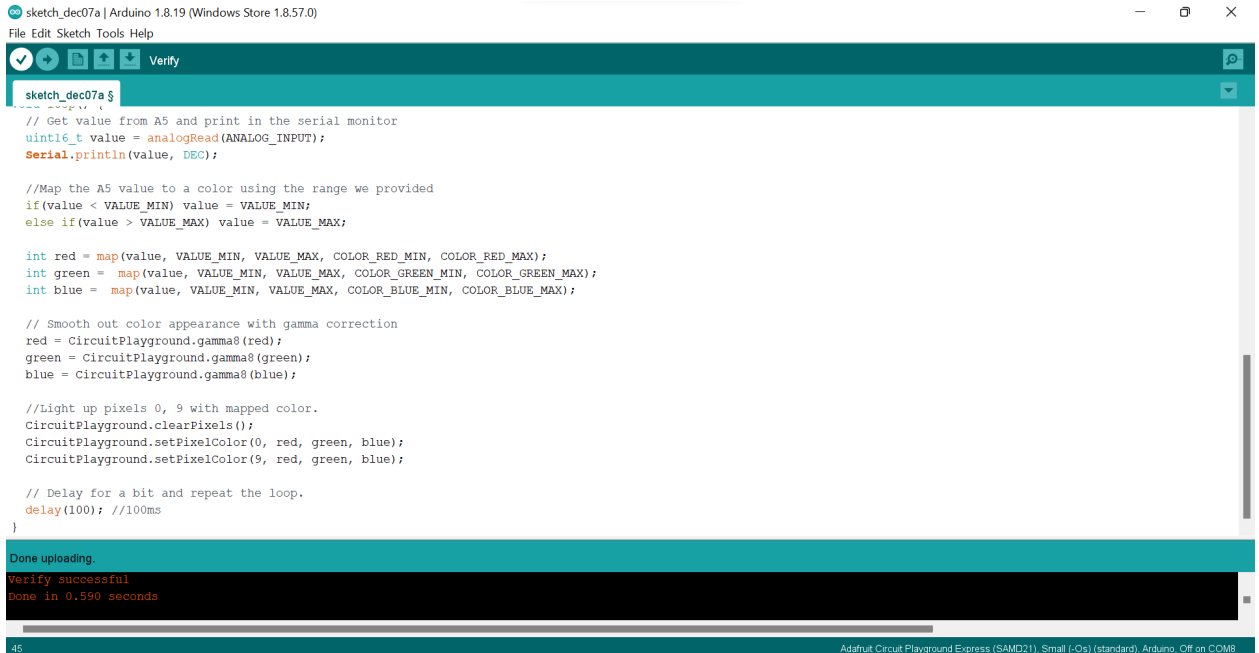
```
//Light up pixels 0, 9 with the mapped color  
CircuitPlayground.clearPixels( );  
CircuitPlayground.setPixelColor(0, red, green, blue);  
CircuitPlayground.setPixelColor(9, red, green, blue);
```

- d) Lastly, we will have the loop delay for 100 ms so that the board can reset itself before receiving more input.

```
//Delay for a bit and repeat the loop  
delay(100); //100ms  
} //Close void loop
```

Now we will save our project. Give it a good name you will remember. i.e. CPE_part1a.ino, where “ino” is the suffix for sketches.

Once we have it saved we can **verify** the code. Click the check-mark in the top left hand corner of the IDE.

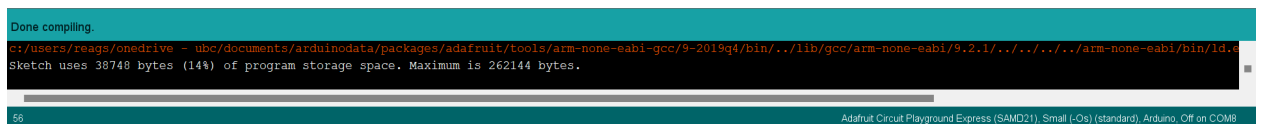


The bottom black panel will give you information on how the verification is going and when it is complete.



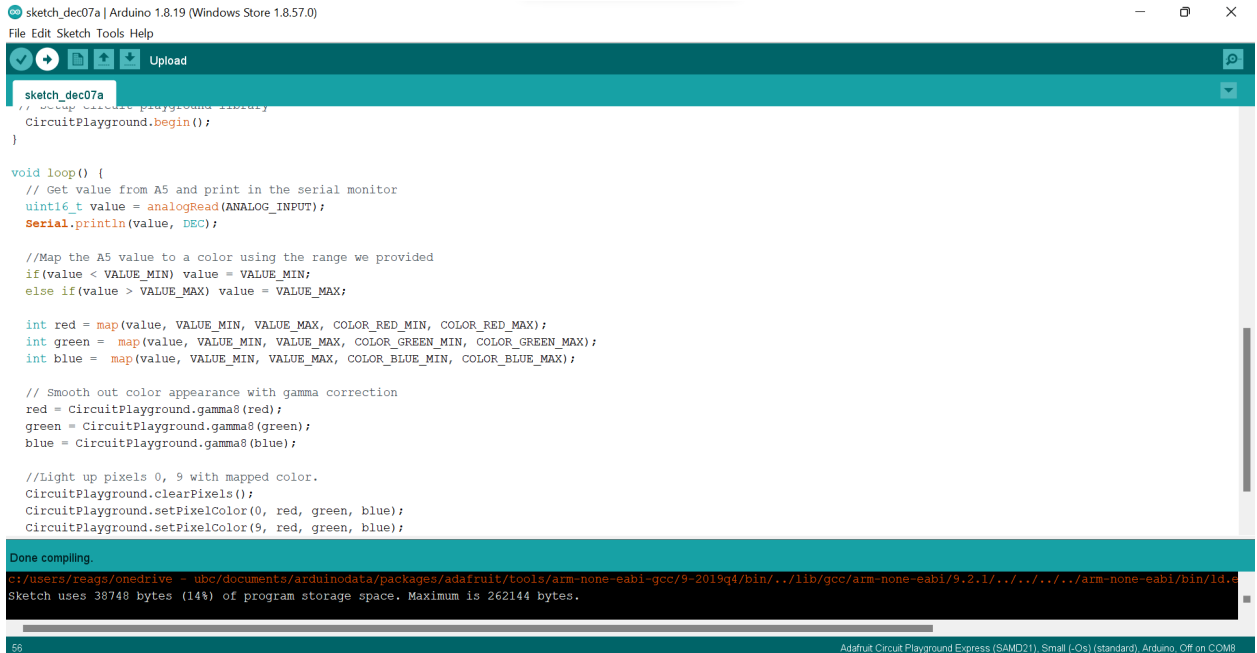
Common errors you may see:

- println is not defined, did you mean printn
- Spelling color as colour



Success!

Upload your code to the CPE. Click the **upload** button next to the verify button.



```
sketch_dec07a
// https://github.com/adafruit/circuit-playground-arduino
CircuitPlayground.begin();
}

void loop() {
  // Get value from A5 and print in the serial monitor
  uint16_t value = analogRead(ANALOG_INPUT);
  Serial.println(value, DEC);

  //Map the A5 value to a color using the range we provided
  if(value < VALUE_MIN) value = VALUE_MIN;
  else if(value > VALUE_MAX) value = VALUE_MAX;

  int red = map(value, VALUE_MIN, VALUE_MAX, COLOR_RED_MIN, COLOR_RED_MAX);
  int green = map(value, VALUE_MIN, VALUE_MAX, COLOR_GREEN_MIN, COLOR_GREEN_MAX);
  int blue = map(value, VALUE_MIN, VALUE_MAX, COLOR_BLUE_MIN, COLOR_BLUE_MAX);

  // Smooth out color appearance with gamma correction
  red = CircuitPlayground.gamma8(red);
  green = CircuitPlayground.gamma8(green);
  blue = CircuitPlayground.gamma8(blue);

  //Light up pixels 0, 9 with mapped color.
  CircuitPlayground.clearPixels();
  CircuitPlayground.setPixelColor(0, red, green, blue);
  CircuitPlayground.setPixelColor(9, red, green, blue);
}

Done compiling.
C:\Users\reaga\onedrive - ubc\documents\arduino\data\packages\adafruit\tools\arm-none-eabi-gcc\9-2019q4\bin\..\lib\gcc\arm-none-eabi\9.2.1\..\..\..\arm-none-eabi\bin\ld.exe
Sketch uses 38748 bytes (14%) of program storage space. Maximum is 262144 bytes.
```

We will also open up the serial monitor while we wait for it to upload. **Tools->Serial Monitor** or **CTL-SHIFT-M**

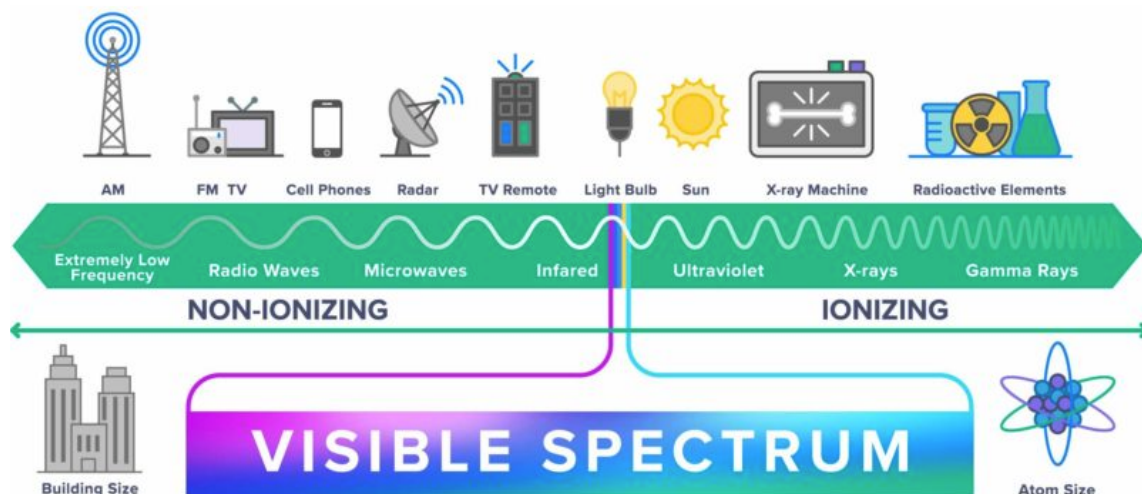
The monitor will be blank until the code has fully uploaded. You will see LEDs all go green and then one blink really fast red, this means the code is being uploaded to the board.

You will then see values being printed to the serial monitor corresponding to the light A5 is receiving. You can change this value by putting your hands over the A5 pin on the board. This will also make the two LEDs flash colors closer to red.

Part 1b. Arduino Sketches - Communication

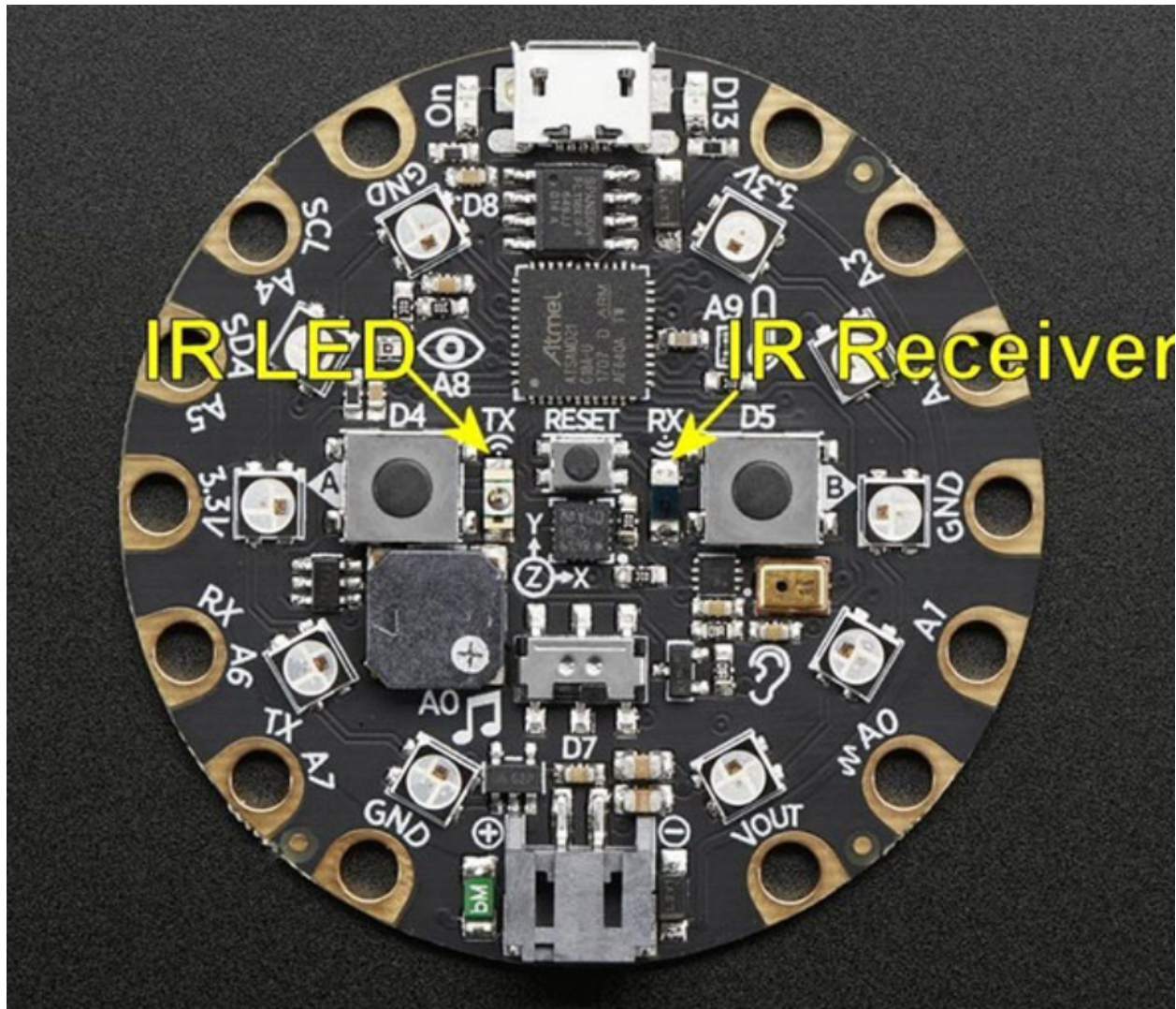
The next part of this lab will focus on communication between two boards. Communication between instruments (e.g., weather sensors and data loggers) can happen in a lot of different ways. The Campbell Scientific sensors use wires. Most of our weather sensors communicate over cellular or WIFI networks to relay data to online servers that we can grab in near real time.

Both cellular signals and WIFI use radio signals to communicate. However, they use much different wavelengths: cell signals can travel much further as their wavelengths are anywhere from 10-1000m, whereas WIFI wavelengths are 6 or 12 cm. This is why WIFI signals may be poor in certain parts of your house/apartment/office. Below is a schematic of many the various different electromagnetic wavelengths and their frequencies.



In this lab we will be using even smaller wavelengths, classified as infrared. Their wavelengths can range anywhere from 780nm to 0.3mm. Thus, they will be able to travel only short distances. This allows the CPE to communicate with other nearby CPEs, and other electronics. You can set your CPE up to be a remote for your TV or a console for a video-game!

- a) Each group will need to partner up with another group. You will then need to decide amongst your paired-group on which board will be the receiver and which board will be the transmitter.
- b) The receiving board will make a sound every time it receives a signal from the transmitting board.



Start with fresh sketches for both parts. For more information on how the infrared signals work on the Circuit Playground Express and the different protocols you can use:

<https://learn.adafruit.com/infrared-transmit-and-receive-on-circuit-playground-express-in-c-plus-plus-2/overview>

Transmitter Code:

Like the sketch we wrote before, we will start by declaring the libraries we need:

```
#include <Adafruit_CircuitPlayground.h>
```

Next, we will define our variables. In IR transmission we need a protocol, a value and the number of bits. The protocol is how the data is transmitted, the value is what is being

transmitted and the bits indicate how much data to send. Here, we will use the most basic protocol and value:

```
uint8_t IR_protocol = NEC;
uint32_t IR_value = 1;
uint16_t IR_bits = 32;
```

We can now declare our void setup. This again includes starting the board we are using and starting the serial monitor. We will also throw in a *while* loop to wait until the serial monitor is fully set up.

```
void setup(){
  CircuitPlayground.begin();
  Serial.begin(9600);
  while (!Serial);
}
```

Lastly, we have our void loop, which will do the heavy lifting. This loop will send the IR signal to the other board. We will have the signal be sent when we press the right button (B). The IR signal will be sent with the protocol, value and bits we declared earlier.

```
void loop(){
  if(CircuitPlayground.rightButton()){
    Serial.println("Right button pressed!");
    Serial.println(IR_protocol);
    CircuitPlayground.irSend.send(IR_protocol, IR_value, IR_bits);
  }
}
```

Receiver Code:

This code is for the group who are receiving the signal. We will start off as we always do, defining the libraries to be used:

```
#include <Adafruit_CircuitPlayground.h>
```

Next, we will create our void setup. Here, we will start out libraries, start the serial, and wait for the serial to start. We will also start the board's receiver so that it is ready to accept the IR signal from the transmitter.

```

void setup() {
  CircuitPlayground.begin();
  Serial.begin(9600);

  while (!Serial); // Wait until serial console is opened

  CircuitPlayground.irReceiver.enableIRIn(); // Start receiver
  Serial.println("Ready to receive IR signals");
}

```

Lastly, we will write the void loop, the heavy lifter of the code. In this loop, the board will receive a signal and decode it into a human-readable format (it is not sent like this, but it is sent in the protocol we declared). To indicate to us that the signal has been sent, we will have it print out into the serial monitor but also have the board play a tone. Here we set up the tone to send a sound of 1760Hz for 1/100 of a second. Finally, we will reset the board so that it can receive another signal after the current one.

```

void loop() {
  //Continue looping until you get a complete signal
  if (CircuitPlayground.irReceiver.getResults()) {
    CircuitPlayground.irDecoder.decode(); // Decode signal
    CircuitPlayground.irDecoder.dumpResults(false); // False gives
less details
    CircuitPlayground.playTone(1760, 100);
    CircuitPlayground.irReceiver.enableIRIn(); // Restart receiver
  }
}

```

Using the boards:

Once the code is written for either the transmitter or receiver, make sure to save your code with a good name. Then *verify* the code using the checkmark in the top left-hand corner of the IDE. Once the code has been confirmed, you can upload it to your board using the horizontal arrow button to the right of the checkmark.

Once both the transmitter and receiver are done. Face the boards towards each other, giving at least a foot of space between them.

Open both serial monitors.

On the transmitter board, press the right button (B). A sound should be heard from the receiving board, and the serial monitor should indicate the protocol, value, and bits sent.

On the transmitter board, it should state that the button has been pressed and also print the protocol being used.

Once you have confirmed that both boards are working properly, see how far away the boards can be while still sending the signal. You can also see what orientations work best and whether or not the boards must be fully facing each other for the signal to work.

Extra Time

If there is extra time in the lab, we invite you to play around with the CPE more!

Suggestions for what you can check out:

- Have the lights indicate how hot the temperature is near the board. You can alter this temperature by manually touching the temperature sensor itself.
- Have the lights indicate how much noise is picked up by the microphone.
- Make the CPE into a computer mouse.
- You can also check out the adafruit site for ideas:

<https://learn.adafruit.com/category/circuit-playground>

*With all coding exercises the internet is your friend!

Assignment

To hand in by the next lab, **each person submits an individual report:**

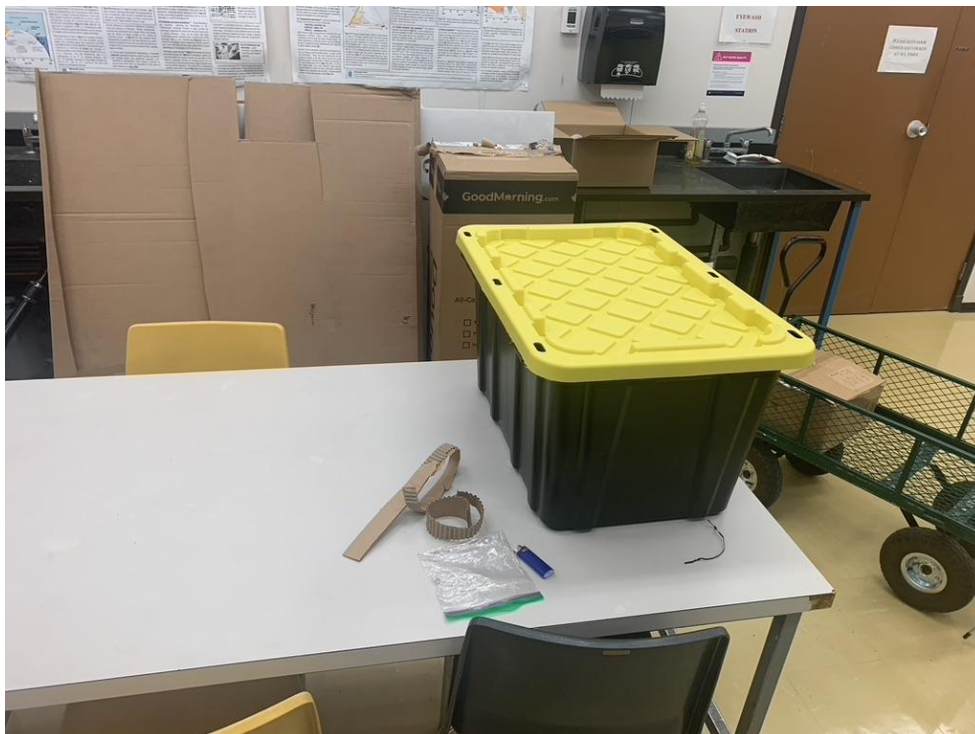
Part 1. Arduino:

1. What was the range of the CPE? Does this make sense, given the spectral frequency we are using? Explain. /2
2. What could infrared sensors be used for in meteorological measurements? /2
3. How would you set up the CPE to measure wind speed? What built-in sensors would you use? /3
4. Given that these boards are only able to measure time as cumulating milliseconds, how would you record time in a DateTime step using code? /3

Part 2. Calibration:

We will also be testing our calibration skills in this lab using PM2.5 data we collected with our [adafruit PM sensors](#) (UBC-4). We will be calibrating our sensors against a [RAMP AQ sensor](#). To conduct the calibration, the PM sensors were placed in a sealed container with the RAMP (see below). Then a piece of botanicals was lit on fire and the lid was closed. We then let the smoke disperse across the container for several hours. Data was collected throughout and will be provided to you to finish off the calibration process. The aim here is to see how well a cheaper sensor performs compared to a much more [expensive pre-packaged sensor suite](#). (Note that both the RAMP and PM sensors use the [Plantower PM2.5 sensors](#), it is the packaging and data relay that are different). Information on this calibration process can be found on our PM sensor web page under *Static Test*.

Pictures of setup



1. Create a graph of the RAMP and PM sensor data (y-axis) through time (x-axis) over the duration of the experiment. *(See provided csv file)* /5
2. Does the distance between the two curves change over time? If so, when are the closest? When are they furthest away? Why do you think this is? /3

3. Create a linear regression model for the RAMP and PM sensor data. What is the correction function? (*You can do this Excel or Python or your favorite language*) /5
4. Explain in your own words what the linear regression did to the data and why this is important. /3
5. Create a new time series plot (RAMP-UBC PM 2.5 vs Time) but this time using the corrected PM sensor data with your function above. /5
6. How effective was the function in correcting the PM data? How does the distance between the two curves differ over time now? /3
7. Is this a thorough calibration? If not, what could be done better? /4

End of lab questions.

Comments for the TAs?

-you can submit these earlier, by email, to improve the next lab.