Circuit Playground Class Scheduler

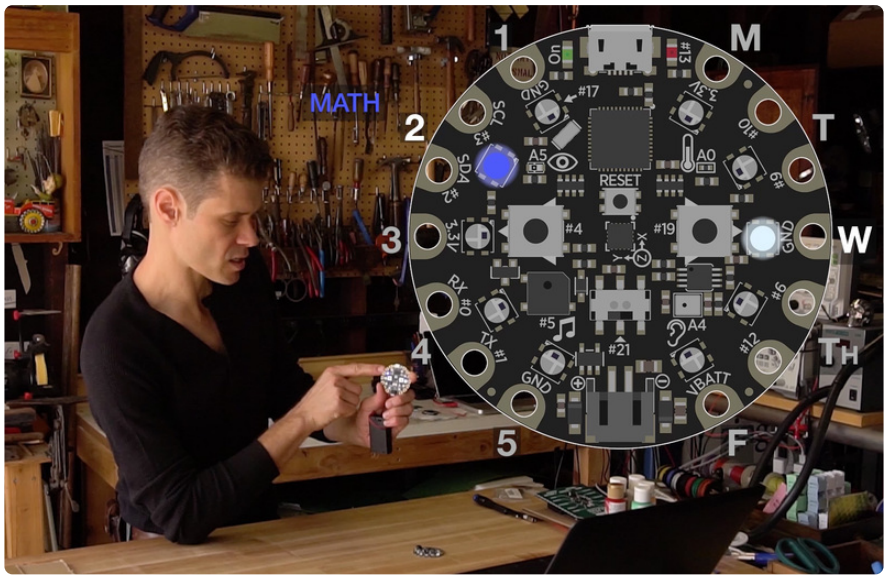
Created by John Park

https://learn.adafruit.com/circuit-playground-class-scheduler

Last updated on 2024-06-03 01:58:47 PM EDT

Table of Contents

Overview



The Problem: a Tricky Schedule

My son's middle school class schedule is kind of complex. In fact, it sounds kind of like the beginning of a word problem: His math class occurs first period on Monday and Tuesday, but is at third period on Wednesday, and fifth period on Thursday and Friday...

In order for him to keep track of which class happens when on each day, we decided to create an electronic schedule reminder using Circuit Playground!

The Solution: the Class Scheduler

We can program the schedule into the Circuit Playground and then use it as a sort of cheat sheet to quickly check which class is happening at which period on a given day.

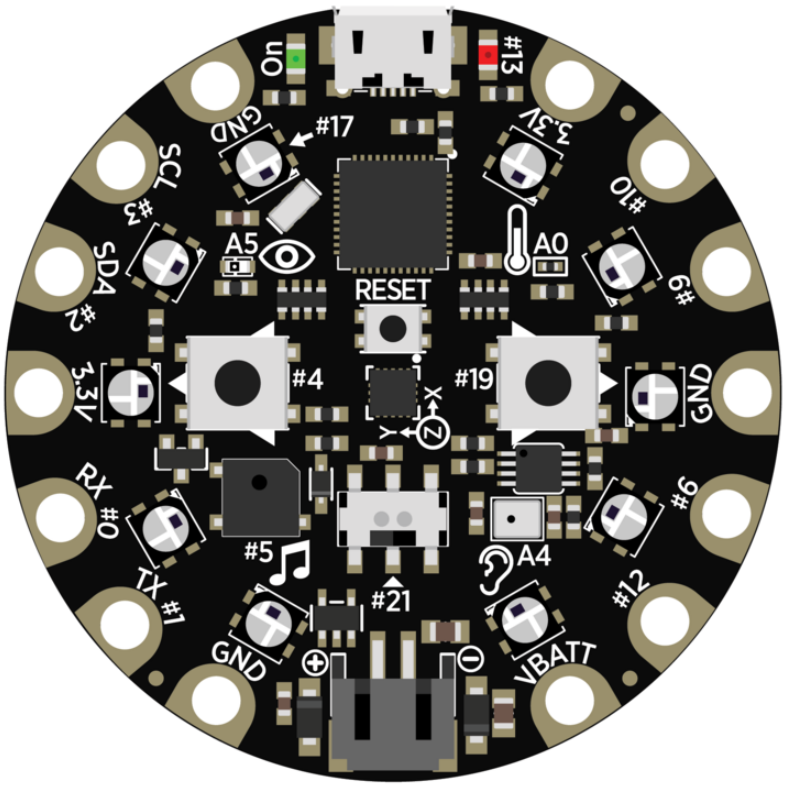
Instead of having the schedule written out as a bunch of text on a paper schedule or a digital display, the ten colored NeoPixels on the Circuit Playground can convey a lot of information very efficiently. We use the five NeoPixel LEDs on the right half of the board to indicate the day of the week, and the five on the left to indicate class periods one through five. The class period NeoPixels are color coded to the same color folder he uses for that class, so, green = science, for example.

We also built a pretty cool mounting panel with informational call-outs for the Circuit Playground that can be attached to his backpack with magnets, but this is optional. You can skip it entirely, or come up with your own solution.

Follow along as this guide will show you how to program and build your own Circuit Playground Class Scheduler!

Prep the Circuit Playground

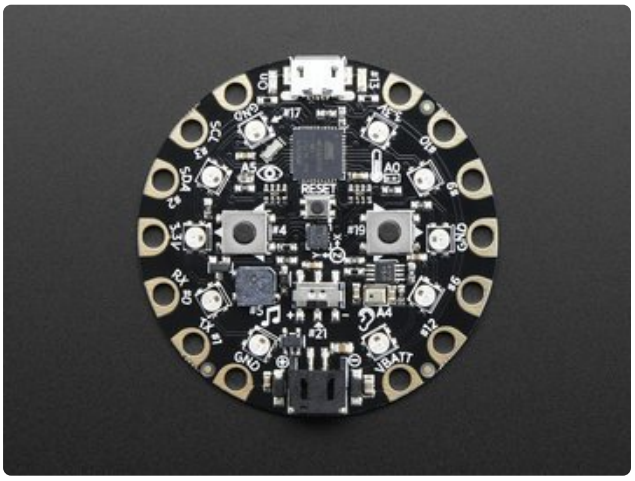
Before you get started, make sure you've been through the basic tutorials on using the Circuit Playground. You'll need to have installed the Arduino IDE, added the Circuit Playground board to Arduino, and added the Circuit Playground library. This excellent guide, Introducing Circuit Playground (https://adafru.it/pAP) and Circuit Playground lesson #0 (https://adafru.it/r0D) will show you how to do all of that and get you going in no time!



Plug your Circuit Playground into your computer now, and launch the Arduino IDE. Double check that you've selected the board and port as in the Lesson #0 tutorial and are ready to upload code.

Required Parts

To make your own Circuit Playground Class Scheduler, you'll need the following:



A **Circuit Playground**!

Available at Adafruit (http://adafru.it/3000)

Micro USB Cable any length. You probably have one of these around, they're the most common USB cable.

Make sure its a data/sync cable!

USB cable available at Adafruit (http://adafru.it/592)

A HUUUUUUGE number of people have problems because they pick a 'charge only' USB cable rather than a "Data/Sync" cable. Make 100% sure you have a good quality syncing cable. Srsly, I can't even express how many times students have nearly given up due to a flakey USB cable!

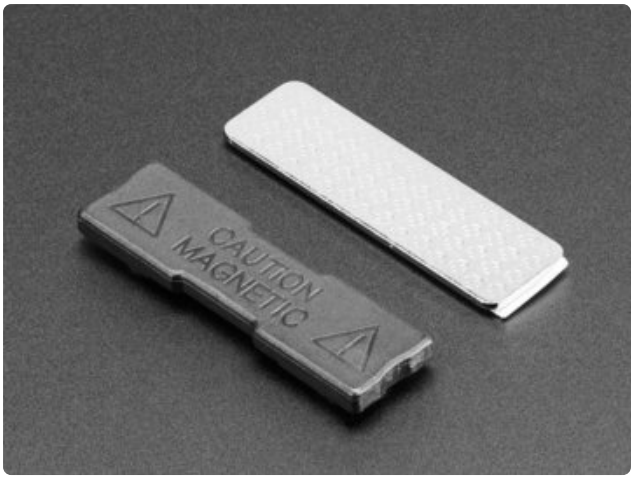
Battery Holder with power switch

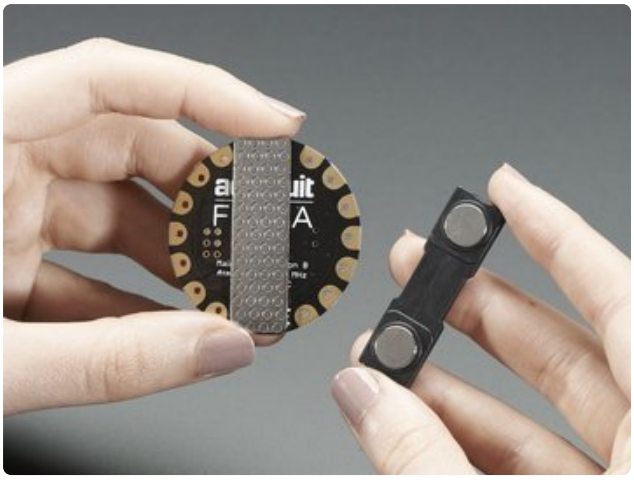
3 x AAA Battery Holder with On/Off Switch and 2-Pin JST (http://adafru.it/727)

Optional Parts

If you'd like to create the same enclosure we made, you'll need some additional parts. You can also use this as a leaping off point for designing your own enclosure or attachment method! Maybe you'll affix yours to your vintage, horse-image Trapper Keeper, or sew it to your hoodie?

- Acrylic craft paints in five different colors
- Small paint brush or cotton swabs



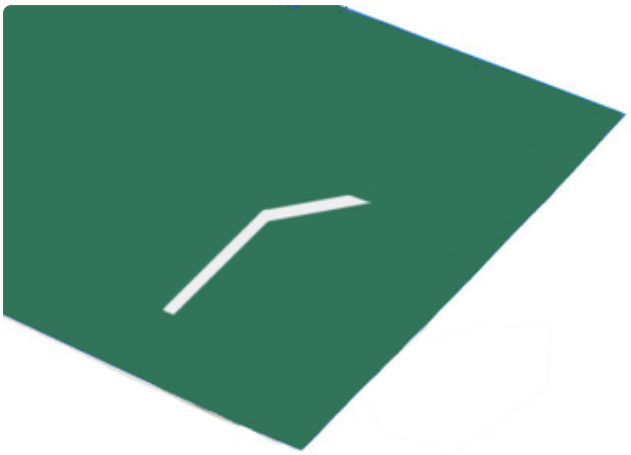


2x Magnetic Pin Backs (http://adafru.it/1170)

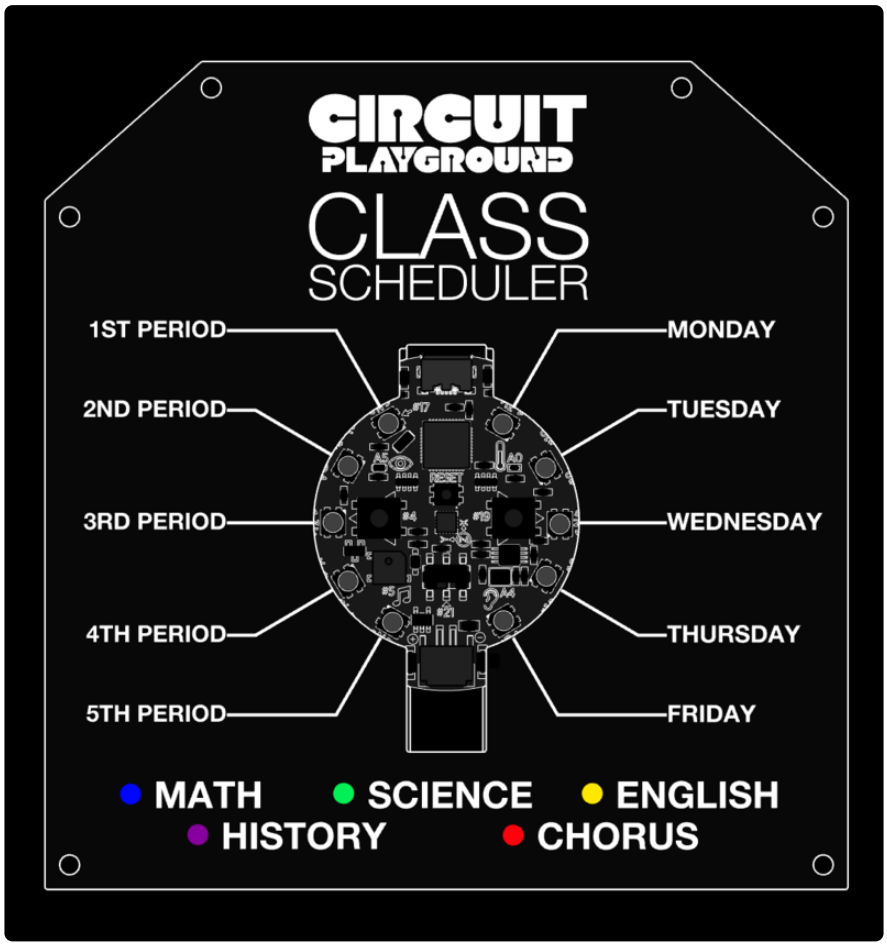
3 sheets 8"x10" two-color acrylic sheet (https://adafru.it/r0E) .022" thickness

6x M3 (3mm diameter) 6mm long cap head machine screws (https://adafru.it/r0F)

6x Hex nuts for M3 screws (https://adafru.it/q1a)



Scheduler Code



Class Scheduler Logic

The sketch that runs on the Circuit Playground Class Scheduler is fairly simple. It really only can do four things:

- Beep and blink a red LED when it's turned on
- Change the day of the week indicator NeoPixels when the right button is pressed
- Change the class period indicator NeoPixels when the left button is pressed
- Show or hide the display/lock the buttons when the toggle switch is flipped (this way you don't have to turn it off and reset everything, but can still save power or avoid annoying people with the bright lights)

Pseudo-code

Before you look at the code below, here's a pseudo-code version of it. "Pseudo-code" is a way of thinking about the logic of your code before diving into the details of proper syntax. Here are the steps the software will go through when it runs:

1. Beep and blink when turned on or reset
2. Check the slide switch to see if it is "on" or "off"
3. If the slide switch is "off", do nothing other than continue checking the slide switch
4. Otherwise, if the slide switch is "on", light up the Monday NeoPixel to white, and first period NeoPixel to the proper class subject color according to the schedule
5. Check the right button. If it's pressed, advance to the next day, adjust the day NeoPixel to the proper day, and adjust the current period NeoPixel to the proper subject color according to the schedule
6. Check the left button. If it's pressed, advance to the next period, adjust the NeoPixel to the proper period and color according to the schedule
7. Repeat

Variables

Variables are a bit like names we can use to refer to values that may change depending on what's going on. For example, we can tell the software to turn on the `day` NeoPixel, where `day` is a variable that stands for then number of the day we've currently selected by pushing the button.

When you think about implementing the above logic as actual code in Arduino, there are a few things that stand out is needing to be expressed as variables:

- Day of the week
- Class period
- Class subject
- Schedule of class subjects per period per day
- Colors correlated to each subject

You can see examples of these variables in the code snippet below. The `schedule` and `classColor` variables are both a special type called a 2D matrix, which just means they have rows and columns like a spreadsheet. This makes it efficient later in the code for a function to ask for, say, the green value of the english class by querying `classColor[4][1]`

The positions of the values in the row and column arrays are indexed to begin at 0, so index 4 is actually the fifth row and so on.

```
      // NUM_CLASSES is the number of class periods in a day
// You could adjust this -- may be fewer than 5 but not greater
// without running into the day-of-the-week pixels -- and you'd also
// need to adjust the "schedule" and "classColor" matrices below

const int NUM_CLASSES = 5;

//*********************************************************
// Adjust this "classColor" matrix to pick your own colors expressed
// as RGB value triplets from 0-255 per class subject
//*********************************************************

int classColor[NUM_CLASSES][3] = {
    {255,  0,  0}, // chorus    is red
    {  0,255,  0}, // science is green
    {  255,0,255}, // history is purple
    {  0,  0,255}, // math  is blue
    {255,255,  0}  // english is yellow
};

//*********************************************************
//Adjust this "schedule" matrix to reflect your own class schedule
//*********************************************************

const int NUM_DAYS = 5;
int schedule[NUM_DAYS][NUM_CLASSES] = {
    {0,1,2,3,4}, // Monday    = chorus, science, history, math, english
    {0,4,1,2,3}, // Tuesday   = chorus, english, science, history, math
    {0,3,4,1,2}, // Wednesday = chorus, math, english, science, history
    {0,2,3,4,1}, // Thursday  = chorus, history, math, english, science
    {0,1,2,3,4}  // Friday    = chorus, science, history, math, english
};

// Global variables that track the current day and current period.

int period = 0;
int day = 0;
```

Anything that follows two slashes // is considered a comment for human use only and won't be read by the Arduino software!

Additionally, there are variables we'll use to store the current state and previous state of the buttons. Each of these variables can be set to a value of "pressed" or "not pressed". By comparing the current and previous state of a these variables, the program knows when a button has been pressed and can then run a particular piece of code.

```
bool leftFirst = CircuitPlayground.leftButton();
  bool rightFirst = CircuitPlayground.rightButton();
  delay(20);
  bool leftSecond = CircuitPlayground.leftButton();
  bool rightSecond = CircuitPlayground.rightButton();
```

Functions

You'll use a number of functions that are built into the Circuit Playground library. Functions are essentially little (or not so little!) pre-built routines that you can use to take care of a whole bunch of steps of processing for you. A good example is the `CircuitPlayground.clearPixels()` function. When you call this function a whole bunch of steps are run that go through and turn off each NeoPixel one by one, so you can imagine your sketch would be much longer and more complicated if you needed to do that step by step.

Here are some of the other key functions you'll use:

- `CircuitPlayground.begin()` initializes the board
- `CircuitPlayground.playTone()` makes sound
- `CircuitPlayground.redLED()` turns the red LED on and off
- `CircuitPlayground.slideSwitch()` checks the slide switch for input
- `CircuitPlayground.leftButton()` checks the left button for input
- `CircuitPlayground.rightButton()` checks the right button for input
- `CircuitPlayground.setPixelColor()` lights up a particular NeoPixel to a specified color value

Arguments

Some functions require additional input when they are called. These clarifications are sometimes called arguments. For example, `CircuitPlayground.redLED(true)` will turn the red LED on, and `CircuitPlayground.redLED(false)` will turn it off.

There are also functions that require multiple arguments, such as turning on a NeoPixel, where you must specify which one to address, and the individual red, green, and blue values. Here's an example of turning NeoPixel **#0** red:

```
CircuitPlayground.setPixelColor(0, 255, 0, 0)
```

While this set of arguements to the function will turn NeoPixel **#9** purple:

```
CircuitPlayground.setPixelColor(9, 255, 0, 255)
```

The Code

The code below is the full script.

```
// ------------------------------------------------------------------------
//Circuit Playground Class Scheduler
// John Park
// for Adafruit Industries
//
//
//Uses NeoPixels 0-4 to indicate class schedule by
//period (position) and subject (color),
//for a give day of the week indicated by NeoPixels 5-9.
//
// For use with the Adafruit Circuit Playground.
//
// MIT license, check LICENSE for more information
//
// ------------------------------------------------------------------------
//Setup
//Set schedule (subject per period per day of the week)
//variables at top of sketch in the schedule matrix.
//Upload the sketch to Circuit Playground.
//
//Usage
//Flip the toggle switch to the right to show display, left to hide it
//Press right momentary button to change day of the week
//Press left momentary button to change class period
// ------------------------------------------------------------------------

#include &lt;Adafruit_CircuitPlayground.h&gt;
#include &lt;Wire.h&gt;
#include &lt;SPI.h&gt;

// NUM_CLASSES is the number of class periods in a day
// You could adjust this -- may be fewer than 5 but not greater
// without running into the day-of-the-week pixels -- and you'd also
// need to adjust the "schedule" and "classColor" matrices below

const int NUM_CLASSES = 5;

//*********************************************************
// Adjust this "classColor" matrix to pick your own colors expressed
// as RGB value triplets from 0-255 per class subject
//*********************************************************

int classColor[NUM_CLASSES][3] = {
    {255,  0,  0}, // chorus    is red
    {  0,255,  0}, // science is green
    {  255,0,255}, // history is purple
    {  0,  0,255}, // math  is blue
    {255,255,  0}  // english is yellow
};

//*********************************************************
//Adjust this "schedule" matrix to reflect your own class schedule
//*********************************************************

const int NUM_DAYS = 5;
int schedule[NUM_DAYS][NUM_CLASSES] = {
    {0,1,2,3,4}, // Monday    = chorus, science, history, math, english
    {0,4,1,2,3}, // Tuesday   = chorus, english, science, history, math
    {0,3,4,1,2}, // Wednesday = chorus, math, english, science, history
    {0,2,3,4,1}, // Thursday  = chorus, history, math, english, science
    {0,1,2,3,4}  // Friday    = chorus, science, history, math, english
```

```
};

// Global variables that track the current day and current period.

int period = 0;
int day = 0;



void setup() {
    CircuitPlayground.begin(); //initialize the board
    CircuitPlayground.playTone(932,200); //startup beep frequency and duration
    //pulse the red LED twice
    int i=0;
    for(i=0;i&lt;2;i++){
        CircuitPlayground.redLED(true);
        delay(200);
        CircuitPlayground.redLED(false);
        delay(200);
    }
    //fairly dim NeoPixel brightness setting -- goes up to a blinding 255!
    CircuitPlayground.setBrightness(15);
    CircuitPlayground.clearPixels(); //turns off all NeoPixels
}


void loop() {

    //check the slide switch, "off" to the right means the buttons
    //won't respond, acting like they are locked, and turning off the
    //NeoPixels to save power/avoid annoying people, "on" to the left

    if (!CircuitPlayground.slideSwitch()) {
        CircuitPlayground.clearPixels();
        return;
    }

    // -------------------------------------------------------------------------
    // Detect if the left or right button is released by taking two
    // readings with a small delay in between.  If the button changes
    // state from pressed -&gt; released then we can update indicators.

    bool leftFirst = CircuitPlayground.leftButton();
    bool rightFirst = CircuitPlayground.rightButton();
    delay(10);
    bool leftSecond = CircuitPlayground.leftButton();
    bool rightSecond = CircuitPlayground.rightButton();

    // Read the day of the week (right) button, if pressed, increment
    // the "day" variable
    //"day" is an integer to represent the day of week 0-4 / M-F
    //starts on Monday which is index 0

    if (rightFirst &amp;&amp; !rightSecond){
        day++;
        //cycles from friday back to monday
        if (day &gt; NUM_DAYS - 1) {
            day=0;
        }
    }

    // Read the class period (left) button, if pressed, increment the
    // "period" variable

    // "period" is the class period counter to indicate current period
    // starts on 1st period, which is index 0

    if (leftFirst &amp;&amp; !leftSecond){
        period++;
```

```
        //cycles from first to fifth period
        if (period &gt; NUM_CLASSES -1) {
            period=0;
        }
    }

    // ------------------------------------------------------------------------
    // Now we're ready to update display with day/period.

    // Start by turning everything off

    CircuitPlayground.clearPixels();

    // Light up the day of the week pixel

    CircuitPlayground.setPixelColor((9-day), 255, 255, 255);

    // Light up the proper color for the class subject

    CircuitPlayground.setPixelColor(period,
                                    (classColor[(schedule[day][period])][0]),
                                    (classColor[(schedule[day][period])][1]) ,
                                    (classColor[(schedule[day][period])][2])) ;
}
```

Download the Code

You can press the button below to download the code. Unzip and move the directory CircuitPlaygroundClassScheduler to your Arduino sketch directory.

CircuitPlaygroundClassScheduler.zip

https://adafru.it/q6c

Edit the Code

Open the ClassScheduler.ino file in the Arduino IDE. You can customize the `schedule[][]` and `classColor[][]` matrices to match your own class schedule.

For example, if your schedule is:

- Monday - English, Cooking, Math, Italian, Science
- Tuesday - English, Cooking, Math, Italian, Science
- Tuesday - English, Cooking, Math, Italian, Science
- Tuesday - Science, Cooking, Math, English, Italian
- Tuesday - Science, Cooking, Math, English, Italian

The matrix would look like this:

```
int schedule[NUM_DAYS][NUM_CLASSES] = {
  {0,1,2,3,4}, // Monday    =  English, Cooking, Math, Italian, Science
```

```
  {0,1,2,3,4}, // Tuesday   =  English, Cooking, Math, Italian, Science
  {0,1,2,3,4}, // Wednesday =  English, Cooking, Math, Italian, Science
  {4,1,2,0,3}, // Thursday  = Science, Cooking, Math, English, Italian
  {4,1,2,0,3}  // Friday    = Science, Cooking, Math, English, Italian
};
```

You can do the same type of editing if you'd like to chose different colors to correspond to your class subjects. For example:

- English is green
- Cooking is red
- Math is white
- Italian is blue
- Science is purple

Would look like this:

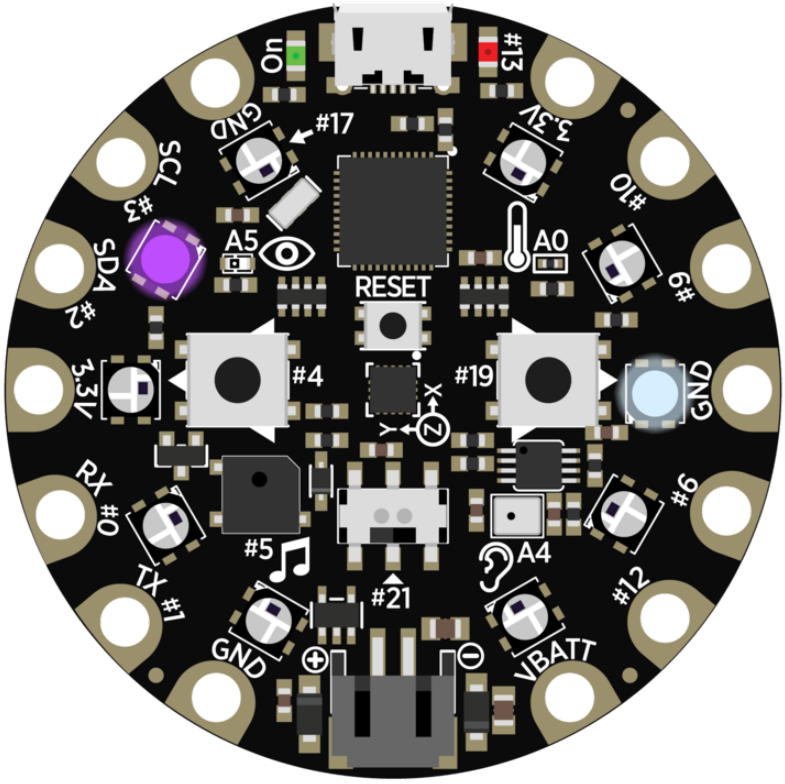
```
int classColor[NUM_DAYS][NUM_CLASSES] = {
  {0, 255, 0},
  {255, 0, 0},
  {255, 255, 255},
  {0, 0, 255},
  {255, 0, 255}
};
```

Save It

Once you've edited the code, save it. In the next section we'll upload it to the board.

Here's a great suggestion from John K Shaffstall aka NorthernPike: "Great idea for using a Playground. And the panel is a perfect way to define its use. I have a suggestion about the code. It has to do with the "day" and "period" variables. I find that variables used as what I call "cyclers" that wrap around are good candidates for the conditional operator. As in... if (rightFirst && !rightSecond){ day = day > (NUM_DAYS - 1) ? 0 : day + 1; //cycles from friday back to monday } Instead of a nested if as in... if (rightFirst && !rightSecond){ day++; //cycles from friday back to monday if (day > NUM_DAYS - 1) { day=0; } } Just a thought... The "Conditional" operator is too overlooked and under utilized in my opinion. MAKE a nice day..."

Run the Code



Upload Code to the Board

Plug your Circuit Playground into your computer with the USB cable.

In the **Arduino IDE**, select the **Adafruit Circuit Playground** item in the menu Tools > Board.

Then, select the proper port in **Tools** > **Ports**.

Press the reset button twice on the Circuit Playground so that the red **#13** LED begins pulsing, indicating it is ready to be programmed.

In the **Arduino IDE**, click **Sketch** > **Upload**.

The board will blink, and after a few second the new code will be uploaded to the board.

The board will blink the red **#13** LED twice and beep when the sketch starts.

Test it

By default, the program starts on Monday, first period, so the Circuit Playground will have NeoPixels **#0** and **#9** lit. If these aren't lit, flip the slide switch to the left and they'll light up.

Press the **right button** to cycle the days. Each time you press and release the button, the day will advance, and then loop to the beginning. Time flies!

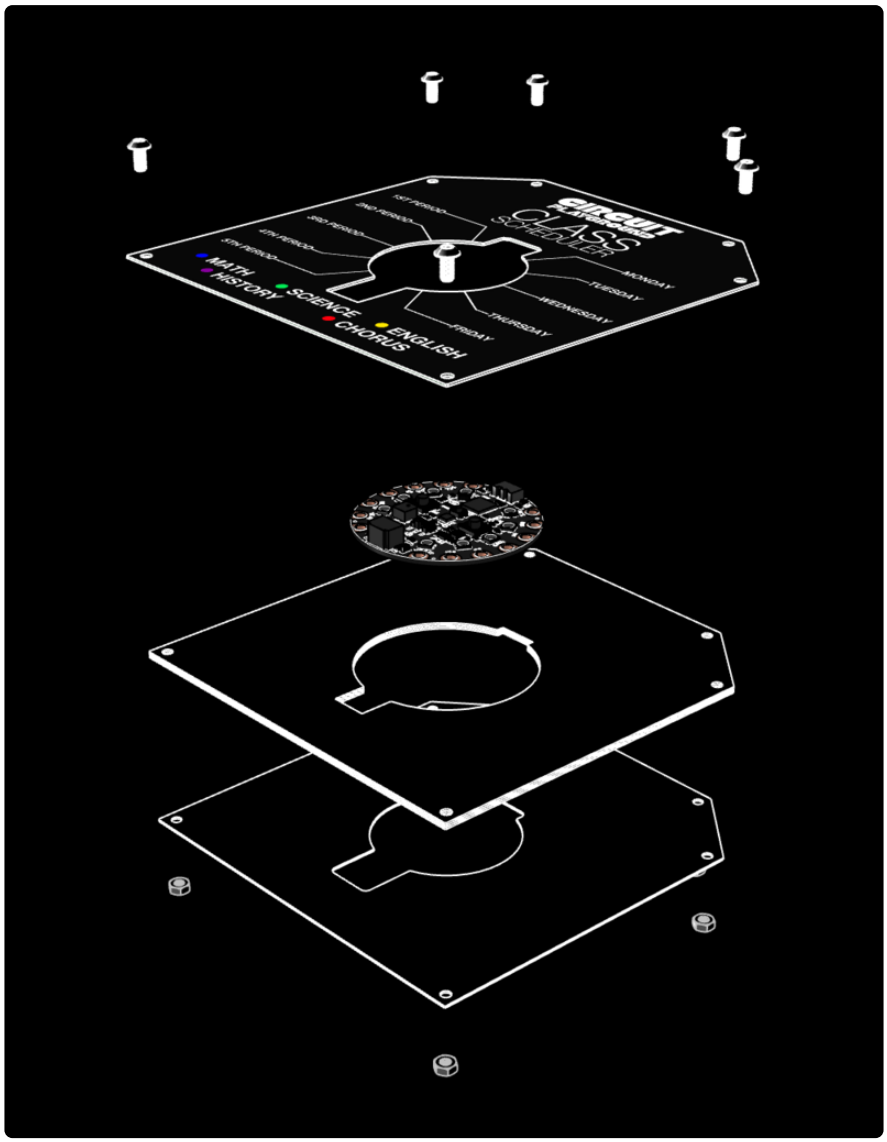
Also note that as the day advances the first period NeoPixel **#0** will update its color to correspond to your schedule (the color won't change if your schedule has the same class every morning).

Now, try pressing the **left button** a few times. You'll see the the current period indicator advances, and updates to the appropriate color.

Time to untether! You can now remove the USB cable from the Circuit Playground and power it from the battery holder. Insert three AAA batteries in the case, plug it in to the **JST** connector on the Circuit Playground, and turn the battery holder switch to **ON**.

You're ready to use it on the go at school! If you like, you can simply carry it in a pocket, or affix it to your backpack or a jacket with a magnetic pin. Or, you can move on to the next section and build a sweet mounting panel for your Circuit Playground Class Scheduler.

Build the Mounting Panel



Layers

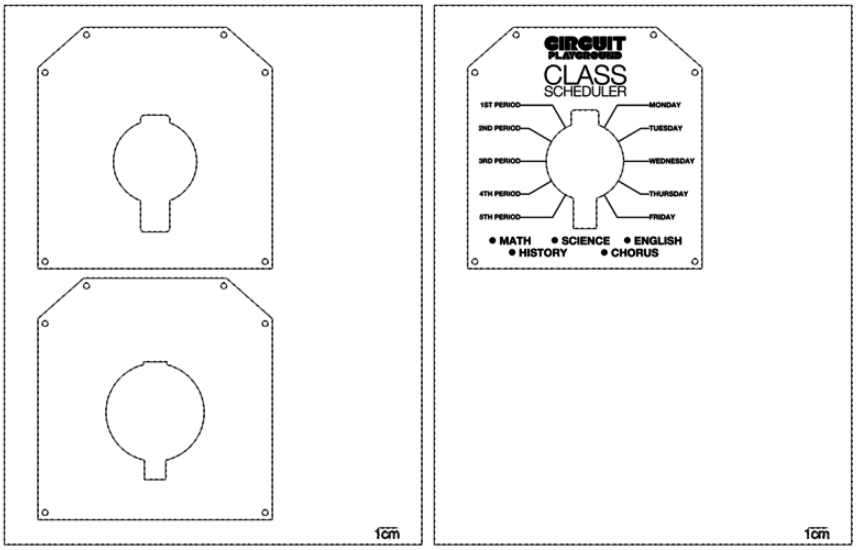
The Circuit Playground will be sandwiched between layers of acrylic which overlap the board edge on top an bottom, leaving most of the board clear for interaction with the buttons and NeoPixel, and allowing access to the **JST** connector for plugging in the battery.

Since the USB port isn't available when the board is encased in the mounting panel, you'll need to be sure you're happy with the program before assembling the case.

Using the **CPClassScheduler_CAD** files linked below, cut the five layers of acrylic on a laser cutter (or send the job out to service such as Ponoko), including the etching pass on the top layer.

CPClassScheduler_CAD.zip

https://adafru.it/q6e



If you don't have access to a laser cutter you can send the job out to a service such as Ponoko. This .eps file works well for Ponoko's requirements.

CPClassScheduler_files05.eps

https://adafru.it/AQ5

Or print out templates and cut the acrylic on a CNC mill, band saw, jig saw, or coping saw. Or, you can skip the acrylic and cut the mounting panel from heavy cardstock or thin cardboard.

Paint it

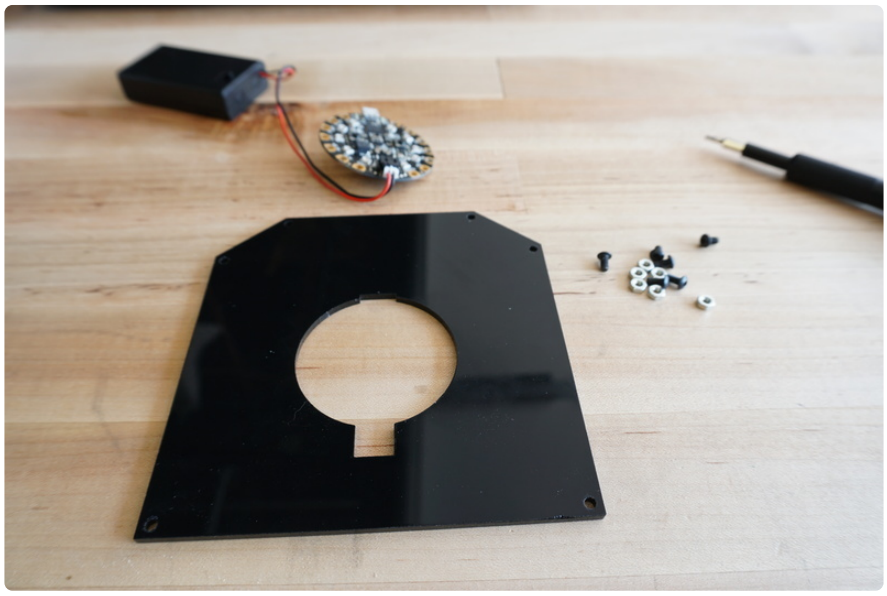
To color code the class subjects, use a small brush or cotton swab to dab a bit of the corresponingly colored acrylic paint onto the circle to the left of each subject name. For example, a blue dot next to "Italian" if you're studying Italian. Bellissimo!

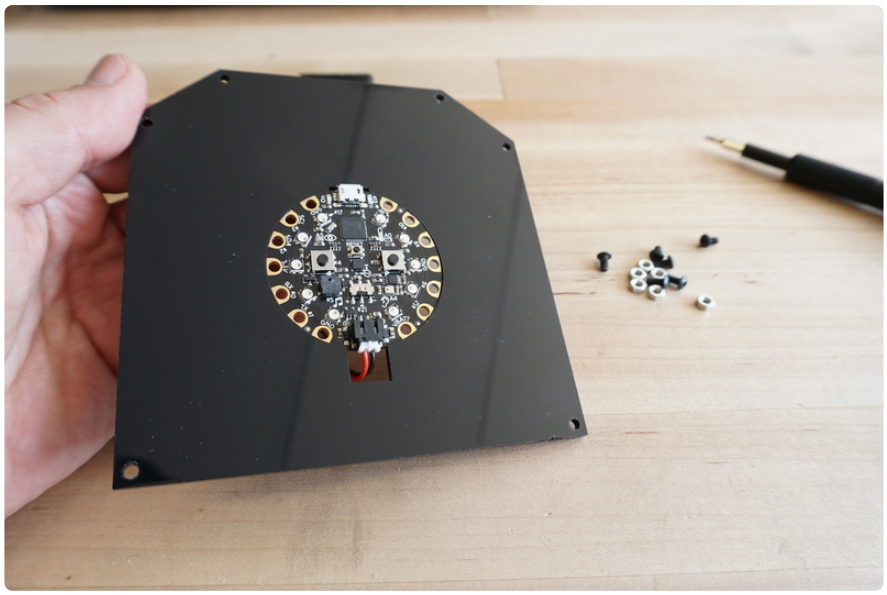
Assembly

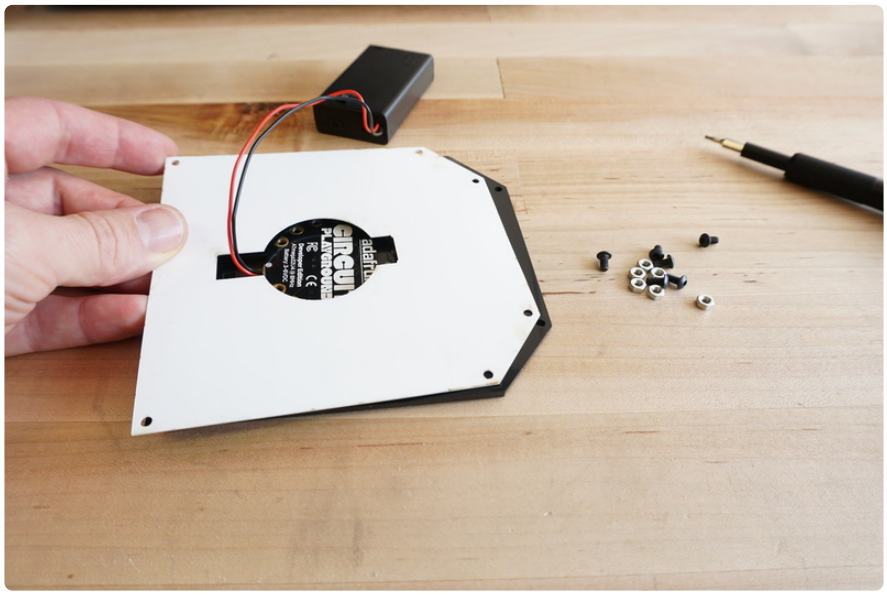
To assemble, lay down the bottom layer, followed by the middle three layers. Then, press the board into its space. Place the top layer on top of the board, then screw in the six screws from the top, affixing the hex nuts from the bottom side. You can use a bit of thread locker, such as Loctite, or even nail polish (clear or maybe a fun color!) on the threads to keep the nuts from unscrewing on their own.

Plug in the battery pack and test that it is working.

Next, peel off the foam tape and attach two magnetic pin backs to the back of the mounting panel. You can use these to pin the Class Scheduler to your backpack, jacket, folder, etc.

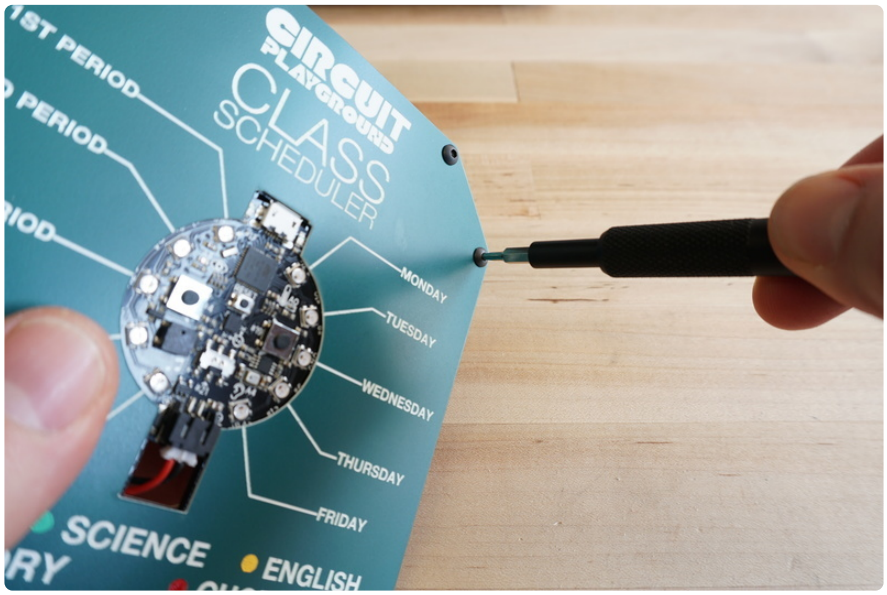




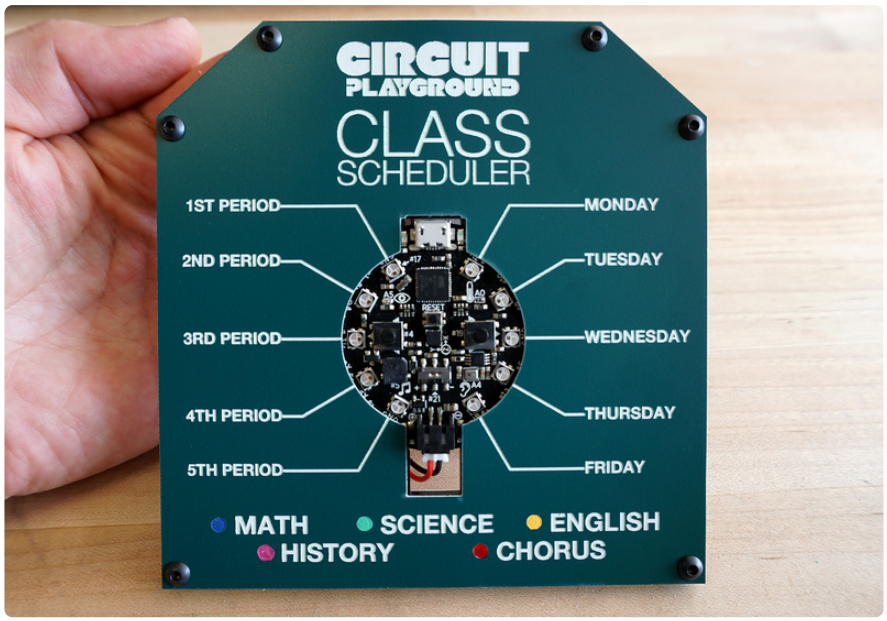



CIRCUIT
PLAYGROUND
CLASS
SCHEDULER
1ST PERIOD
2ND PERIOD
3RD PERIOD
4TH PERIOD
5TH PERIOD
MONDAY
TUESDAY
WEDNESDAY
THURSDAY
FRIDAY
MATH
SCIENCE
ENGLISH
HISTORY
CHORUS


adafruit
CIRCUIT
PLAYGROUND
Developer Edition


CIRCUIT
PLAYGROUND
CLASS
SCHEDULER
MONDAY
TUESDAY
WEDNESDAY
THURSDAY
FRIDAY
SCIENCE
ENGLISH

Use the Class Scheduler



Time to attatch the Class Scheduler to your backpack or notebook and put it to use! Now, you can check the class schedule any time you like. Be sure to put it to sleep with the slider switch when not in use, or turn it off using the battery pack ON/OFF switch if you aren't using it for longer periods of time.

Now get to class!