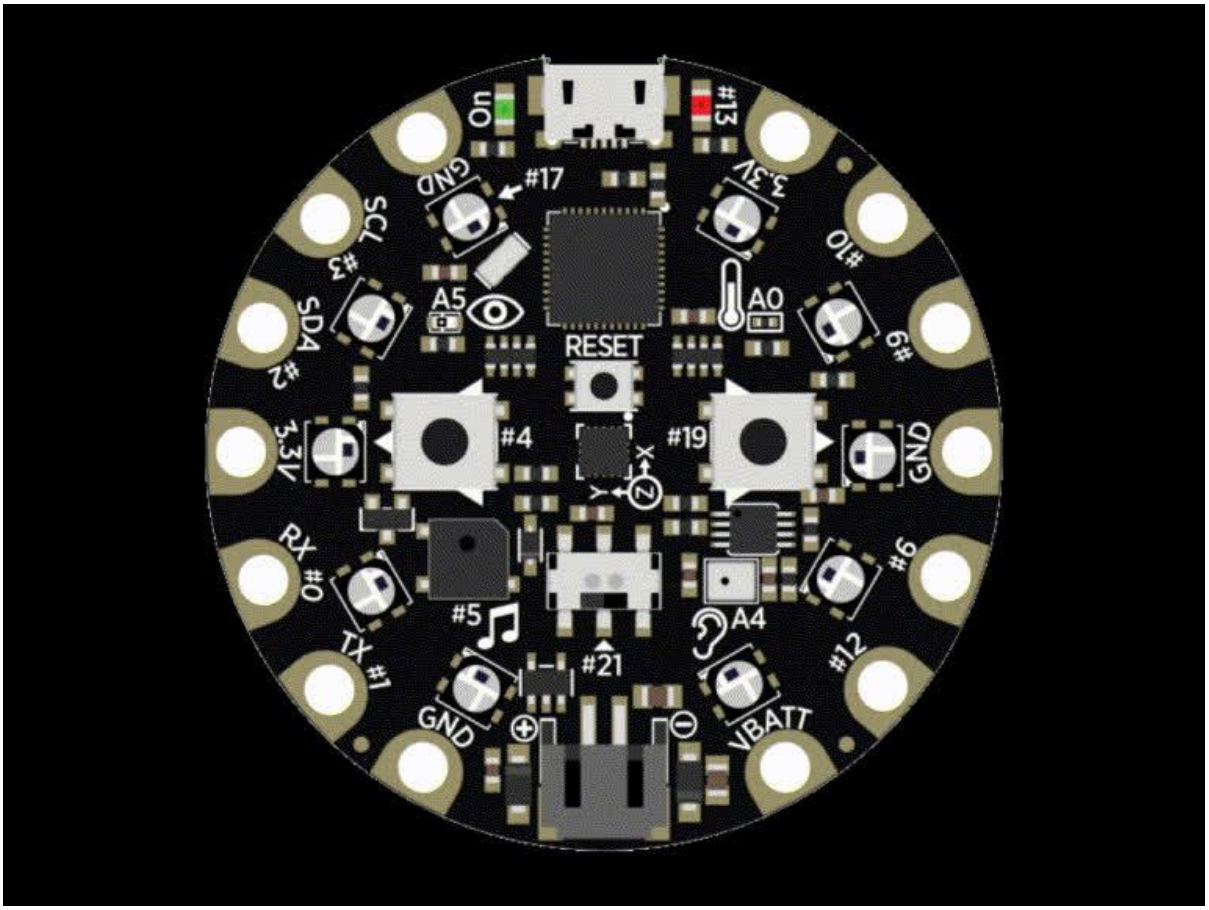
# Circuit Playground Beep Beep

Created by Carter Nelson




https://learn.adafruit.com/circuit-playground-beep-beep

Last updated on 2023-08-29 03:17:29 PM EDT

# Table of Contents

# Overview

In the original Star Trek series, there are a couple of episodes () that feature a severely disabled Captain Pike (former captain of the Enterprise) who is immobilized and can not speak. He can only communicate via a little light on the front of his wheel chair. It blinks (and beeps) once for "yes", and twice for "no".

We can create a similar yes/no functionality using our Circuit Playground. We can use the buttons to select "yes" or "no". The speaker can provide the characteristic monotonic "beep", and the NeoPixels can act like the light. In fact, the resulting program is pretty simple. However, it serves as a good way to introduce a fairly advanced coding topic - refactoring ().

We will start with a fully functional program and then incrementally go through it and show how it can be "improved" by applying various refactoring techniques.

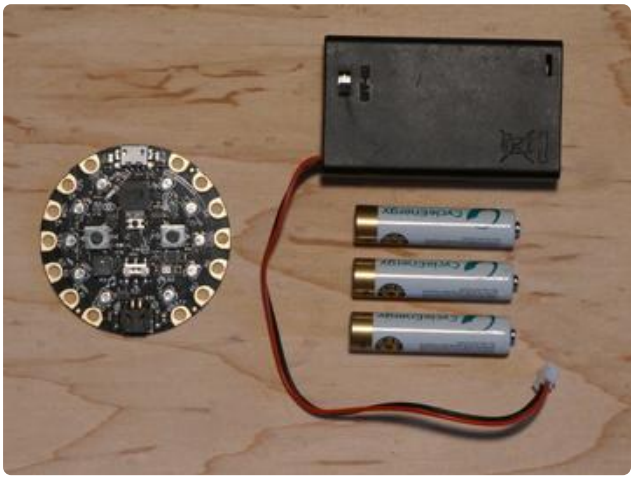
Ready to get started?

BEEP!

Good, let's go.

# Required Parts

This project uses the sensors already included on the Circuit Playground, either a Classic (http://adafru.it/3000) or an Express (http://adafru.it/3333). The only additional items needed are batteries for power and a holder for the batteries.



Circuit Playground
Classic (http://adafru.it/3000)
Express (http://adafru.it/3333)
3 x AAA Battery Holder (http://adafru.it/727)
3 x AAA Batteries (NiMH work great!)

# Before Starting

If you are new to the Circuit Playground, you may want to first read these overview guides.

## Circuit Playground Classic

- Overview ()
- Lesson #0 ()

## Circuit Playground Express

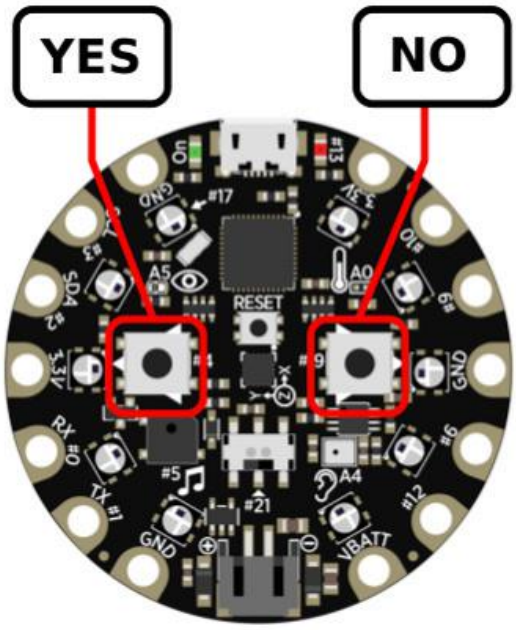
- Overview ()

---

# Starting Point

We will start with a fully functional program. In fact, the functionality will not change throughout this guide.

First, let's assign functions to the two push buttons. We'll use the left button to indicate YES and the right button indicate NO as shown in the figure below.




And here is our starting version of the code.

```
////////////////////////////////////////////////////////////////////////////
// Circuit Playground Yes No v0
//
// One beep (left button) = Yes
// Two beeps (right button) = No
//
// Author: Carter Nelson
// MIT License (https://opensource.org/licenses/MIT)

#include &lt;Adafruit_CircuitPlayground.h&gt;

////////////////////////////////////////////////////////////////////////////
void setup() {
  CircuitPlayground.begin();
}

////////////////////////////////////////////////////////////////////////////
void loop() {
  if (CircuitPlayground.leftButton()) {
    //
    // YES
    //
    for (int p=0; p&lt;10; p++) {
      CircuitPlayground.setPixelColor(p, 0xFF6600);
    }
    CircuitPlayground.playTone(700, 750);
    CircuitPlayground.clearPixels();
  }
  if (CircuitPlayground.rightButton()) {
    //
    // NO
    //
    for (int p=0; p&lt;10; p++) {
      CircuitPlayground.setPixelColor(p, 0xFF6600);
    }
    CircuitPlayground.playTone(700, 500);
    CircuitPlayground.clearPixels();
    delay(250);
    for (int p=0; p&lt;10; p++) {
      CircuitPlayground.setPixelColor(p, 0xFF6600);
    }
    CircuitPlayground.playTone(700, 500);
    CircuitPlayground.clearPixels();
  }
}
```

With this program loaded and running on the Circuit Playground, you should be able to press the buttons to create the Yes/No beeps. The NeoPixels also light up.

---

# What is Refactoring?

Well, to paraphrase the first line of the wikipedia () entry:

> Code refactoring is the process of restructuring existing computer code...without changing its external behaviour.

So basically, it's code clean up. But it's also more. Read the wikipedia article, and other sources, for the gory details. We are going to just hit some main highlights in this guide.

# Why Refactor?

Good question. It is something you will come to appreciate with time. However, the main motivators are:

- Readability
- Maintainability
- Extensibility

The best way to gain an appreciation for this is to see it in practice. This guide will hopefully help start you on that journey.

---

# Refactor 1

Let's start with some low hanging fruit. Take a look at the main loop of the initial program.

```
void loop() {
  if (CircuitPlayground.leftButton()) {
    //
    // YES
    //
    for (int p=0; p&lt;10; p++) {
      CircuitPlayground.setPixelColor(p, 0xFF6600);
    }
    CircuitPlayground.playTone(700, 750);
    CircuitPlayground.clearPixels();
  }
  if (CircuitPlayground.rightButton()) {
    //
    // NO
    //
    for (int p=0; p&lt;10; p++) {
      CircuitPlayground.setPixelColor(p, 0xFF6600);
    }
    CircuitPlayground.playTone(700, 500);
    CircuitPlayground.clearPixels();
    delay(250);
    for (int p=0; p&lt;10; p++) {
      CircuitPlayground.setPixelColor(p, 0xFF6600);
    }
    CircuitPlayground.playTone(700, 500);
    CircuitPlayground.clearPixels();
  }
}
```

It is comprised of two `if` blocks that check for either a left or a right button press. Then, there's a bunch of stuff that happens if a button is pressed. You should think of that 'bunch of stuff' as one chunk of code. There's a 'bunch of stuff' for what should happen when the left button is pressed, and another 'bunch of stuff' for when the right button is pressed. In each case, that 'bunch of stuff' is what creates the YES and NO indications.

So how about doing something like this?

```
void loop() {
  if (CircuitPlayground.leftButton()) {
    indicateYes();
  }
  if (CircuitPlayground.rightButton()) {
    indicateNo();
  }
}
```

That should be much more readable compared to the original code. When then just need to implement the two functions `indicateYes()` and `indicateNo()`. To do so, we just move (refactor) the code from the original version into the functions.

Here's the complete code with the new functions defined.

```
////////////////////////////////////////////////////////////////////////////
// Circuit Playground Yes No v1
//
// One beep (left button) = Yes
// Two beeps (right button) = No
//
// Author: Carter Nelson
// MIT License (https://opensource.org/licenses/MIT)

#include &lt;Adafruit_CircuitPlayground.h&gt;

////////////////////////////////////////////////////////////////////////////
void indicateYes() {
  for (int p=0; p&lt;10; p++) {
    CircuitPlayground.setPixelColor(p, 0xFF6600);
  }
  CircuitPlayground.playTone(700, 750);
  CircuitPlayground.clearPixels();
}

////////////////////////////////////////////////////////////////////////////
void indicateNo() {
  for (int p=0; p&lt;10; p++) {
    CircuitPlayground.setPixelColor(p, 0xFF6600);
  }
  CircuitPlayground.playTone(700, 500);
  CircuitPlayground.clearPixels();
  delay(250);
  for (int p=0; p&lt;10; p++) {
    CircuitPlayground.setPixelColor(p, 0xFF6600);
  }
  CircuitPlayground.playTone(700, 500);
  CircuitPlayground.clearPixels();
}
```

```
////////////////////////////////////////////////////////////////////////
void setup() {
  CircuitPlayground.begin();
}

////////////////////////////////////////////////////////////////////////
void loop() {
  if (CircuitPlayground.leftButton()) {
    indicateYes();
  }
  if (CircuitPlayground.rightButton()) {
    indicateNo();
  }
}
```

And BOOM - that's refactoring. While this is a fairly simple example, hopefully you can see how it can make the code more readable. The `loop()` now looks very simple - and it should. This is a very simple program.

---

# Refactor 2

Multiple lines of code that are repeated in multiple locations are another good candidate for refactoring. For example, these lines appear in two locations.

```
  for (int p=0; p&lt;10; p++) {
    CircuitPlayground.setPixelColor(p, 0xFF6600);
  }
```

Functionally, they turn on all of the NeoPixels to the given color. Let's refactor that functionality out into a new function called `showPixels()`.

```
void showPixels(uint32_t color) {
  for (int p=0; p&lt;10; p++) {
    CircuitPlayground.setPixelColor(p, color);
  }
}
```

Now we can just call that function in our `indicateYes()` and `indicateNo()` functions. Here's the new code with all these new changes.

```
////////////////////////////////////////////////////////////////////////
// Circuit Playground Yes No v2
//
// One beep (left button) = Yes
// Two beeps (right button) = No
//
// Author: Carter Nelson
// MIT License (https://opensource.org/licenses/MIT)

#include &lt;Adafruit_CircuitPlayground.h&gt;

////////////////////////////////////////////////////////////////////////
void showPixels(uint32_t color) {
  for (int p=0; p&lt;10; p++) {
```

```
    CircuitPlayground.setPixelColor(p, color);
  }
}

////////////////////////////////////////////////////////////////////////////
void indicateYes() {
  showPixels(0xFF6600);
  CircuitPlayground.playTone(700, 750);
  CircuitPlayground.clearPixels();
}

////////////////////////////////////////////////////////////////////////////
void indicateNo() {
  showPixels(0xFF6600);
  CircuitPlayground.playTone(700, 500);
  CircuitPlayground.clearPixels();
  delay(250);
  showPixels(0xFF6600);
  CircuitPlayground.playTone(700, 500);
  CircuitPlayground.clearPixels();
}

////////////////////////////////////////////////////////////////////////////
void setup() {
  CircuitPlayground.begin();
}

////////////////////////////////////////////////////////////////////////////
void loop() {
  if (CircuitPlayground.leftButton()) {
    indicateYes();
  }
  if (CircuitPlayground.rightButton()) {
    indicateNo();
  }
}
```

More importantly, we can now use this function anywhere else. This could be useful as the code grows and you find you want to turn on all the NeoPixels again somewhere else for some reason. Well, now there's a convenient little function you can call to do that. No need to write the `loop()` over and over again.

---

# Refactor 3

Here's some sagely guidance I once got from an excellent programmer:

> Design your code to work for 0, 1, or infinite cases.

Designing for 0 cases is the trivial answer, don't write any code at all. Done. Designing for 1 case makes things easy as you can get away with sort of hardwiring things for that case. For example, we loop over only 10 NeoPixels in `showPixels()` as we are designing specifically for the Circuit Playground, which has 10 NeoPixels.

Designing for infinite cases is quite often much more difficult. However, we have an opportunity to do some of this in our code. The two functions `indicateYes()` and `indicateNo()` basically do the same thing - they turn on the NeoPixels and make a

beep noise a certain amount of times. Yeah, we're only doing it once for "yes" and twice for "no", but it's not too difficult to create a function that could do this any number of times.

```
void lightsBeeps(int repeats, int note, int duration, uint32_t color) {
  for (int n=0; n&lt;repeats; n++) {
    showPixels(color);
    CircuitPlayground.playTone(note, duration);
    CircuitPlayground.clearPixels();
    if (repeats&gt;1) delay(duration/2);
  }
}
```

Now we just call this function from `indicateYes()` and `indicateNo()` passing in the behavior we want. Like this:

```
void indicateYes() {
  lightsBeeps(1, 700, 750, 0xFF6600);
}

void indicateNo() {
  lightsBeeps(2, 700, 500, 0xFF6600);
}
```

Here's the complete code with all the new changes.

```
////////////////////////////////////////////////////////////////////////////
// Circuit Playground Yes No v3
//
// One beep (left button) = Yes
// Two beeps (right button) = No
//
// Author: Carter Nelson
// MIT License (https://opensource.org/licenses/MIT)

#include &lt;Adafruit_CircuitPlayground.h&gt;

////////////////////////////////////////////////////////////////////////////
void lightsBeeps(int repeats, int note, int duration, uint32_t color) {
  for (int n=0; n&lt;repeats; n++) {
    showPixels(color);
    CircuitPlayground.playTone(note, duration);
    CircuitPlayground.clearPixels();
    if (repeats&gt;1) delay(duration/2);
  }
}

////////////////////////////////////////////////////////////////////////////
void showPixels(uint32_t color) {
  for (int p=0; p&lt;10; p++) {
    CircuitPlayground.setPixelColor(p, color);
  }
}

////////////////////////////////////////////////////////////////////////////
void indicateYes() {
  lightsBeeps(1, 700, 750, 0xFF6600);
}

////////////////////////////////////////////////////////////////////////////
void indicateNo() {
```

```
  lightsBeeps(2, 700, 500, 0xFF6600);
}
////////////////////////////////////////////////////////////////////////////
void setup() {
  CircuitPlayground.begin();
}

////////////////////////////////////////////////////////////////////////////
void loop() {
  if (CircuitPlayground.leftButton()) {
    indicateYes();
  }
  if (CircuitPlayground.rightButton()) {
    indicateNo();
  }
}
```

---

# Refactor 4

Now consider how we are calling `lightsBeep()` in the `indicateYes()` and `indicateNo()` functions.

```
void indicateYes() {
  lightsBeeps(1, 700, 750, 0xFF6600);
}

void indicateNo() {
  lightsBeeps(2, 700, 500, 0xFF6600);
}
```

Note how we are specifying the color of the NeoPixels in two places. What if you wanted to change the color? You'd have to edit both of those lines of code. With only two places to look, this isn't that difficult. But as code grows, there's the possibility of it showing up in more places and the chances of overlooking one or more of them increases. Also, what if you needed that value somewhere else in your code. You'd have to remember what it was, or go find it, so you could type it in again.

The term 'magic number' is used to refer to these types of values.

A better way to deal with values like this is to define them in some global manner and then use that reference as needed in the code. For things that are constant, this is typically done using a `#define` as follows.

```
#define PIXEL_COLOR 0xFF6600
```

Note that there is no = sign. These lines that begin with # are actually dealt with before the code is even compiled. In the case of a `#define`, it works likes a simple search-and-replace. Every occurrence of `PIXEL_COLOR` is replaced with `0xFF6600`.

Now we can use this in our code like this:

```
void indicateYes() {
  lightsBeeps(1, 700, 750, PIXEL_COLOR);
}

void indicateNo() {
  lightsBeeps(2, 700, 500, PIXEL_COLOR);
}
```

And anytime we want to change the color, we only have to change one thing in one place. Here's the complete code with all the changes.

```
////////////////////////////////////////////////////////////////////////////
// Circuit Playground Yes No v4
//
// One beep (left button) = Yes
// Two beeps (right button) = No
//
// Author: Carter Nelson
// MIT License (https://opensource.org/licenses/MIT)

#include &lt;Adafruit_CircuitPlayground.h&gt;

#define PIXEL_COLOR 0xFF6600

////////////////////////////////////////////////////////////////////////////
void lightsBeeps(int repeats, int note, int duration, uint32_t color) {
  for (int n=0; n&lt;repeats; n++) {
    showPixels(color);
    CircuitPlayground.playTone(note, duration);
    CircuitPlayground.clearPixels();
    if (repeats&gt;1) delay(duration/2);
  }
}

////////////////////////////////////////////////////////////////////////////
void showPixels(uint32_t color) {
  for (int p=0; p&lt;10; p++) {
    CircuitPlayground.setPixelColor(p, color);
  }
}

////////////////////////////////////////////////////////////////////////////
void indicateYes() {
  lightsBeeps(1, 700, 750, PIXEL_COLOR);
}

////////////////////////////////////////////////////////////////////////////
void indicateNo() {
  lightsBeeps(2, 700, 500, PIXEL_COLOR);
}

////////////////////////////////////////////////////////////////////////////
void setup() {
  CircuitPlayground.begin();
}

////////////////////////////////////////////////////////////////////////////
void loop() {
  if (CircuitPlayground.leftButton()) {
    indicateYes();
  }
  if (CircuitPlayground.rightButton()) {
    indicateNo();
```

```
  }
}
```

---

# Refactor N

Does it ever end? Well, for this guide yes. But in real life, refactoring is something that is constantly done. It can also be a bit of a double edged sword. While some amount of refactoring is definitely good, it is possible to start chasing rabbits down holes or going in circles chasing your own tail.

The best practices for how and when to refactor come with experience. For now, just realize what it is and how it can help improve your code.

Make sense?

BEEP! BEEP!

Double yes! Thanks. Glad it helped.

---

# Questions and Code Challenges

The following are some questions related to this project along with some suggested code challenges. The idea is to provoke thought, test your understanding, and get you coding!

While the sketches provided in this guide work, there is room for improvement and additional features. Have fun playing with the provided code to see what you can do with it.

# Questions

- Should the values for `repeats`, `tone`, and `duration` be refactored?
- `indicateYes()` and `indicateNo()` are down to only one line, should they be refactored?

# Code Challenges

- Try changing the color in the `#define` to prove that is works.
- Add a third indication that happens when both buttons are pressed at the same time.

- Use the slide switch to set different modes for the indications.