# Table of Contents

## Overview
- Required Parts
- Other Items
- Before Starting

## Hello Capacitive Touch

## Basic Print

## Tone Piano

## Cirkey Cirkey

## Fruit Drums
- MIDI USB Library Install
- MIDI Drum Sketch
- MIDI Drum Machine Software
- Making the Fruit Drums
Overview

In this guide we will explore the capacitive touch capabilities of the Circuit Playground and work our way towards putting together a USB MIDI drum machine driven by fruit. We'll also turn the Circuit Playground into a 'keyboard' so you can use it to play games or run an application that uses arrow keys.

Required Parts

This project will work mainly with the Circuit Playground attached to a computer via a USB cable. Alligator clips will be used to attach to the capacitive touch pads.
Alligator Clips (http://adafruit.it/1008)

USB cable (http://adafruit.it/592) for programming and power.

Other Items

Various other items will be used. These do not need to be exact, anything similar will be fine.

Paper (6"x4" cards)
Marker
Ruler

And of course.....FRUIT!
Before Starting

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

- Overview
- Lesson #0

This project will use the Arduino IDE. Make sure you have added the board support for the Circuit Playground as well as installed the Circuit Playground library. MUST DO BOTH. This is covered in the guides linked above.

Hello Capacitive Touch

Since this project uses the capacitive touch capabilities of the Circuit Playground, let's start by exploring how they work. You can read some technical details in the Lesson #0 Guide. There are 8 pads total located on the edge of the Circuit Playground as shown below. Note the numbering scheme used to identify each pad: #3, #2, #0, #1, #12, #6, #9, and #10.
The library function `readCap()` is all that is needed to use capacitive touch. It is used the same for all 8 pads. Simply call it with the pad number of interest and it returns a value. This value will be near zero when not touched and have some higher value when touched. We can use the simple sketch below to examine how this value behaves, focusing on just one pad, #1 in this case.

```cpp
#include <Adafruit_CircuitPlayground.h>

void setup() {
  // Initialize serial.
  Serial.begin(9600);

  // Initialize Circuit Playground library.
  CircuitPlayground.begin();
}

void loop() {
  // Print cap touch reading on serial port.
  Serial.println(CircuitPlayground.readCap(1));

  // Update rate.
  delay(100);
}
```
This sketch only uses the #1 touch pad.

With this sketch loaded and running on the Circuit Playground, open the Serial Plotter.

Tools -> Serial Plotter

The capacitive touch value from the #1 pad will be plotted like a strip chart as shown below. Try touching the pad quickly several times and also holding your finger on it.

The `readCap()` function is different from the push button functions `leftButton()` and `rightButton()` which return true when the button is pressed and false otherwise. However, we can take a very simple approach to create a similar behavior for the capacitive touch pads. We compare the value returned by `readCap()` to a preset threshold value which we'll call CAP_THRESHOLD. If it exceeds this value, it is touched (pressed), otherwise, it is not. Here's a simple function to do just that.

```java
boolean capButton(uint8_t pad) {
  if (CircuitPlayground.readCap(pad) > CAP_THRESHOLD) {
    return true;
  } else {
    return false;
  }
}
```
The actual value for CAP_THRESHOLD will be defined globally. But what is a good value? Well, that will depend on your setup. If the value is set too low, you may get incorrect detections due to noise. But if you set it too high, you may never detect any touches. You could use the Serial Monitor or Plotter with the above sketch to see what values your setup returns, and then choose a value based on that. For the examples here, we will use a value of 50.

Here is another simple sketch that uses the `capButton()` function along with a CAP_THRESHOLD of 50 to detect when the #1 pad is touched.

```cpp
#include <Adafruit_CircuitPlayground.h>

#define CAP_THRESHOLD 50
#define DEBOUNCE 250

boolean capButton(uint8_t pad) {
  if (CircuitPlayground.readCap(pad) > CAP_THRESHOLD) {
    return true;
  } else {
    return false;
  }
}

void setup() {
  // Initialize serial.
  Serial.begin(9600);

  // Initialize Circuit Playground library.
  CircuitPlayground.begin();
}

void loop() {
  // Check if capacitive touch exceeds threshold.
  if (capButton(1)) {
    // Print message.
    Serial.println("Touched!");
    // But not too often.
    delay(DEBOUNCE);
  }
}
```

The additional value DEBOUNCE is used to prevent the output from happening too fast. With the above sketch loaded and running, open the Serial Monitor.

Tools -> Serial Monitor

Whenever the #1 pad is touched, you should see a message printed out.
Now let's move on to generalizing the code to detect all of the touch pads and take various actions depending on what pad is pressed.

# Basic Print

Let's start simple. This sketch prints a unique message for each touch pad. Download it and load it to the Circuit Playground.

```cpp
// SPDX-FileCopyrightText: 2020 Carter Nelson for Adafruit Industries
// SPDX-License-Identifier: MIT

///// Circuit Playground Capacitive Touch Basic
//
// Print text messages for each touch pad.
//
// Author: Carter Nelson
// MIT License (https://opensource.org/licenses/MIT)

#include <Adafruit_CircuitPlayground.h>
#define CAP_THRESHOLD   50
#define DEBOUNCE        250
uint8_t pads[] = {3, 2, 0, 1, 12, 6, 9, 10};
uint8_t numberOfPads = sizeof(pads)/sizeof(uint8_t);

void takeAction(uint8_t pad) {
    Serial.print("PAD "); Serial.print(pad); Serial.print(" says: ");
    switch (pad) {
      case 3:
        Serial.println("The goggles! They do nothing!");
        break;
      case 2:
        Serial.println("Dental plan!");
        break;
      case 0:
        Serial.println("Lisa needs braces.");
        break;
      case 1:
        Serial.println("I call the large one Bitey.");
        break;
    }
}
```
Serial.println("I'm Idaho!");
break;
case 6:
    Serial.println("Monorail!");
    break;
case 9:
    Serial.println("I choo-choo choose you.");
    break;
case 10:
    Serial.println("Maaaatllllock!");
    break;
default:
    Serial.println("THIS SHOULD NEVER HAPPEN.");
}

////////////////////////////////////////////////////////////////////////////
boolean capButton(uint8_t pad) {
    // Check if capacitive touch exceeds threshold.
    if (CircuitPlayground.readCap(pad) > CAP_THRESHOLD) {
        return true;
    } else {
        return false;
    }
}

////////////////////////////////////////////////////////////////////////////
void setup() {
    // Initialize serial.
    Serial.begin(9600);

    // Initialize Circuit Playground library.
    CircuitPlayground.begin();
}

////////////////////////////////////////////////////////////////////////////
void loop() {
    // Loop over every pad.
    for (int i=0; i<numberOfPads; i++) {

        // Check if pad is touched.
        if (capButton(pads[i])) {

            // Do something.
            takeAction(pads[i]);

            // But not too often.
            delay(DEBOUNCE);
        }
    }
}

With this sketch loaded and running on the Circuit Playground, open the Serial Monitor.

Tools -> Serial Monitor

Now press any of the 8 touch pads. You should see a message printed out for each one.
Take a look at the `loop()` function in this sketch. The main change from the previous sketch is the addition of an outer `for` loop which scans over each of the touch pads on the Circuit Playground.

```cpp
void loop() {
  // Loop over every pad.
  for (int i=0; i<numberOfPads; i++) {
    // Check if pad is touched.
    if (capButton(pads[i])) {
      // Do something.
      takeAction(pads[i]);
      // But not too often.
      delay(DEBOUNCE);
    }
  }
}
```

The variable `numberOfPads` and the array `pads[]` have been defined globally near the top of the sketch. The sequence of the numbers in `pads[]` is such that they are scanned in a counter-clockwise fashion starting with pad #3.

```cpp
uint8_t pads[] = {3, 2, 0, 1, 12, 6, 9, 10};
uint8_t numberOfPads = sizeof(pads)/sizeof(uint8_t);
```

Also, a new function called `takeAction()` has been created. This allows us to use this same `loop()` setup for all of the sketches. All we have to do is change the code in `takeAction()` to change the behavior of the Circuit Playground.

If you look at the code in `takeAction()` you will see it contains a `switch` statement. This is where the main plumbing between Circuit Playground touch pads and actual code takes place. The `case` statements correspond to pad numbers and the code inside the case statement defines what is executed for each pad.

For example, for pad #12, the following lines of code are executed:
The `break` is needed to exit out of the `switch` statement.

Keen. Now let's do something a little more exciting than just printing out Simpsons quotes.

---

**Tone Piano**

The following sketch uses the Circuit Playground speaker to play a different tone for each pad when touched.

```cpp
// SPDX-FileCopyrightText: 2020 Carter Nelson for Adafruit Industries
// SPDX-License-Identifier: MIT

const uint8_t pads[] = {3, 2, 0, 1, 12, 6, 9, 10};
uint8_t numberOfPads = sizeof(pads)/sizeof(uint8_t);

void takeAction(uint8_t pad) {
    Serial.print("PAD "); Serial.print(pad); Serial.print(" playing note: ");
    switch (pad) {
    case 3:
        Serial.println("C");
        CircuitPlayground.playTone(262, 100, false);
        break;
    case 2:
        Serial.println("D");
        CircuitPlayground.playTone(294, 100, false);
        break;
    case 0:
        Serial.println("E");
        CircuitPlayground.playTone(330, 100, false);
        break;
    case 1:
        Serial.println("F");
        CircuitPlayground.playTone(349, 100, false);
        break;
    case 12:
        Serial.println("G");
        CircuitPlayground.playTone(392, 100, false);
        break;
    case 6:
        Serial.println("A");
    }
```
CircuitPlayground.playTone(440, 100, false);
break;
case 9:
    Serial.println("B");
    CircuitPlayground.playTone(494, 100, false);
    break;
case 10:
    Serial.println("C");
    CircuitPlayground.playTone(523, 100, false);
    break;
default:
    Serial.println("THIS SHOULD NEVER HAPPEN.");
}

boolean capButton(uint8_t pad) {
    // Check if capacitive touch exceeds threshold.
    if (CircuitPlayground.readCap(pad) > CAP_THRESHOLD) {
        return true;
    } else {
        return false;
    }
}

void setup() {
    // Initialize serial.
    Serial.begin(9600);

    // Initialize Circuit Playground library.
    CircuitPlayground.begin();
}

void loop() {
    // Loop over every pad.
    for (int i=0; i<numberOfPads; i++) {

        // Check if pad is touched.
        if (capButton(pads[i])) {

            // Do something.
            takeAction(pads[i]);
        }
    }
}

Make sure you are using version 1.6.4 or newer of the Circuit Playground library.

Look in the `takeAction()` function and you will see a call to `playTone()` for each of the pads. The frequencies correspond to the basic musical notes. With this sketch loaded and running on the Circuit Playground, you should hear different tones played when a pad is touched.

We can use alligator clips and some paper to create a little piano. On a 6"x4" index card or other piece of paper, draw something similar to the picture below.
Now attach the alligator clips to the edge of the paper and the Circuit Playground as shown below.

Once you've clipped the alligators, press the reset button on the Circuit Playground so it will re-calibrate the capacitive touch sensors.
To play a note, just touch the alligator clip on the edge of the paper. For example, in the picture below, E is being played.

Try out the following sequence of notes.

C-D-E-C-E-C-E  D-E-F-F-E-D-F

Do you recognize the tune? Here's a hint.

Cirkey Cirkey

One of the more advanced features of the Circuit Playground is its ability to emulate a USB keyboard and mouse. To do this, we use the Arduino Mouse and Keyboard libraries.
The following sketch uses these libraries to emulate a USB keyboard and mouse and send key and mouse events when the pads are touched.

```c
// SPDX-FileCopyrightText: 2020 Carter Nelson for Adafruit Industries
// SPDX-License-Identifier: MIT

////////////////////////////////////////////////////////////////////////////
// Circuit Playground Capacitive Touch Tones
// Send keyboard and mouse events for each touch pad.
// https://www.arduino.cc/en/Reference/MouseKeyboard
// Author: Carter Nelson
// MIT License (https://opensource.org/licenses/MIT)

#include <Adafruit_CircuitPlayground.h>
#include <Mouse.h>
#include <Keyboard.h>

#define CAP_THRESHOLD   50
#define DEBOUNCE        250

uint8_t pads[] = {3, 2, 0, 1, 12, 6, 9, 10};
uint8_t numberOfPads = sizeof(pads)/sizeof(uint8_t);

boolean emulatorActive = false;

////////////////////////////////////////////////////////////////////////////
void takeAction(uint8_t pad) {
  Serial.print("PAD "); Serial.println(pad);
  switch (pad) {
    case 3:
      sendKey(KEY_LEFT_ARROW);
      break;
    case 2:
      sendKey(KEY_UP_ARROW);
      break;
    case 0:
      sendKey(KEY_DOWN_ARROW);
      break;
    case 1:
      sendKey(KEY_RIGHT_ARROW);
      break;
    case 12:
      sendKey(' ');
      break;
    case 6:
      sendMouse(MOUSE_LEFT);
      break;
    case 9:
      sendMouse(MOUSE_MIDDLE);
      break;
    case 10:
      sendMouse(MOUSE_RIGHT);
      break;
    default:
      Serial.println("THIS SHOULD NEVER HAPPEN.");
      break;
  }
}

////////////////////////////////////////////////////////////////////////////
boolean capButton(uint8_t pad) {
  // Check if capacitive touch exceeds threshold.
  if (CircuitPlayground.readCap(pad) > CAP_THRESHOLD) {
    return true;
  }
}```
} else {
    return false;
}

void sendKey(char key) {
    Keyboard.press(key);
    Keyboard.releaseAll();
}

void sendMouse(char key) {
    Mouse.click(key);
}

void setup() {
    // Initialize serial.
    Serial.begin(9600);

    // Initialize Circuit Playground library.
    CircuitPlayground.begin();
}

void loop() {
    // Indicate emulator status on red LED.
    CircuitPlayground.redLED(emulatorActive);

    // Check slide switch.
    if (!CircuitPlayground.slideSwitch()) {
        //-----------| EMULATOR OFF |-------------
        if (emulatorActive) {
            Keyboard.end();
            Mouse.end();
            emulatorActive = false;
        }
    } else {
        //-----------| EMULATOR ON |-------------
        if (!emulatorActive) {
            Keyboard.begin();
            Mouse.begin();
            emulatorActive = true;
        }
        // Loop over every pad.
        for (int i=0; i<numberOfPads; i++) {
            // Check if pad is touched.
            if (capButton(pads[i])) {
                // Do something.
                takeAction(pads[i]);
                // But not too often.
                delay(DEBOUNCE);
            }
        }
    }
}
If this sounds familiar that's because this is essentially what a Makey Makey does. The default key mapping in the sketch provided above is similar to the original Makey Makey.

- PAD #3: Left Arrow
- PAD #2: Up Arrow
- PAD #0: Down Arrow
- PAD #1: Right Arrow
- PAD #12: Space Bar
- PAD #6: Left Mouse Button Click
- PAD #9: Middle Mouse Button Click
- PAD #10: Right Mouse Button Click

We can use alligator clips and some paper to create a cheat sheet for the key mapping. On a 6"x4" index card or other piece of paper, draw something similar to the picture below.

Now attach the alligator clips to the edge of the paper and the Circuit Playground as shown below.
To send the key press or mouse button click, touch the associated alligator clip.

READ THE FOLLOWING TO PREVENT UNEXPECTED BEHAVIOR.

Since there is a danger of flooding your computer with too many key presses or mouse clicks, the slide switch on the Circuit Playground is used to turn the keyboard/mouse emulation on and off. With the slide switch to the left (+), the emulator is active and the red LED will come on. The Circuit Playground will send key and mouse events with every capacitive touch. To turn the emulator off, set the slide switch to the right (-) position.

With this sketch loaded and running on the Circuit Playground, turn on the emulator using the slide switch. Then try out these Makey Makey apps:

- Bongos ()
- Piano ()
Fruit Drums

The Circuit Playground can also emulate a USB MIDI drum kit. This will let us play drums using the capacitive touch pads, or anything we hook up to them, like a bunch of fruit. But first, a bit of info on MIDI.

MIDI stands for Musical Instrument Digital Interface and encompasses a whole world of various hardware and software products. For our purposes, we can think in terms of a very simple setup as shown below.

The MIDI PRODUCER generates MIDI SIGNALS which are received by a MIDI CONSUMER that processes the command in the MIDI SIGNAL and takes some kind of action.

For a general overview of the structure of MIDI signals and the commands they contain, check out this episode of Collin's Lab.

For our setup, the Circuit Playground will be the MIDI PRODUCER and a computer will be the MIDI CONSUMER. The MIDI SIGNALS will be transmitted over USB. Software running on the computer will receive and process the MIDI commands.

Since we are going to turn the Circuit Playground into a drum kit, we can use a special set of MIDI commands defined in the General MIDI Specification. These commands use channel 10 to specify percussion (drums) and note numbers 35-81 to specify a specific instrument (kick, snare, hi-hat, etc.). For example, note number 56 is a cowbell.
Ok. Let's put the pieces together.

MIDI USB Library Install

In order to emulate a USB MIDI device, we will use the Arduino Library [MIDIUSB](https://www.arduino.cc/). To install it, open up the Library Manager.

Sketch -> Include Library -> Manage Libraries...

Filter the libraries by typing 'midi usb' in the search box in the upper right. Then select the MIDIUSB library and click Install.

MIDI Drum Sketch

Here is the Arduino sketch to turn the Circuit Playground into a USB MIDI drum machine.

```cpp
// SPDX-FileCopyrightText: 2020 Carter Nelson for Adafruit Industries
// // SPDX-License-Identifier: MIT

// Circuit Playground Capacitive Touch USB MIDI Drums
// // Send MIDI percussion commands over USB.
```

©Adafruit Industries
#include <Adafruit_CircuitPlayground.h>
#include "MIDIUSB.h"

#define CAP_THRESHOLD 50
#define DEBOUNCE 100

uint8_t pads[] = {3, 2, 0, 1, 12, 6, 9, 10};
uint8_t numberOfPads = sizeof(pads)/sizeof(uint8_t);

void takeAction(uint8_t pad) {
  Serial.print("PAD "); Serial.print(pad); Serial.print(. Sending MIDI: ");
  switch (pad) {
    case 3:  
      Serial.println("36");
      drumHit(36);
      break;
    case 2:  
      Serial.println("37");
      drumHit(37);
      break;
    case 0:  
      Serial.println("38");
      drumHit(38);
      break;
    case 1:  
      Serial.println("39");
      drumHit(39);
      break;
    case 12: 
      Serial.println("40");
      drumHit(40);
      break;
    case 6:  
      Serial.println("41");
      drumHit(41);
      break;
    case 9:  
      Serial.println("42");
      drumHit(42);
      break;
    case 10: 
      Serial.println("43");
      drumHit(43);
      break;
    default:
      Serial.println("THIS SHOULD NEVER HAPPEN.");
  }
}

boolean capButton(uint8_t pad) {
  // Check if capacitive touch exceeds threshold.
  if (CircuitPlayground.readCap(pad) > CAP_THRESHOLD) {
    return true;
  } else {
    return false;
  }
}

void noteOn(byte channel, byte pitch, byte velocity) {
  // Send a MIDI Note On command.

```c
midiEventPacket_t noteOn = {0x09, 0x90 | channel, pitch, velocity};
MidiUSB.sendMIDI(noteOn);
}

void noteOff(byte channel, byte pitch, byte velocity) {
// Send a MIDI Note Off command.
    midiEventPacket_t noteOff = {0x08, 0x80 | channel, pitch, velocity};
    MidiUSB.sendMIDI(noteOff);
}

void drumHit(byte drum) {
    // Send Note On/Off command for given drum (note) number.
    noteOn(9, drum, 125);
    MidiUSB.flush();
    noteOff(9, drum, 0);
    MidiUSB.flush();
}

void setup() {
    // Initialize serial.
    Serial.begin(9600);

    // Initialize Circuit Playground library.
    CircuitPlayground.begin();
}

void loop() {
    // Loop over every pad.
    for (int i=0; i<numberOfPads; i++) {
        // Check if pad is touched.
        if (capButton(pads[i])) {
            // Do something.
            takeAction(pads[i]);
            // But not too often.
            delay(DEBOUNCE);
        }
    }
}
```

Download it and use the Arduino IDE to upload it to the Circuit Playground.

The following was only tested on Linux.

**MIDI Drum Machine Software**

We will use the software application [Hydrogen](#) to receive the MIDI signals from the Circuit Playground and play drum sounds. [Downloads](#) are available for Linux, Windows, and OS X.

The Hydrogen application is a full featured drum machine that let's you create drum patterns and full songs. For our simple Circuit Playground drum machine, we will
ignore most of the user interface. The drum sounds are shown on the left side of the window and the corresponding Circuit Playground touch pad is shown in the figure below.

To enable the Circuit Playground in the Hydrogen software, first load the above sketch and make sure the Circuit Playground is connected via USB. Now, open the preferences window:

Tools -> Preferences

and select the Midi System tab. In the Input drop down, select the option with Circuit Playground in it. Also, check the Ignore note-off box. Then click OK.
At this point, if you touch one of the pads on the Circuit Playground you should hear a drum sound.

Making the Fruit Drums

Let's connect some fruit! Pretty much anything should work. Just get a bunch of fruit and hook them up using the alligator clips.

The alligator clip can be connected directly to fruit with a stem.
For fruit without a stem, a short jumper wire can be used.

One end of the alligator clip goes to the fruit, the other end to the touch pad on the Circuit Playground.

Once you've got your fruit all connected and arranged, it's time to rock out. The video below shows a brief demo of the Circuit Playground USB MIDI Fruit Drum Machine in action. Note that the Hydrogen app is running on the monitor in the background. The Circuit Playground is attached to the same PC via the USB cable.