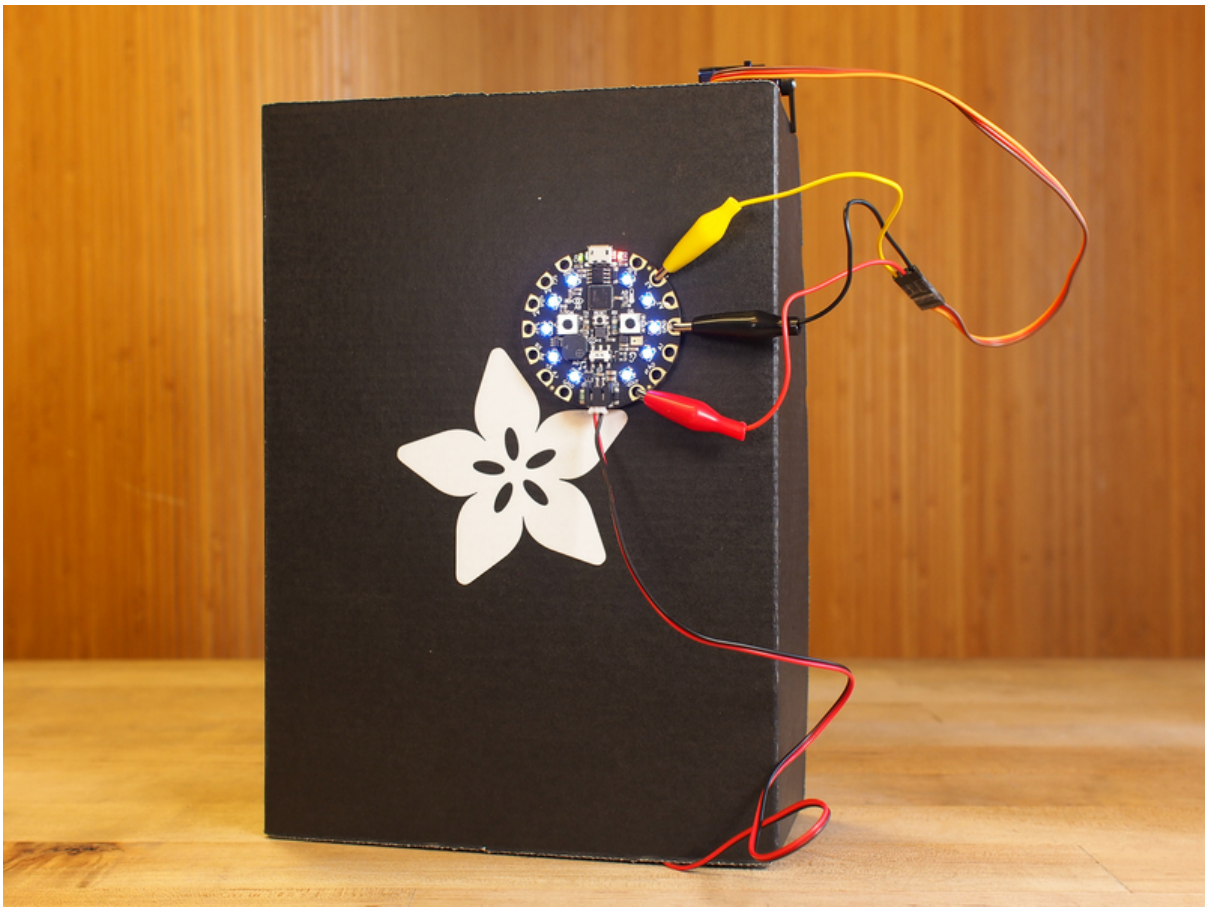
Combo Dial Safe with Circuit Playground Express

Created by John Park

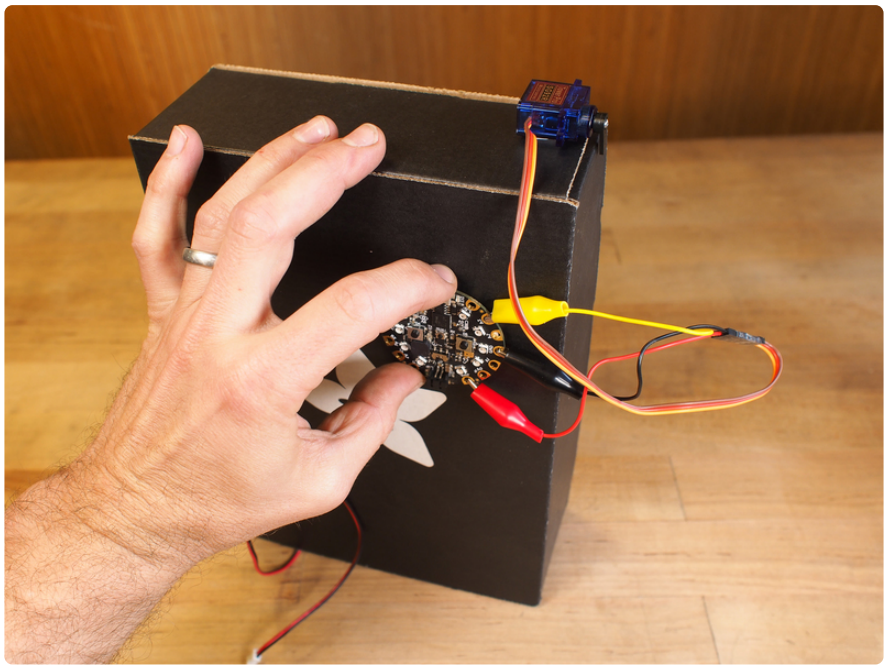


https://learn.adafruit.com/combo-dial-safe-with-circuit-playground-express

Last updated on 2024-03-08 02:53:31 PM EST

Table of Contents

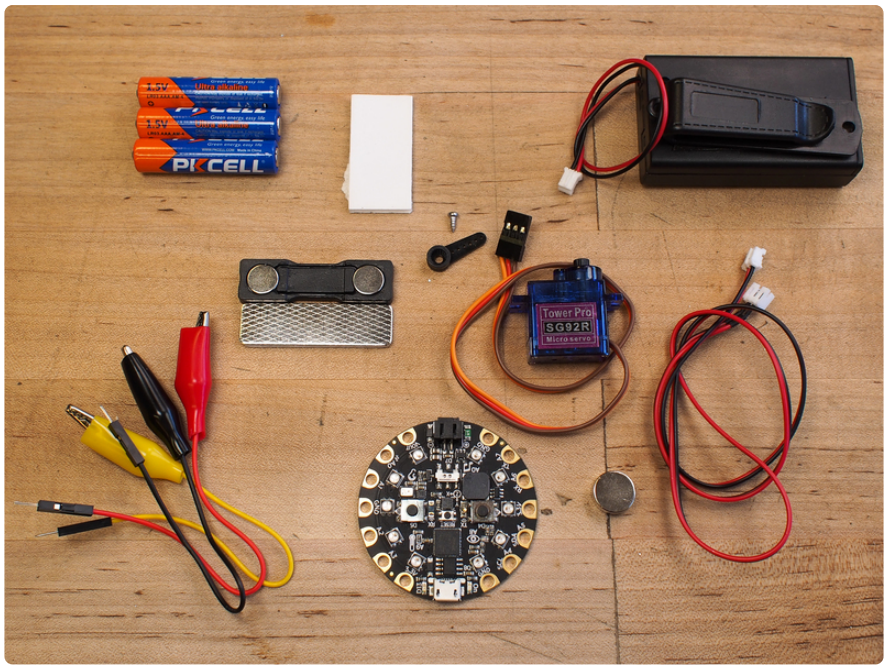
Overview



You can build a working combination dial safe using the Circuit Playground Express, a servo motor, a box, and a few magnets, wires, and batteries!

The Circuit Playground Express has everything we'll need to act as a dial that measures rotation, can give feedback through colored lights and sounds, and even control the lock!

You'll be able to code it all in CircuitPython, including reading the built-in accelerator, setting a custom combinations, and more! Plus, you don't need any compilers or IDEs, just a text editor and a USB cable to connect.



1 x Circuit Playground Express
with CircuitPython
https://www.adafruit.com/product/3333

---

1 x Micro servo
with servo arms
https://www.adafruit.com/product/169

---

1 x 3 x AAA Battery Holder
with On/Off Switch, JST, and Belt Clip
https://www.adafruit.com/product/3286

---

1 x Alkaline AAA batteries
3 pack
https://www.adafruit.com/product/3520

---

3 x Alligator Clip To Breadboard Plug
set of 12
https://www.adafruit.com/product/3255

---

1 x Rare Earth Magnet
high strength
https://www.adafruit.com/product/9

---

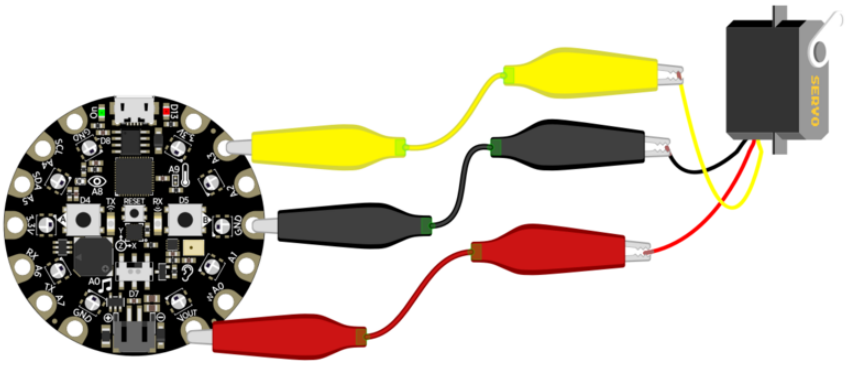
1 x Magnetic Pin Back
with foam tape
https://www.adafruit.com/product/1170

Materials & Tools

You can use any lidded box of your choosing, but a really good choice is the black, cardboard Adafruit box!

The only tools you'll need are a small Philips screwdriver and some Scotch tape. Optionally, you can use a silver or white marker to decorate your safe.

Build the Circuit



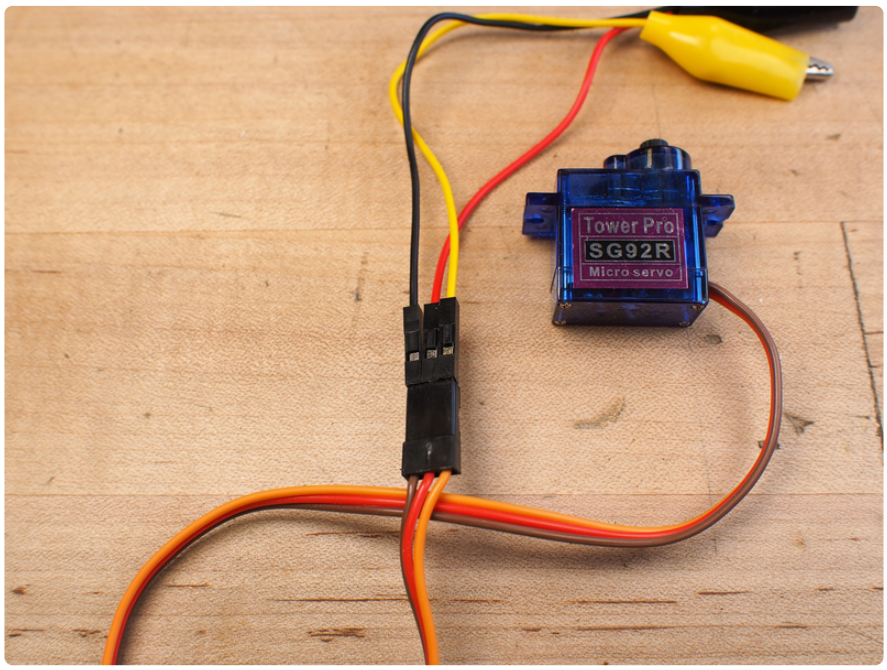
We'll be using the servo motor as the locking/unlocking latch for our safe. The servo motor is different from a regular DC motor, in that it can be precisely controlled to rotate to a specific angle. This is why there are three wires coming from the servo, instead of the usual two on a DC motor for ground and voltage.

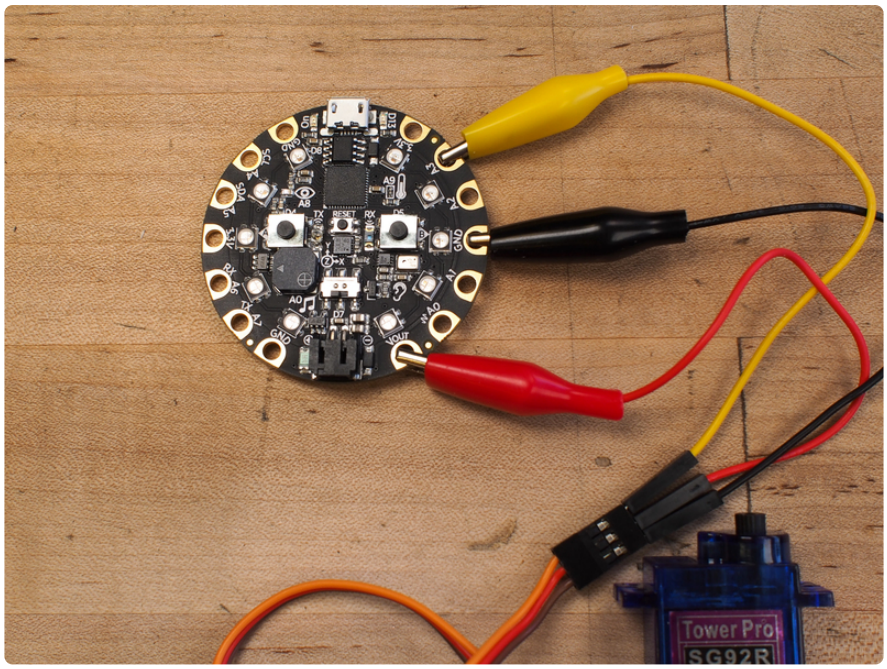
The third wire on the servo controls the desired angle by receiving a particular command from the Circuit Playground Express. This control is send in the form of a pulse-width modulated (PWM) signal. We'll take a closer look at this in the coding section on the next page.

Wiring

Using three small alligator clip leads, connect the Circuit Playground Express to the servo's wires as shown in the diagram above, plugging the male ends of the small leads into the servo connector plug.

- Red goes from **VOUT** to servo red voltage wire
- Black goes from **GND** to servo brown ground wire
- Yellow goes from **A3** to servo orange signal wire





Before we write the safe code, let's go ahead and try out the servo!

Copy the code below into your text editor and then save it to the Circuit Playground Express as `main.py`

```
import time
import board
import pulseio
from adafruit_motor import servo

# create a PWMOut object on Pin A3.
pwm = pulseio.PWMOut(board.A3, duty_cycle=2 ** 15, frequency=50)

# Create a servo object, my_servo.
my_servo = servo.Servo(pwm)

while True:
    for angle in range(0, 180, 5):  # 0 - 180 degrees, 5 degrees at a time.
        my_servo.angle = angle
        time.sleep(0.05)
    for angle in range(180, 0, -5): # 180 - 0 degrees, 5 degrees at a time.
        my_servo.angle = angle
        time.sleep(0.05)
```

The servo will sweep back and forth. Very satisfying.

Next, we'll code the Combo Dial Safe with CircuitPython!

Code with CircuitPython

Let's look at the code in CircuitPython to control the dial and lock.

We'll be using the `Adafruit Circuit Playground Express` library, as well as the `Adafruit_Motor` library. These will give us the commands we need to read the accelerometer for rotation angle of the entire board, play sounds, light up NeoPixels, as well as to control the servo motor's rotation.

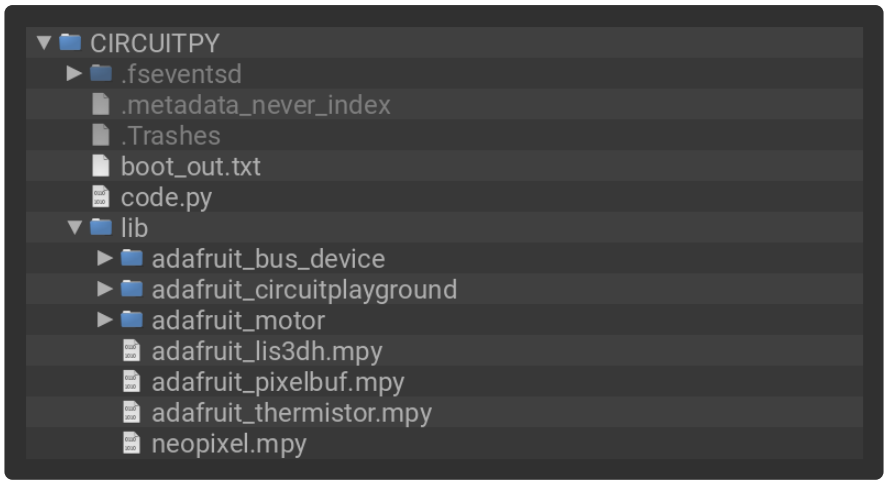
Get Ready!

1. First, make sure you're familiar with the basics of using CircuitPython on the Circuit Playground Express. Follow this guide (https://adafru.it/Biv) to familiarize yourself.
2. Then, install CircuitPython on your board by following these instructions (https://adafru.it/AFI).
3. The last thing to do to prepare is to install the library bundle onto your board as shown here (https://adafru.it/C9M). The libraries give us what we need to code easily with high level commands!

Download the latest bundle from this link.

https://adafru.it/zB-

Once you've uncompressed the contents of the zip file, drag its contents to your Circuit Playground Express **lib** directory.



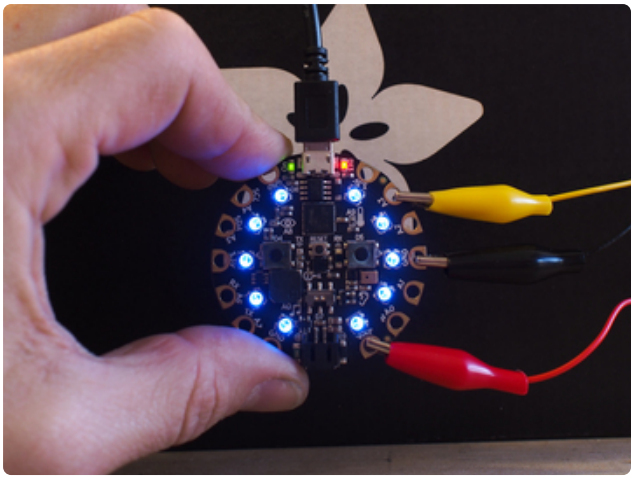
Download the Combo Dial Python Code

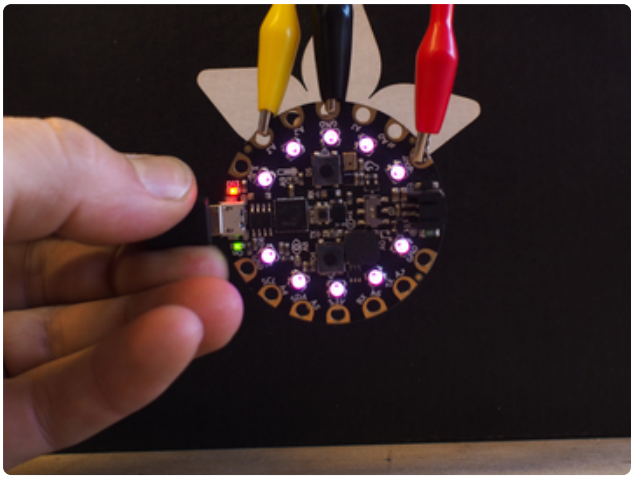
Copy the code below, and paste it into a new text document in your text editor, or in the Mu code editor for CircuitPython.

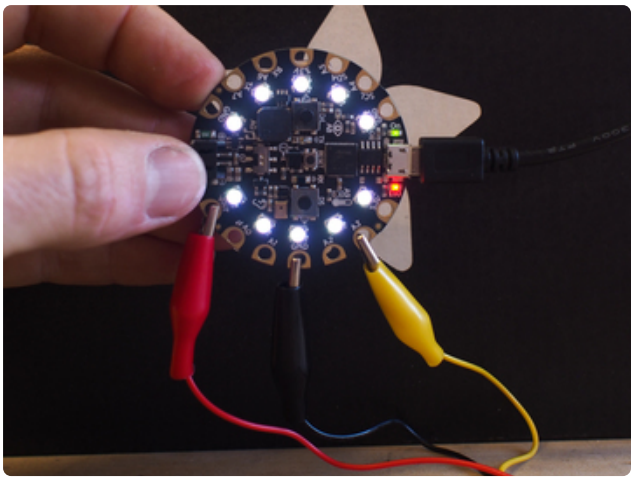
then save it to you Circuit Playground Express board as **code.py**.

```
# SPDX-FileCopyrightText: 2017 John Park for Adafruit Industries
#
# SPDX-License-Identifier: MIT

# Combo Dial Safe
# for Adafruit Circuit Playground express
# with CircuitPython

import time

import board
import pwmio
from adafruit_motor import servo
from adafruit_circuitplayground.express import cpx

pwm = pwmio.PWMOut(board.A3, duty_cycle=2 ** 15, frequency=50)

#  plug red servo wire to VOUT, brown to GND, yellow to A3
servo = servo.Servo(pwm)

cpx.pixels.brightness = 0.05  # set brightness value


def unlock_servo():
    servo.angle = 180


def lock_servo():
    servo.angle = 90


correct_combo = ['B', 'D', 'C']  # this is where to set the combo
entered_combo = []  # this will be used to store attempts
current_dial_position = 'X'

cpx.red_led = 1  # turn off the on-board red LED while locked
lock_servo()  # lock the servo

while True:
    x_float, y_float, z_float = cpx.acceleration  # read acceleromter
    x = int(x_float)  # make int of it
    y = int(y_float)
    z = int(z_float)

    # four simple rotation positions, A-D
    # the combination entries are based on which letter is facing up
    #
    #              A
    #            .___.
    #         .         .
    #     D  .           .  B
    #        .           .
    #         .         .
    #            .|_|.
    #              C

    if x == 0 and y == 9:
        current_dial_position = 'A'  # used to store dial position
        cpx.pixels.fill((0, 0, 255))

    if x == 9 and y == 0:
        current_dial_position = 'B'
        cpx.pixels.fill((80, 0, 80))

    if x == 0 and y == -9:
```

```
        current_dial_position = 'C'
        cpx.pixels.fill((255, 70, 0))

    if x == -9 and y == 0:
        current_dial_position = 'D'
        cpx.pixels.fill((255, 255, 255))

    # press the right/B button to lock the servo
    if cpx.button_b:  # this is a more Pythonic way to check button status
        print('Locked/Reset')
        cpx.red_led = 1
        cpx.pixels.fill((50, 10, 10))
        lock_servo()
        cpx.play_tone(120, 0.4)
        cpx.pixels.fill((0, 0, 0))
        entered_combo = []  # clear this for next time around
        time.sleep(1)

    # press the left/A button to enter the current position as a combo entry
    if cpx.button_a:  # this means the button has been pressed
        # grab the current_dial_position value and add to the list
        entered_combo.append(current_dial_position)
        dial_msg = 'Dial Position: ' + \
                   str(entered_combo[(len(entered_combo) - 1)])
        print(dial_msg)
        cpx.play_tone(320, 0.3)  # beep
        time.sleep(1)  # slow down button checks

    if len(entered_combo) == 3:
        if entered_combo == correct_combo:  # they match!
            print('Correct! Unlocked.')
            cpx.red_led = 0  # turn off the on board LED
            cpx.pixels.fill((0, 255, 0))
            unlock_servo()
            cpx.play_tone(440, 1)
            time.sleep(3)
            entered_combo = []  # clear this for next time around

        else:
            print('Incorret combination.')
            cpx.pixels.fill((255, 0, 0))
            cpx.play_tone(180, 0.3)  # beep
            cpx.play_tone(130, 1)  # boop
            time.sleep(3)
            entered_combo = []  # clear this for next time around
```

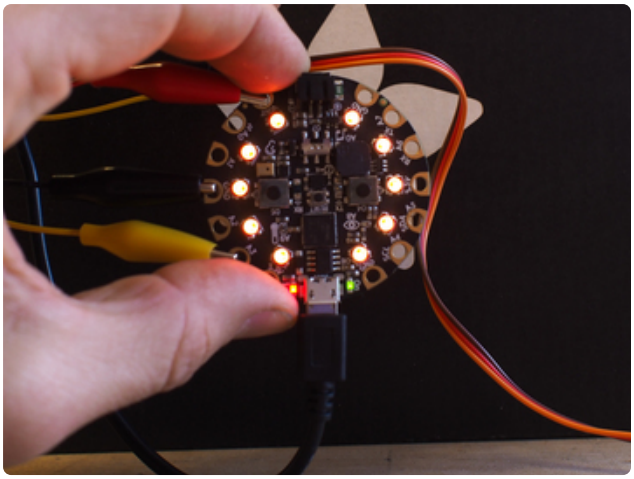


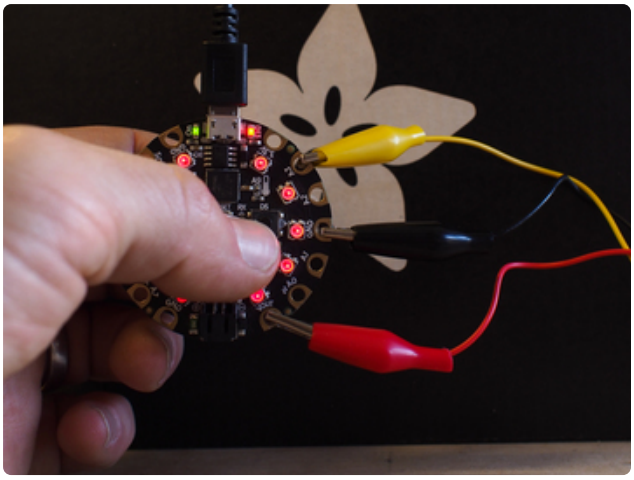


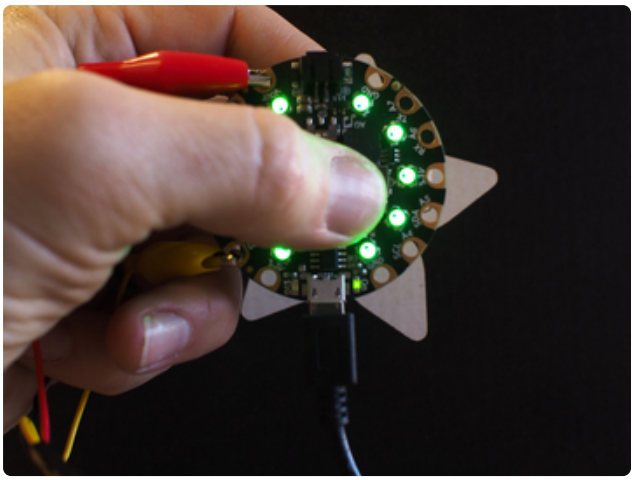


First, the servo should go to the locked position. Now, it awaits your combination.

Turn to the pink quadrant, then press the **A** button
Turn to the white quadrant, then press the **A** button
Finally, turn to the orange quadrant and press the **A** button







It beeps, opens the lock, and turns the NeoPixels green. Success!

If you enter the incorrect combination, you'll get a sadder sounding beep, and the lights turn red. Try again!

Also note, you can press the **B** button to reset and try from the start at any time. The **B** button also re-locks the servo after it's been unlocked. Try it now!

The Combo Dial Box Code in Detail

Let's dive into the code now!

Library Import

First, we'll import a few libraries that give us access to advanced functions we'll need that don't exist in the base CircuitPython.

These code snippets aren't meant to be run on their own, they serve to explain the full program already listed above

```
import board
import pulseio
from adafruit_motor import servo
from adafruit_circuitplayground.express import cpx
```

Next, we'll create a PWMOut object on Pin A3:

```
pwm = pulseio.PWMOut(board.A3, duty_cycle=2 ** 15, frequency=50)
```

and create a servo object:

```
servo = servo.Servo(pwm)
```

We'll also set the NeoPixels to a 0.05 brightness.

```
pwm = pulseio.PWMOut(board.A3, duty_cycle=2 ** 15, frequency=50)

#  plug red servo wire to VOUT, brown to GND, yellow to A3
servo = servo.Servo(pwm)

cpx.pixels.brightness = 0.05  # set brightness value
```

Servo Locking & Unlocking Helpers

For clarity later on in our code, we'll define a couple of functions for the servo positions:

```
def unlock_servo():
    servo.angle = 180
```

```
def lock_servo():
    servo.angle = 90
```

This is helpful, because the angle that is the "locked" and "unlocked" position is relative to the servo position, so it isn't always apparent which is which. By using clear names like `unlock_servo` and `lock_servo`, the code is easier to understand.

Combination Storage

Next, we'll set variable to store the `correct_combo` , and another to hold the attempted combinations when in use, the `entered_combo`. We'll be able to compare those two variables to each other to see whether or not the safe should unlock after the final position is entered.

We also will make a variable called `current_dial_position` to store the current dial position based upon the accelerometer reading before it is entered into the `entered_combo` list. It starts off with a dummy value of 'X' which is not a position on the dial.

```
correct_combo = ['B', 'D', 'C']  # this is where to set the combo
entered_combo = []  # this will be used to store attempts
current_dial_position = 'X'
```

Finish Setting Up

Final stage of setup is to turn on the on-board red LED, and lock the servo:

```
cpx.red_led = 1  # turn off the on-board red LED while locked
lock_servo()  # lock the servo
```

The Main Loop

Now, we get to the main loop that will run over and over again. We'll set up a set of variables that read the built-in accelerometer and convert the values to easy-to-use integers.

```
    x_float, y_float, z_float = cpx.acceleration  # read acceleromter
    x = int(x_float)  # make int of it
    y = int(y_float)
    z = int(z_float)
```

Here's a visual representation of how we'll break the board's rotation up into four quadrants:

```
    # four simple rotation positions, A-D
    # the combination entries are based on which letter is facing up
    #
    #              A
    #            .___.
    #         .         .
    #     D  .           .  B
    #        .           .
    #         .         .
    #            .|_|.
    #              C
```

To determine the orientation, we'll compare the x and y axis values from the `cpx.acceleration` query against these four conditions:

```
if x == 0 and y == 9:
        current_dial_position = 'A'  # used to store dial position
        cpx.pixels.fill((0, 0, 255))

    if x == 9 and y == 0:
        current_dial_position = 'B'
        cpx.pixels.fill((80, 0, 80))

    if x == 0 and y == -9:
        current_dial_position = 'C'
        cpx.pixels.fill((255, 70, 0))

    if x == -9 and y == 0:
        current_dial_position = 'D'
        cpx.pixels.fill((255, 255, 255))
```

Since gravity is -9.8 m/s2, a value reading of 9 on the y-axis means that the A quadrant of the board is facing "up" away from the gravitational pull of our planet Earth.

Using this method, we can determine the orientation of the board as it rotates around its local z-axis.

Button Time!

Next, we'll set up the code for reacting to button presses. When the **B** button is pressed, we'll lock the servo and reset the board to the clear state, ready to accept a new combination attempt.

```
# press the right/B button to lock the servo
    if cpx.button_b:  # this is a more Pythonic way to check button status
        print('Locked/Reset')
        cpx.red_led = 1
        cpx.pixels.fill((50, 10, 10))
        lock_servo()
        cpx.play_tone(120, 0.4)
        cpx.pixels.fill((0, 0, 0))
        entered_combo = []  # clear this for next time around
```

```
        time.sleep(1)
```

That bit of code queries the button, and if the result comes back True, meaning it's been pressed, it will:

- turn on the on-board red LED
- turn all of the NeoPixels red
- lock the servo
- play a sad beeping sound
- turn off the NeoPixels
- clear the enteredCombo list
- pause for a second

Next, we'll set up button **A** so that we can enter combination attempts.

```
# press the left/A button to enter the current position as a combo entry
    if cpx.button_a is True:  # this means the button has been pressed
        # grab the current_dial_position value and add to the list
        enteredCombo.append(current_dial_position)
        dial_msg = 'Dial Position: ' + str(entered_combo[(len(entered_combo)-1)])
        print(dial_msg)
        cpx.play_tone(320, 0.3)  # beep
        time.sleep(1)  # slow down button checks
```

Again, the button is queried, and if it returns True, the code below will run.

First, it will append the `current_dial_position` to the `entered_combo` list, based on the current orientation of the board at the time the button is pressed.

If this is the first time pressing **A** since the list was cleared, this value will go into the first position in the list. If it's the second time, it goes to the second position, and so on.

We include some print statements so that you can look at text feedback in the REPL if you like. These are not necessary for the program to run, just informational.

Then, we play a beep to indicate that the attempt has been entered, and finally, take a short pause.

Did We Dial Right?

The next bit of code is used to determine when the third entry has been made in the `entered_combo` list, and then test the result against the hard coded `correct_combo` list.

```
if len(entered_combo) == 3:
        if entered_combo == correct_combo:  # they match!
            print('Correct! Unlocked.')
            cpx.red_led = 0  # turn off the on board LED
            cpx.pixels.fill((0, 255, 0))
            unlock_servo()
            cpx.play_tone(440, 1)
            time.sleep(3)
            entered_combo = []  # clear this for next time around
```

First, the length of the list is checked, and when it reaches three, the code below is executed.

Got it Right!

Then, the two lists are compared. If the combination is correct:

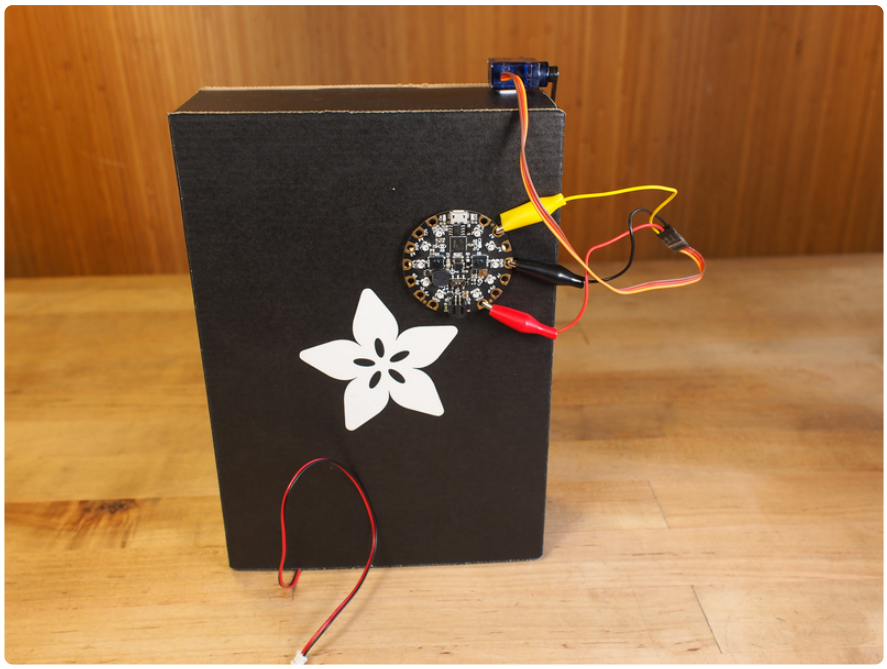
- a victorious message is printed
- the red LED turns off
- all the NeoPixels fill with a happy green
- the servo is unlocked
- a friendly beep is played
- there's a short pause
- the `entered_combo` list is cleared for next time

Try Again!

The last bit of code deals with the case where the two lists do NOT match. This means the incorrect combination was entered. So:

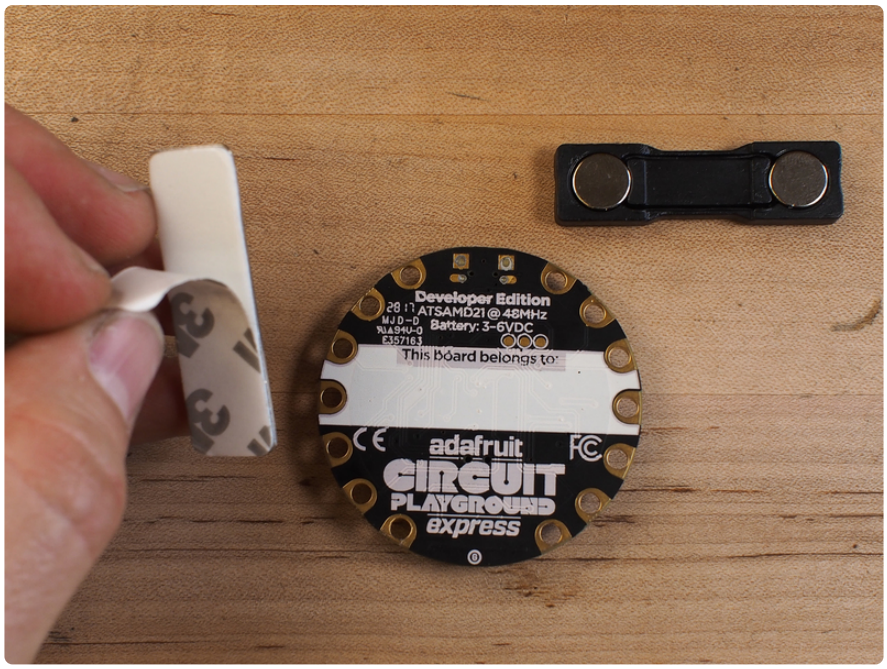
- The 'Incorrect combination.' message is printed
- All NeoPixels fill with angry red
- A disapproving pair of tones plays
- there's a short pause
- the `entered_combo` list is cleared for next time

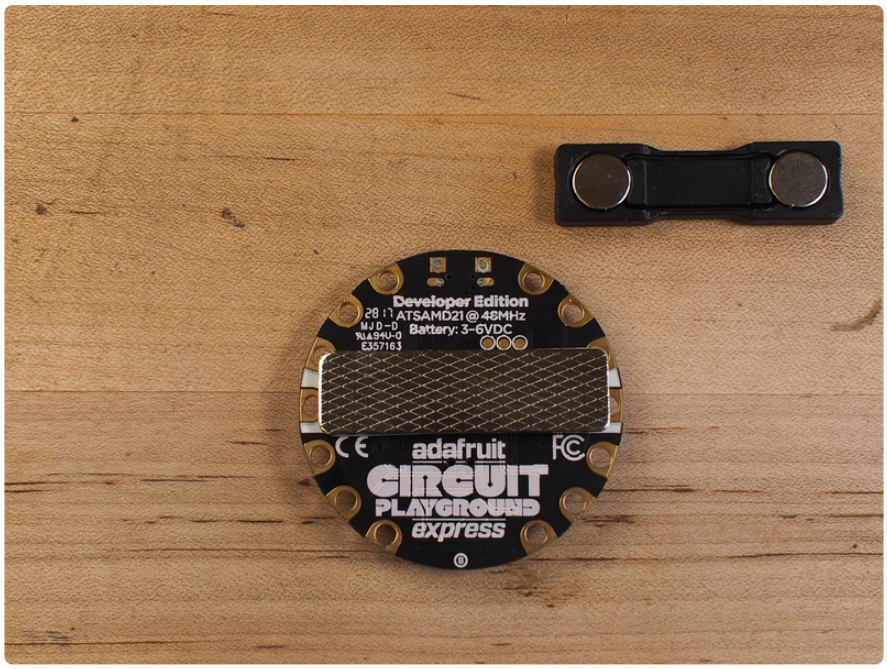
Make the Safe



Now that we've got the board coded with CircuitPython, you can mount the servo motor to act as a lock inside the box. You'll also mount the Circuit Playground Express to the box, but with magnets so it can turn freely and rotate to the combination, so you can turn it like a dial to enter the right combination and unlock the safe!

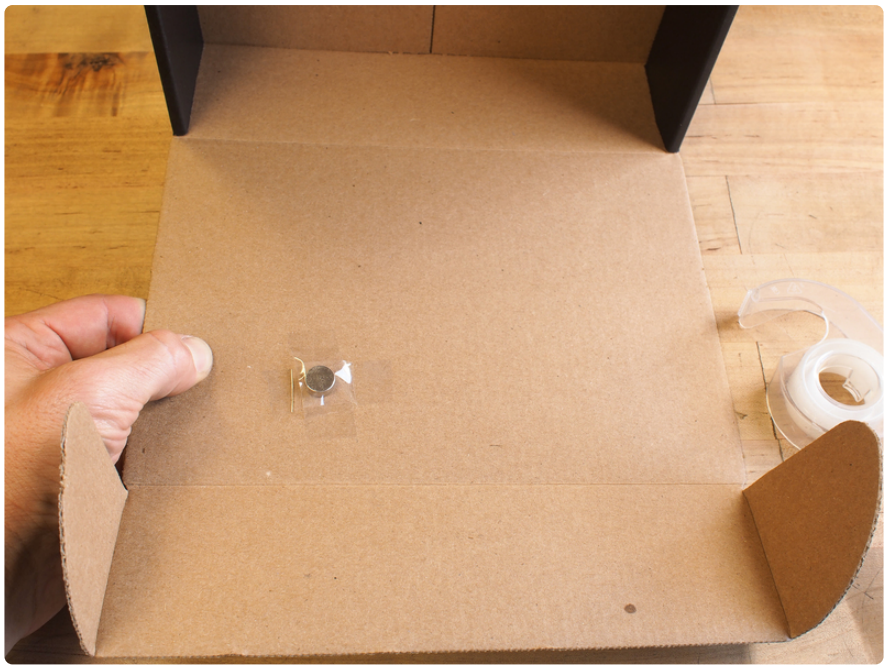
To build the safe, first prepare the combination dial by affixing the metal pin back to the back of the Circuit Playground Express using the foam tape.





Now, place the Circuit Playground Express on the door (formerly the lid of the box) where you'd like the dial to be, and then place the magnet on the inside of the door so it sits at the center of the dial.

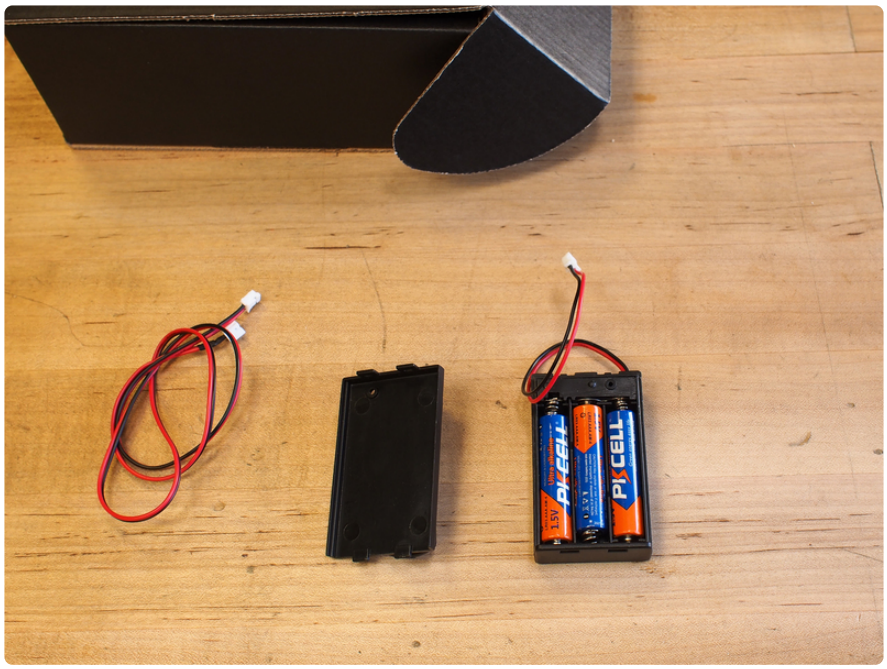
Use two pieces of tape to hold it down and prevent it from falling off.

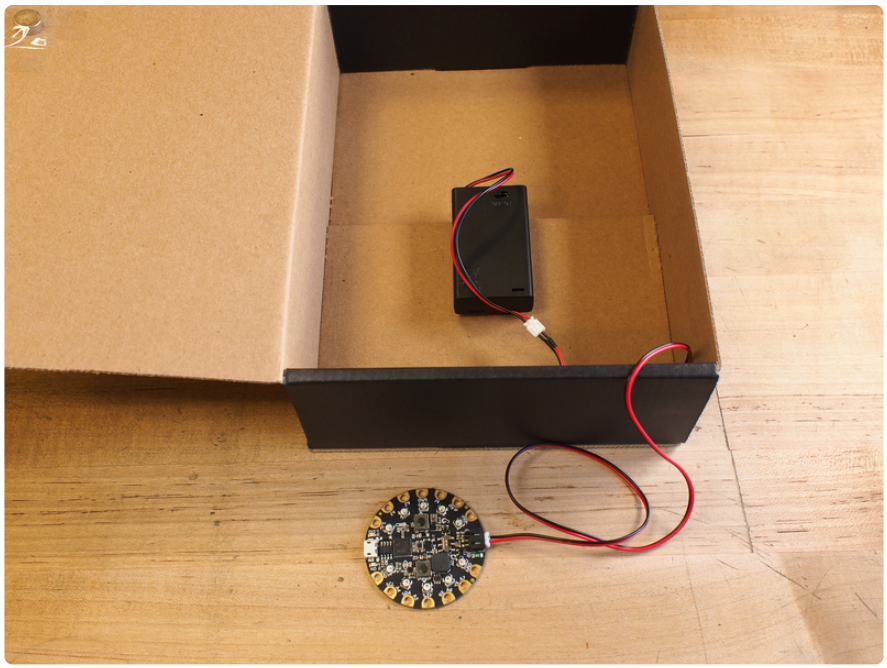


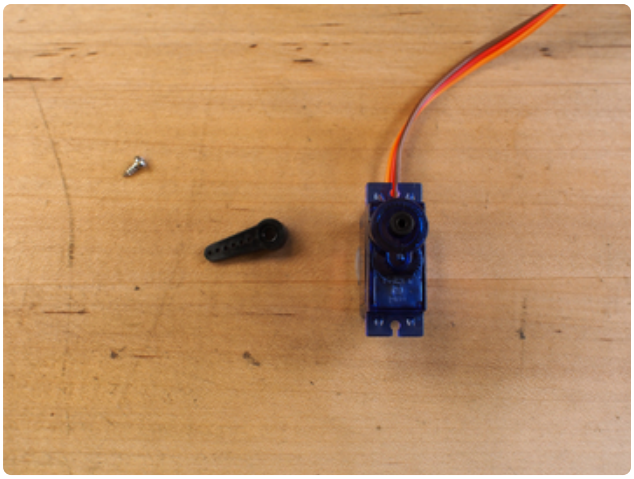
Put batteries in the case.

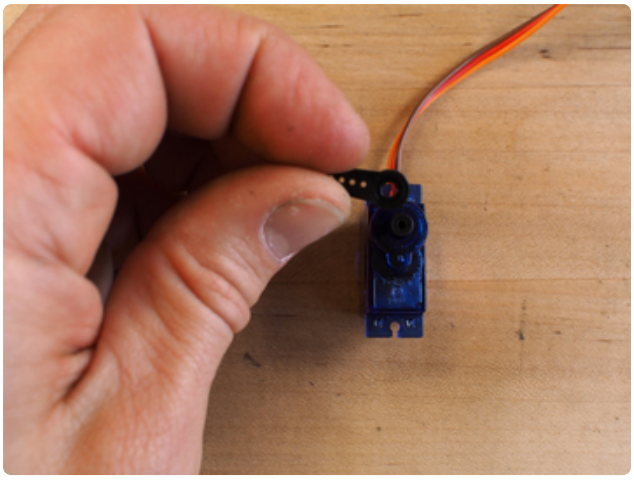
Use the JST extension cable to extend the battery pack, then plug it into the Circuit Playground Express.

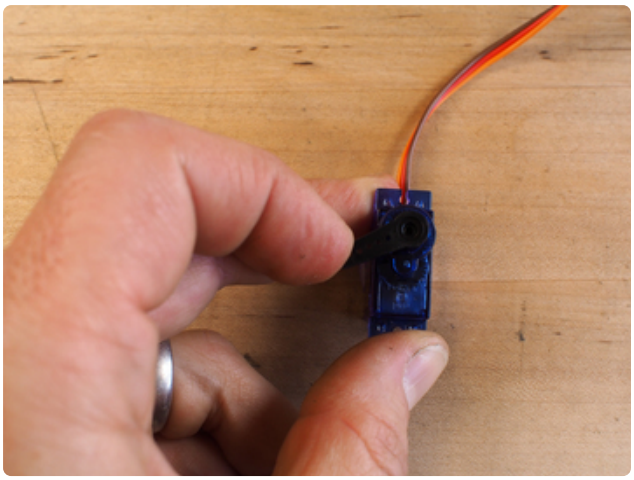
Hook the battery pack's clip to the flap on the wall at the back of the safe, leaving plenty of slack for the dial to turn.







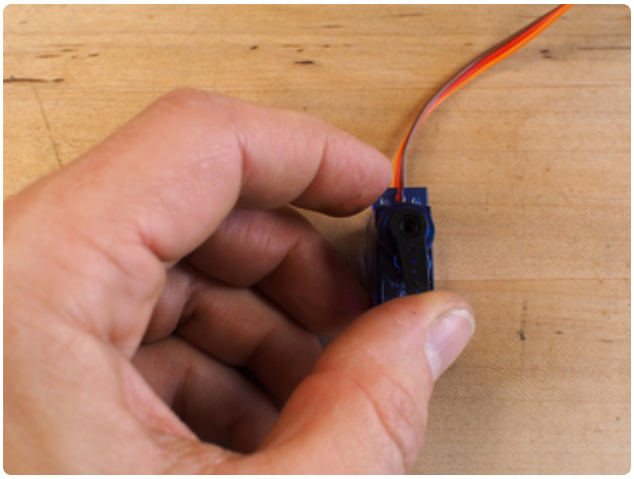


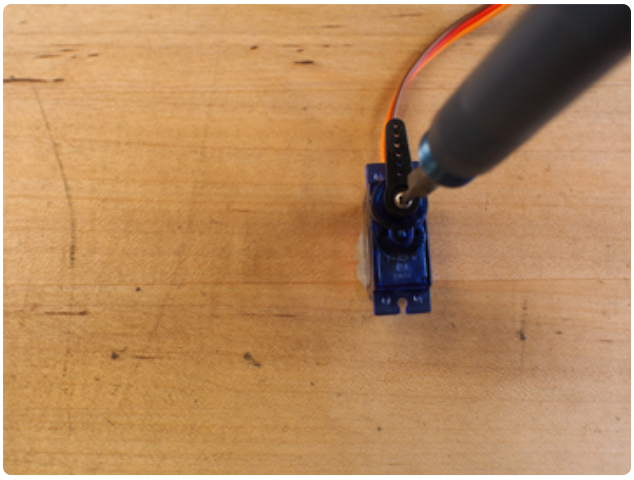


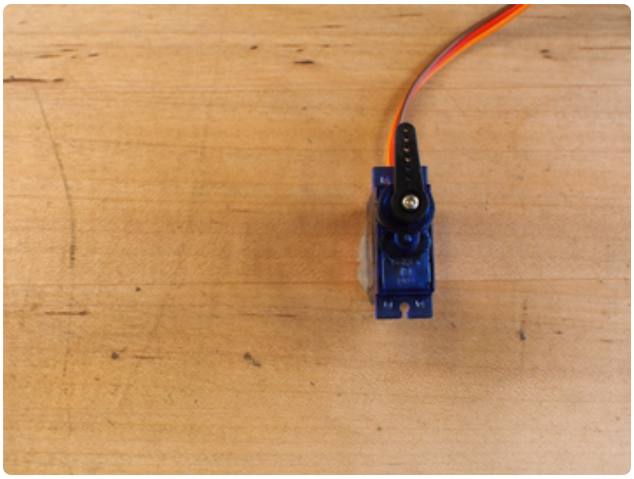
Use the small servo arm like a wrench to turn the servo's shaft until it is at it's farthest counter-clockwise position when looking down at the shaft oriented as shown.

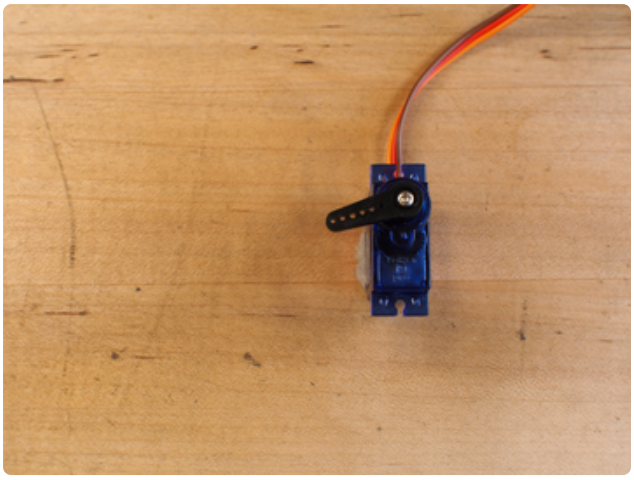
Remove and re-position the servo arm so that it points left at the 9 o'clock position as shown. It should be free to rotate to the 1 o'clock position, traveling counter-clockwise past 6 o'clock.

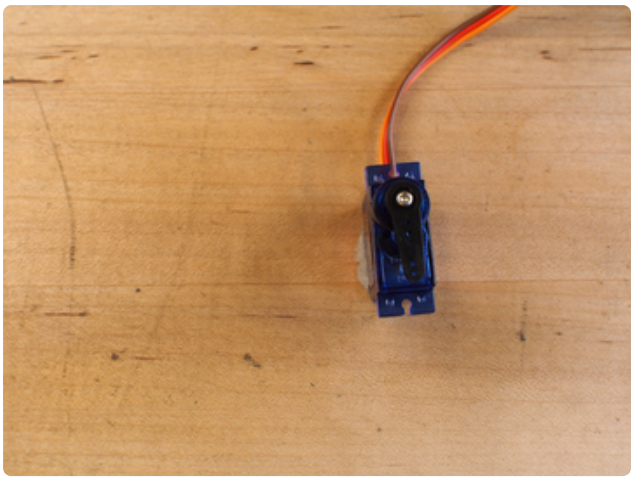
Screw in the small retaining screw into the center of the servo shaft to secure the arm.





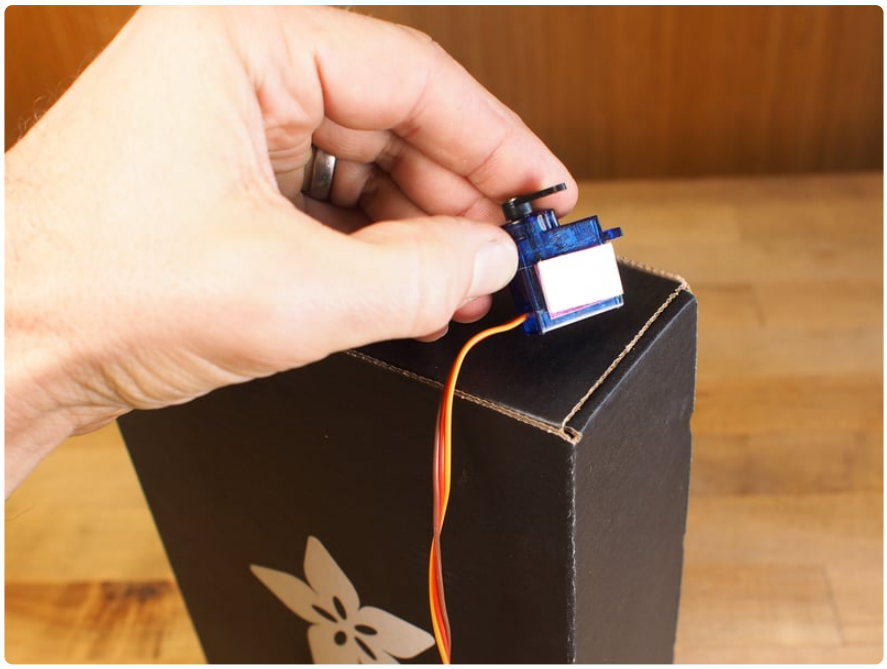


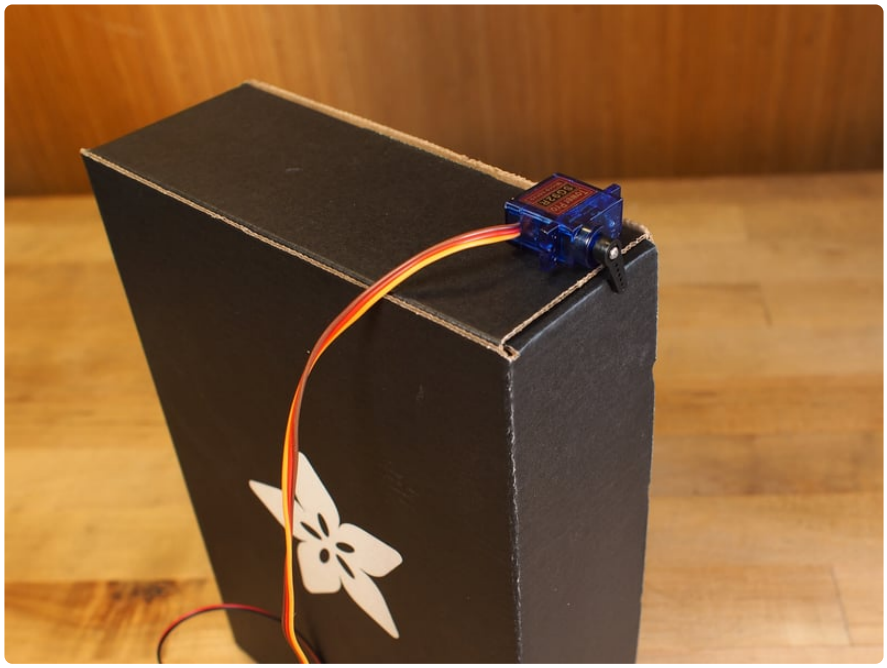




This is what the full range of motion should look like.

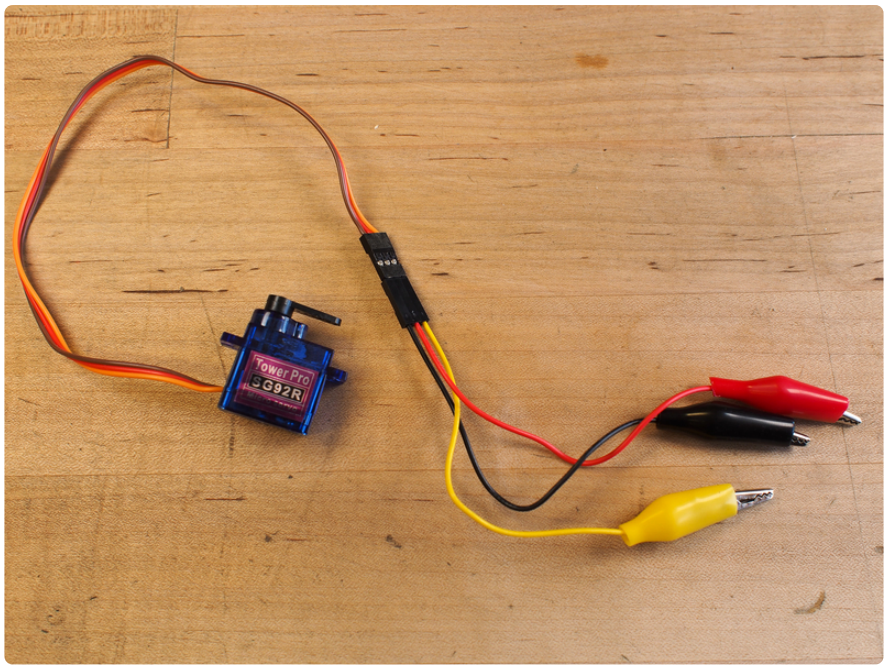
Now, we can create the "lock". Use a small piece of double stick foam tape to affix the servo on top of the box so that the servo arm blocks the lid from opening as shown here.



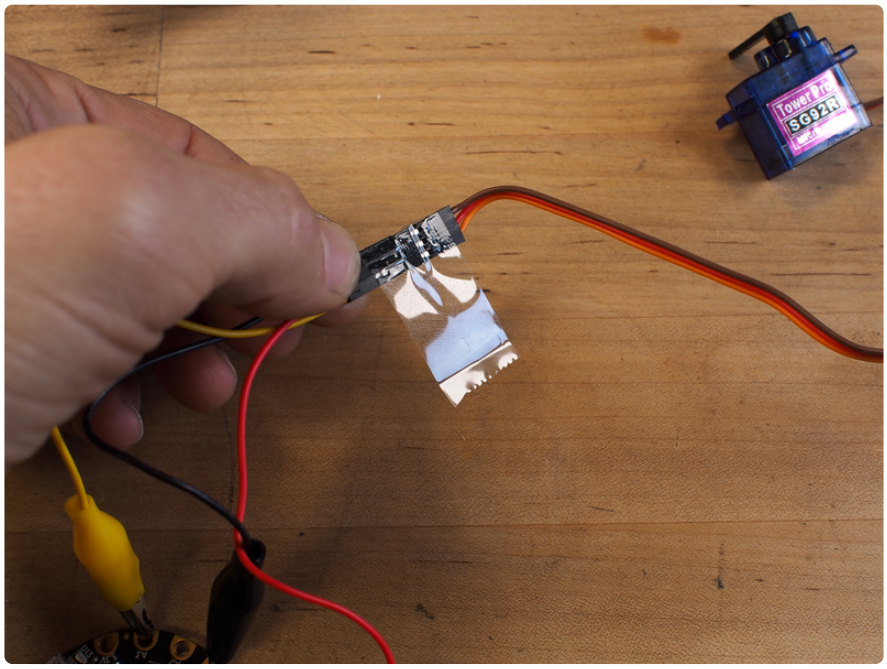


In case you disconnected them after coding, re-connect the alligator clips back to the Circuit Playground Express as before:

- Red goes from **VOUT** to servo red voltage wire
- Black goes from **GND** to servo brown ground wire
- Yellow goes from **A3** to servo orange signal wire

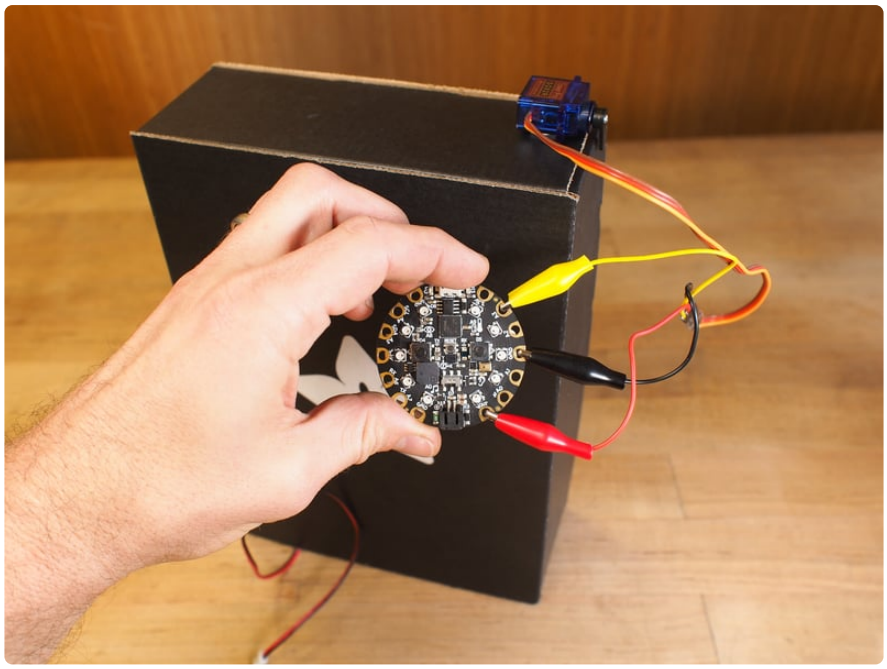


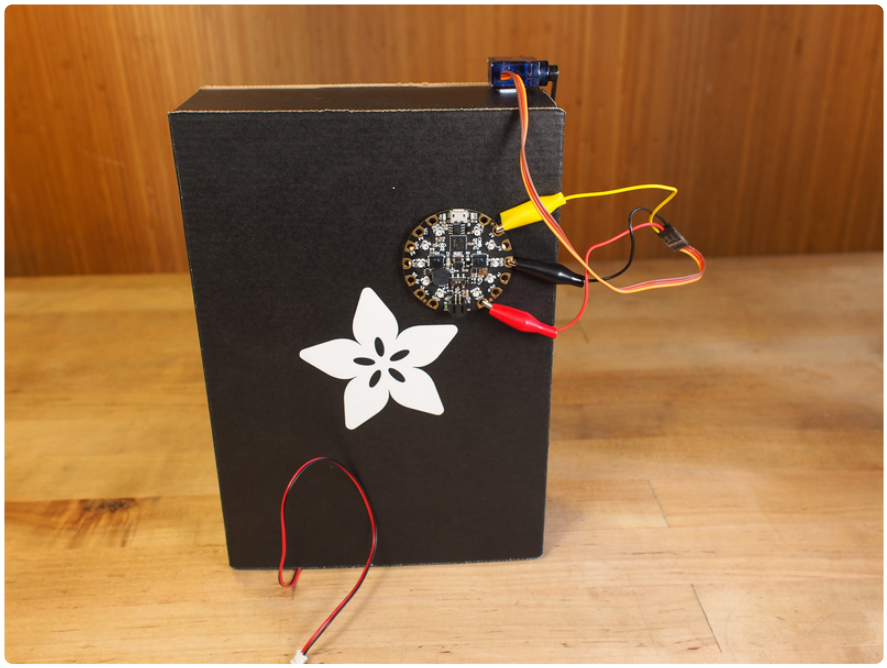
You can use a small piece of tape to secure the alligator leads to the servo connector.



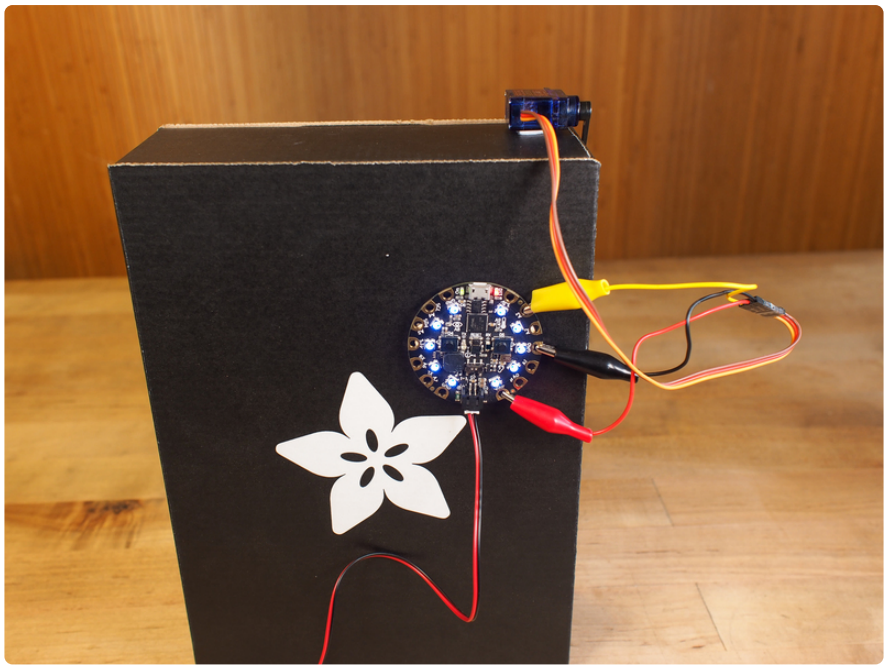
Turn on the battery pack, but leave the Circuit Playground Express unplugged, then close the door.

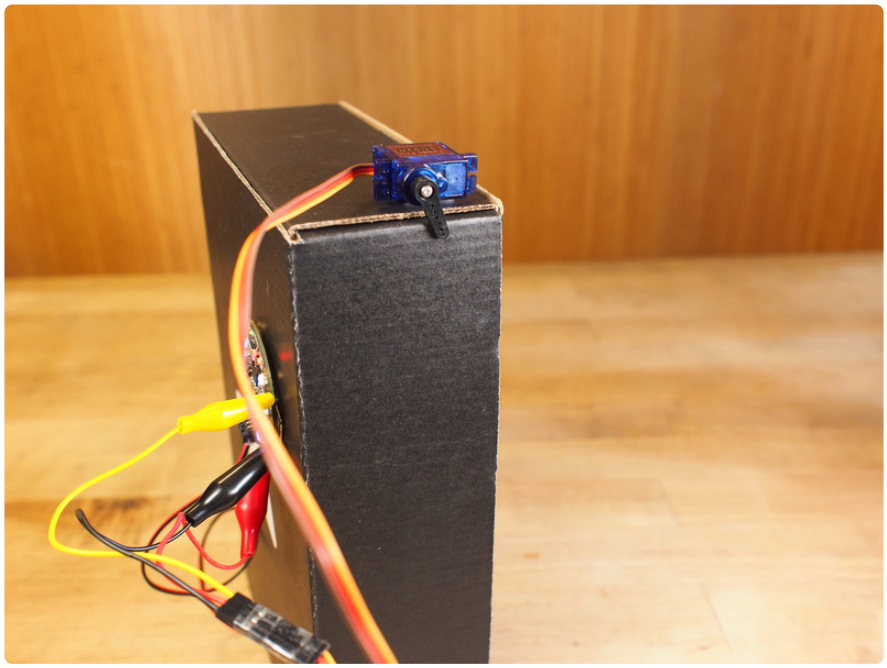
Now, place the "dial" over the chosen spot -- the magnet on the other side will hold it nicely in place.





Time to "arm" the safe! Plug in the JST connector. The system will boot up, and the servo arm will lock down into place.

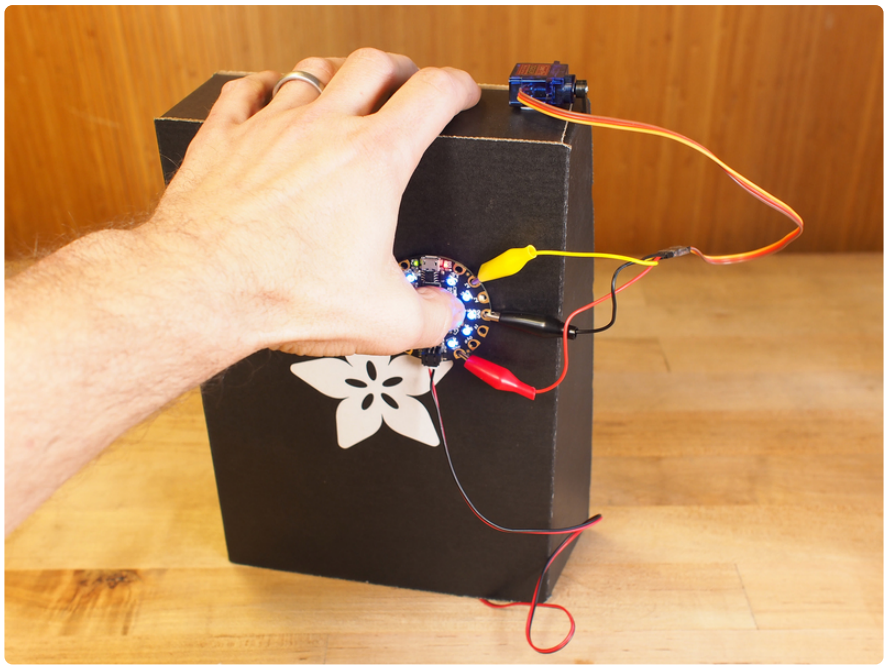




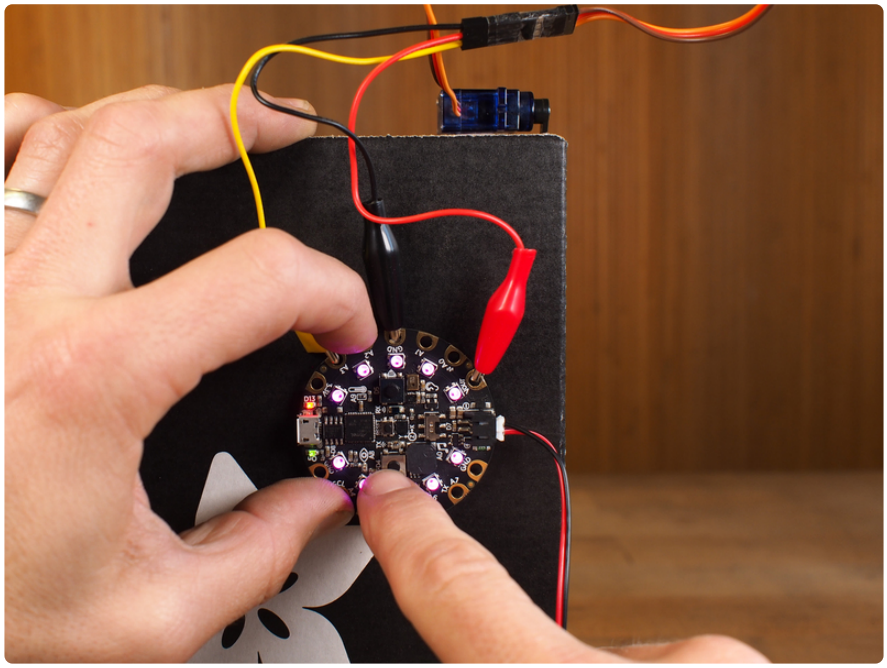
Now, it's time to try out your safe!

Open the Safe

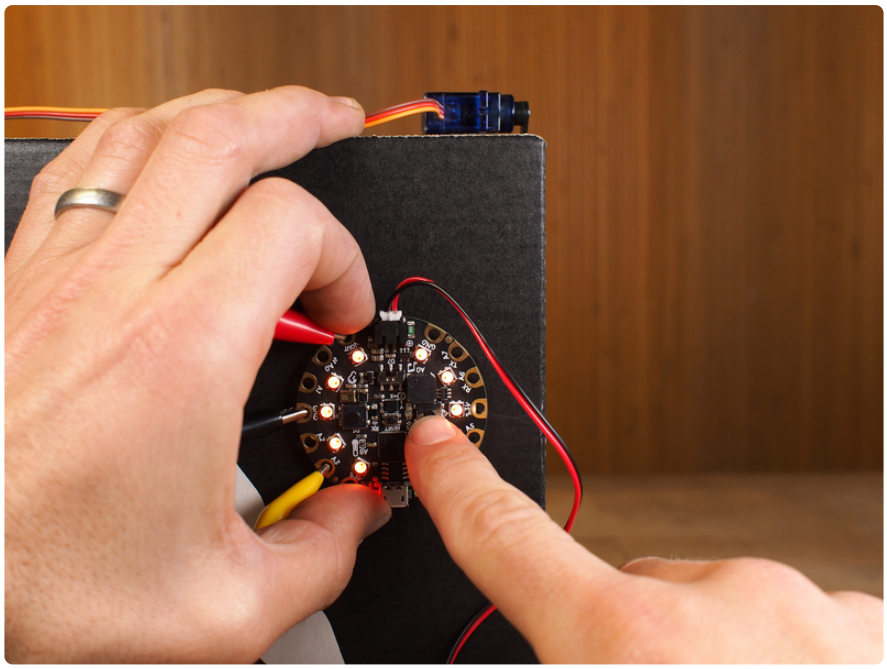
Make sure the safe is locked by pressing button B, the one on the right.



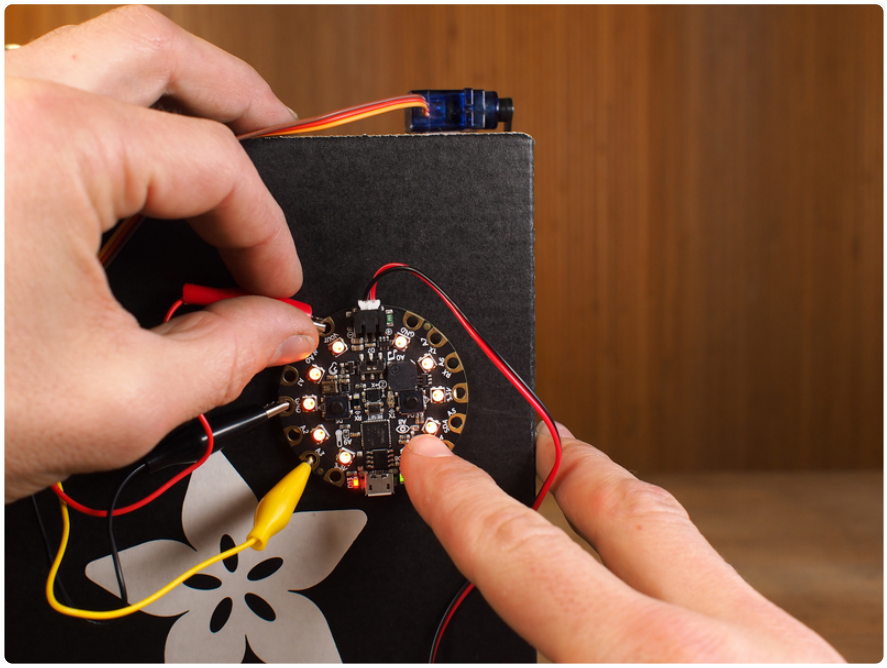
Turn the dial -- the Circuit Playground Express -- to the first position, in the case of our default code, this is to the left so the NeoPixels glow pink. Then press button A, the one on the left.

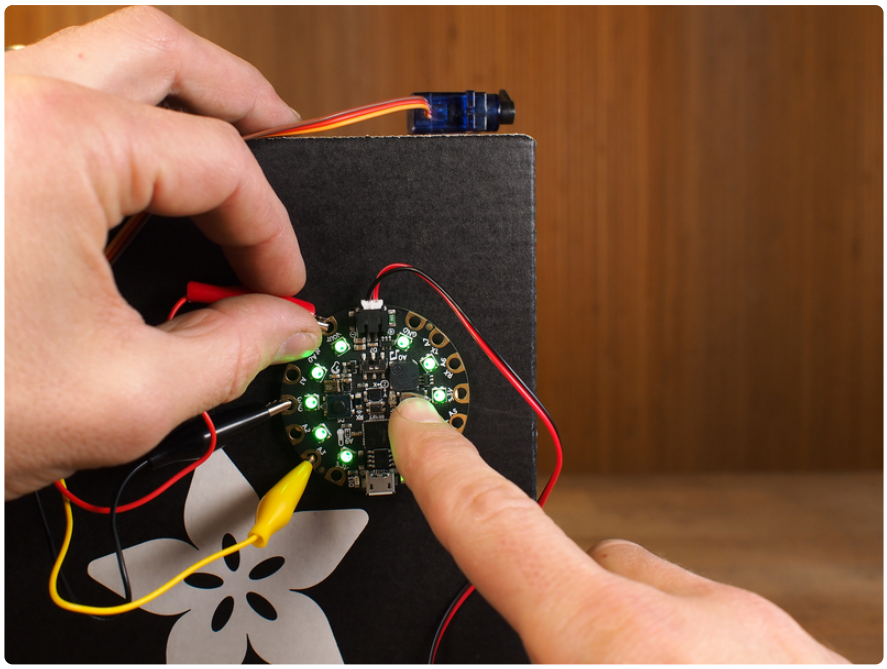


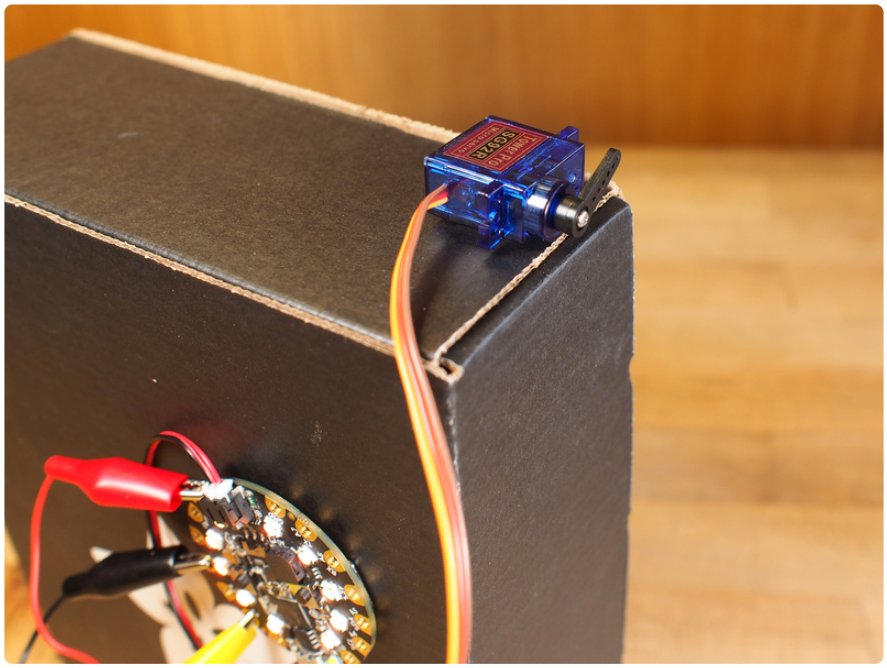
Now, turn it so the lights are white, and press the A button.



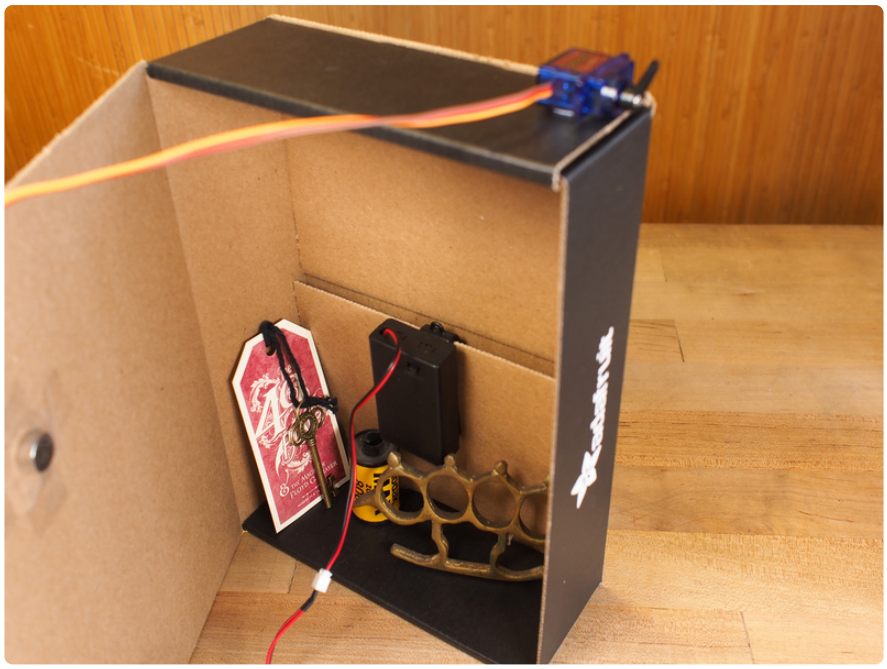
Finally, the moment we've been waiting for! -- turn the dial so it lights up orange and press the A button one last time. The lights will turn green -- success! -- and the lock will open. You're in!



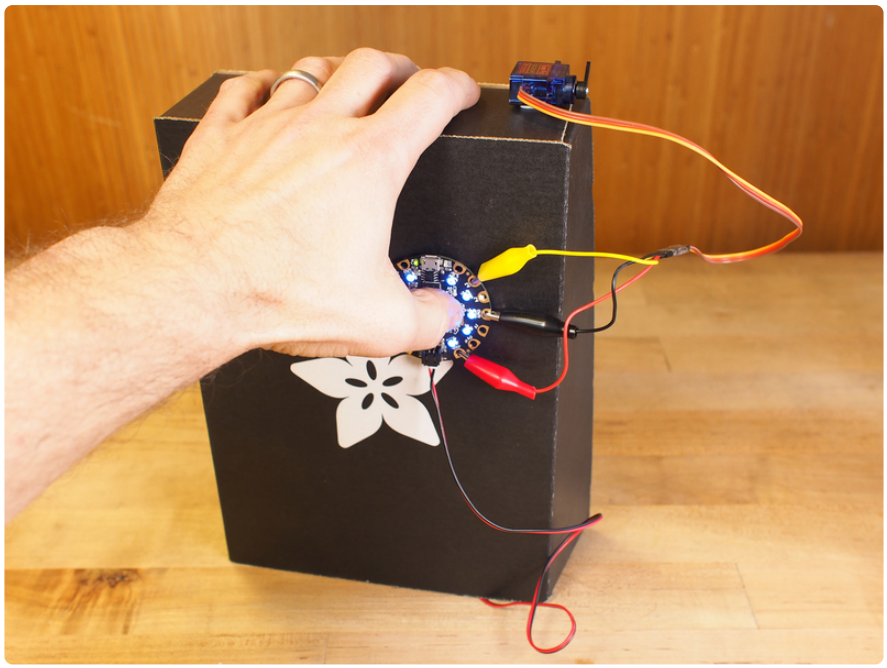




Swing open your safe door, and now you can hide another valuable treasure within. Such as an undeveloped roll of 35mm film from the 1990's...



Close the door, press the B button and it will lock again.



Safe.