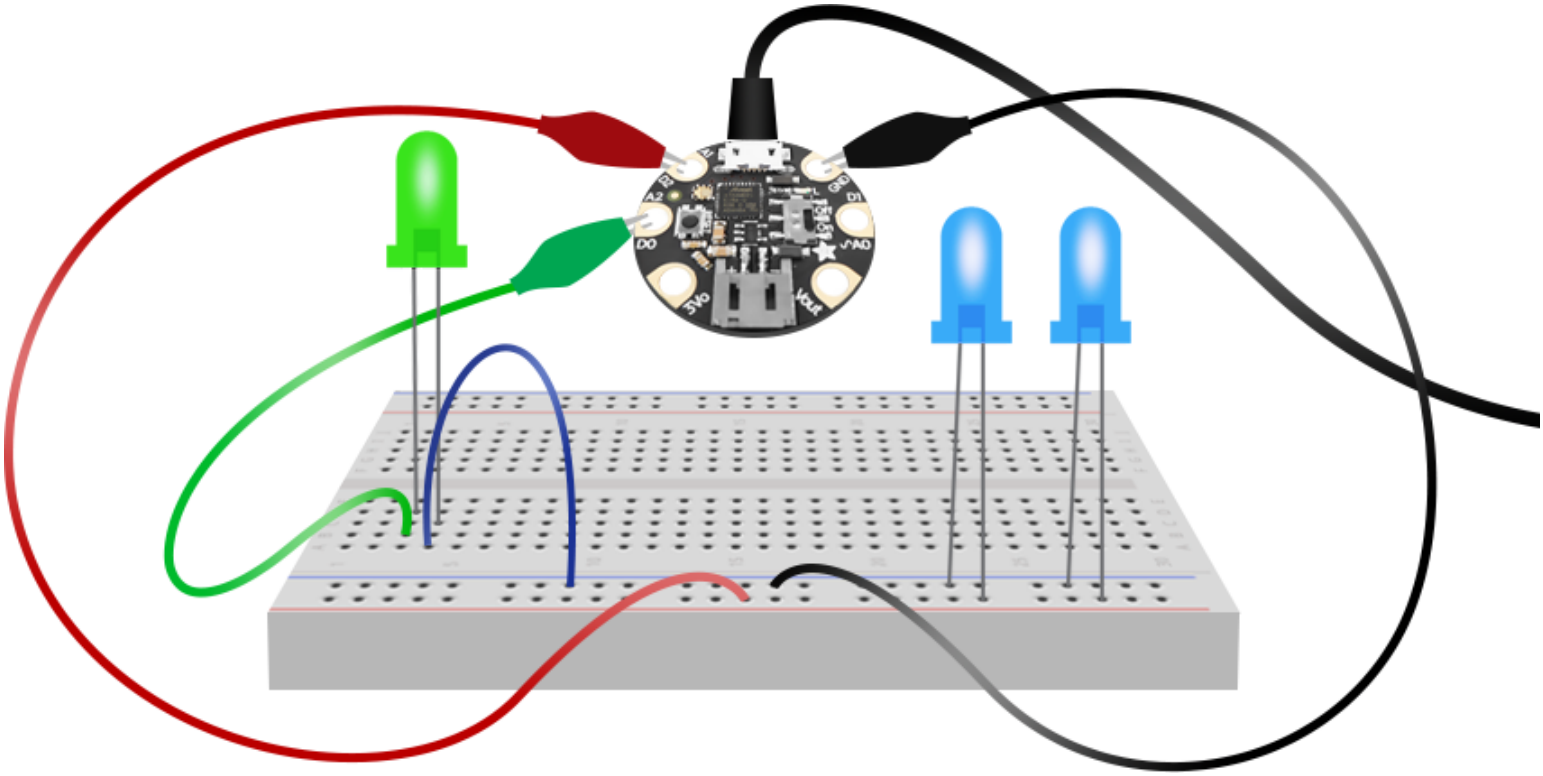
Smart Circuits: Lights On!

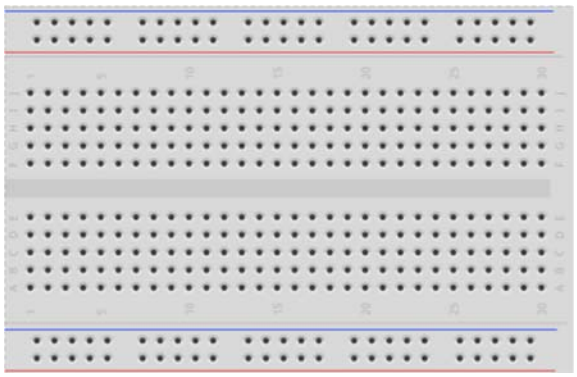


This material is based upon work supported by the Department of Energy and the Department of Homeland Security

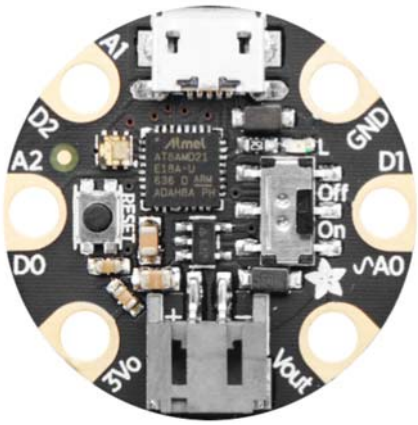
MATERIALS NEEDED

JST connector for use with the Gemma

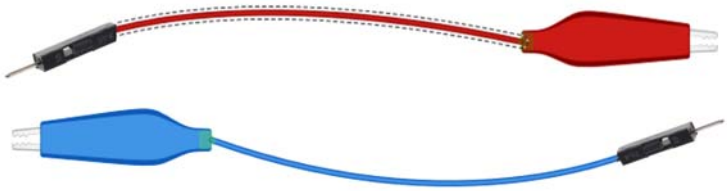
Breadboard



Gemma Mo



Alligator to jumper



Jumper wires

Alligator to alligator

This material is based upon work supported by the Department of Energy and the Department of Homeland Security

MATERIALS NEEDED

Copper tape

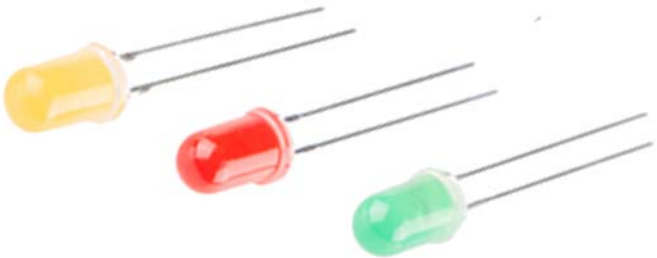
Photo sensor

10 KΩ resistor

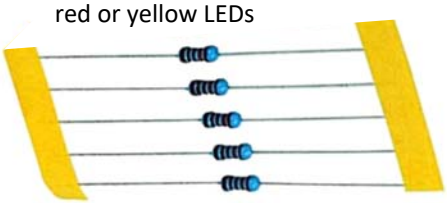
Piezo buzzer

USB to micro USB data cable

Multiple LEDs



200Ω resistors for use with red or yellow LEDs



This material is based upon work supported by the Department of Energy and the Department of Homeland Security

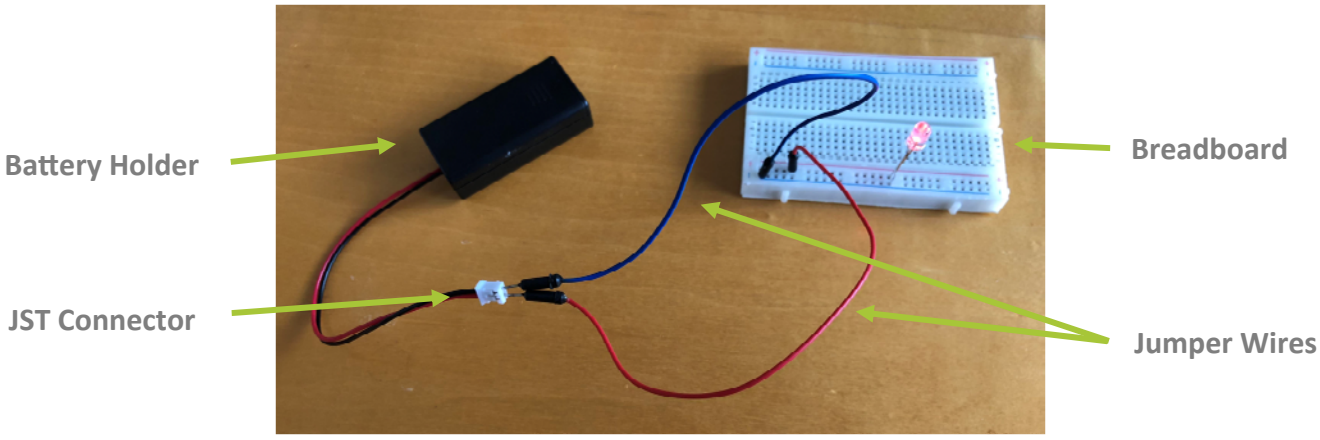
First LED Circuits

PART 1 - BUILD A CIRCUIT

Insert the AA batteries into the battery holder.

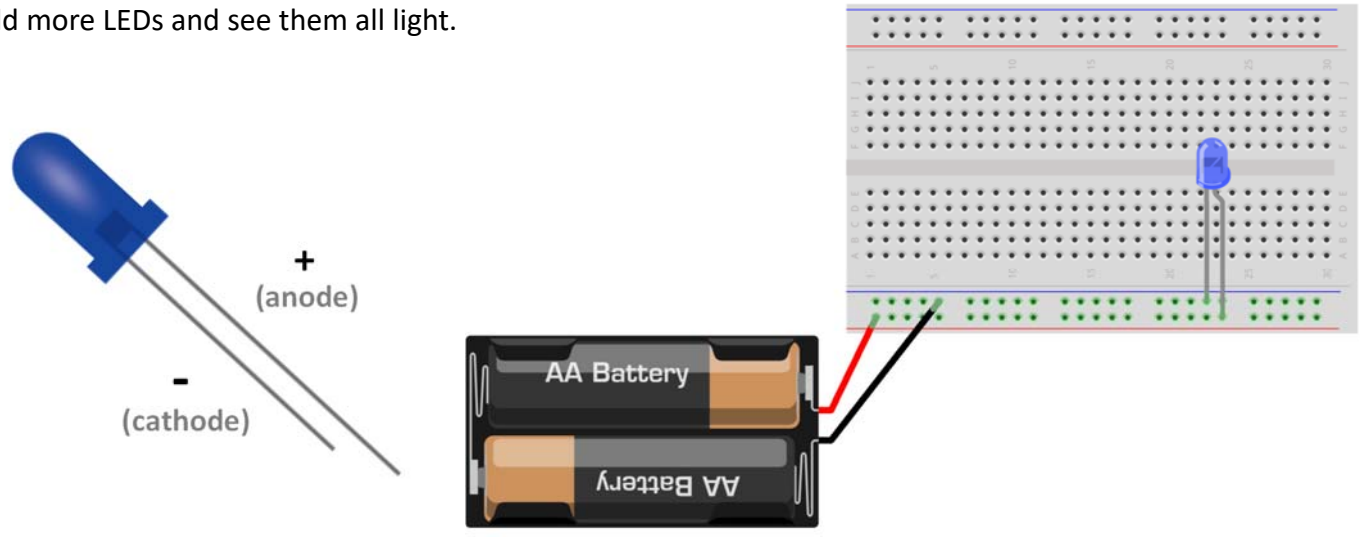
The wires from the battery holder have a white JST connector on the end. We'll need that connector for the Gemma later.

Use the jumper wires to connect the battery holder to ties on the outside rails of the breadboard. Connect red to red and dark to dark.



Connect a blue LED by pushing the anode (longer leg) into any tie on the red (+) strip and the other leg (the cathode) into the blue (-) strip. If all your connections are tight, you should see the LED light.

Add more LEDs and see them all light.



This material is based upon work supported by the Department of Energy and the Department of Homeland Security

PART 2 - ADD SOME CODE

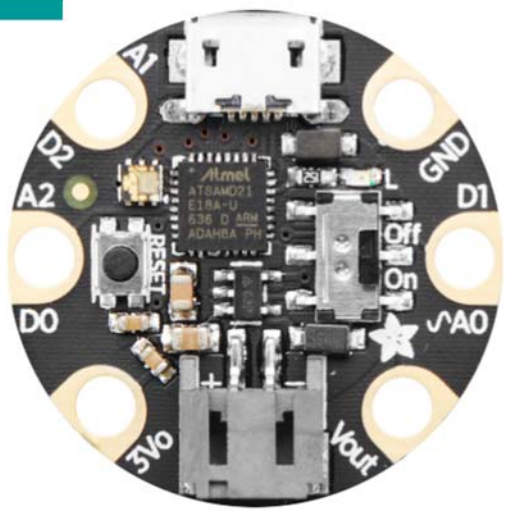
Introducing Gemma Mo

Gemma is a small microcontroller developed by Adafruit that comes with Circuit Python on board.

To write Python code for the Gemma M0, first install the Mu editor. Go to https://codewith.mu/ and download the installer for your computer.

Connect your Gemma to your computer using the USB to micro USB data cable. Your computer will see it as CIRCUITPY disk drive. While the Gemma is connected it gets power from the computer.

Mu automatically attempts to detect your Gemma and open Python for you to start editing right away. The first time you launch the Mu editor you may need to choose the Adafruit CircuitPython mode.

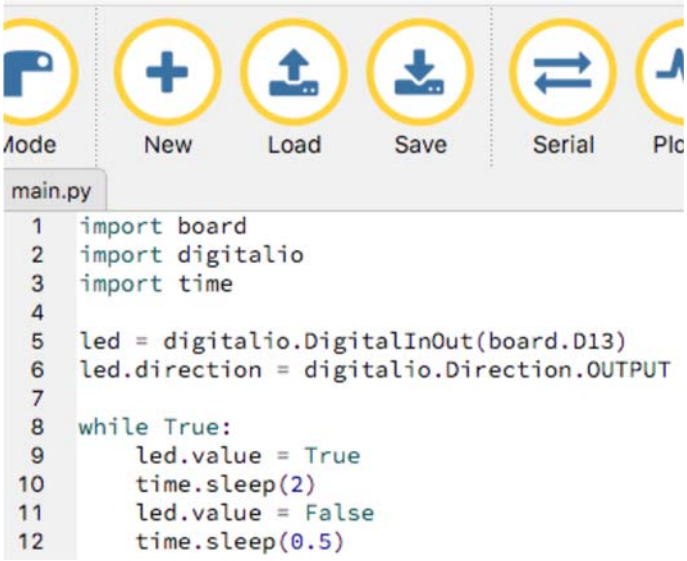


ALWAYS eject the CIRCUITPY disk before unplugging it or turning Gemma off!

It's ok to turn Gemma off after you eject the CIRCUITPY disk.

Create a Program to Control LEDs

To begin a new program in Mu click the **New** button in the top menu bar. Then type the code below into the editor and save the code to the CIRCUITPY disk as **main.py**. If Mu says main.py already exists, choose to replace it.




```
1  import board
2  import digitalio
3  import time
4
5  led = digitalio.DigitalInOut(board.D13)
6  led.direction = digitalio.Direction.OUTPUT
7
8  while True:
9      led.value = True
10     time.sleep(2)
11     led.value = False
12     time.sleep(0.5)
```

READ THE CODE

The first three lines import libraries. The board library gives access to the physical pins of the microcontroller. The digital input output library allows control of digital pins on the Gemma, and the time library allows control of how long the LED is on or off.

Lines 5 and 6 send data to the red LED on the Gemma that is connected to pin D13.

The while True loop turns the red LED on and then off. When **led.value = True** the LED is on and stays on for 2.0 seconds. Then **led.value = False** turns the LED off for 0.5 seconds.

Make changes to Your Program

Edit the **time.sleep** values to make the LED turn off and on more slowly or more quickly. Make the changes in the Mu editor and then click **Save**. How does the blinking LED change?

This material is based upon work supported by the Department of Energy and the Department of Homeland Security

First LED Circuits

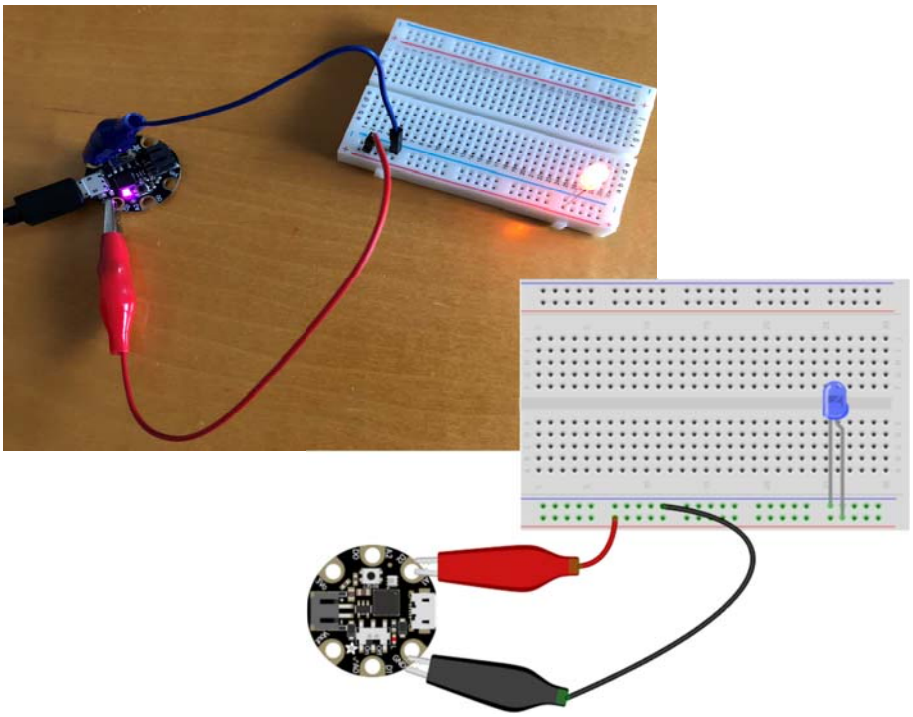
More Changes to Your Program

Use the data cable to connect the Gemma to your computer and open Mu. Mu automatically detects and shows your **main.py** program. You can also load the program from the CIRCUITPY disk.

Edit line 5 of your **main.py** code to change the output to pin D2 and save. The red LED stops blinking because the code sent the output to pin D2 instead of pin D13.

main.py * main.py

```
1  import board
2  import digitalio
3  import time
4
5  led = digitalio.DigitalInOut(board.D2)
6  led.direction = digitalio.Direction.OUTPUT
7
8  while True:
9      led.value = True
10     time.sleep(2)
11     led.value = False
12     time.sleep(0.5)
```



Connect the Gemma to Your Breadboard Circuit

Keep the Gemma connected to your computer.

Use an alligator to jumper wire to attach the GND (ground) pin to any tie in the blue rail and another to attach the D2 pin to any tie in the red rail.

Connect an LED into the rails. Remember, the positive leg is longer.

What do you see?

Add more code

Edit lines 5 and 6 so that the variable name is ledlarge and then add two more lines, 8 and 9, of similar code using variable name ledsmall. In line 8 change the output to D13.

Add **ledsmall.value = True** and **ledsmall.value = False** to the **while True** loop. Save.

What happens now?

```
1  import board
2  import digitalio
3  import time
4
5  ledlarge = digitalio.DigitalInOut(board.D2)
6  ledlarge.direction = digitalio.Direction.OUTPUT
7
8  ledsmall = digitalio.DigitalInOut(board.D13)
9  ledsmall.direction = digitalio.Direction.OUTPUT
10
11 while True:
12     ledlarge.value = True
13     ledsmall.value = True
14     time.sleep(2)
15     ledlarge.value = False
16     ledsmall.value = False
17     time.sleep(0.5)
18
```

CHALLENGE :	Make changes to the code to make the LEDs blink at different rates.

This material is based upon work supported by the Department of Energy and the Department of Homeland Security

First LED Circuits

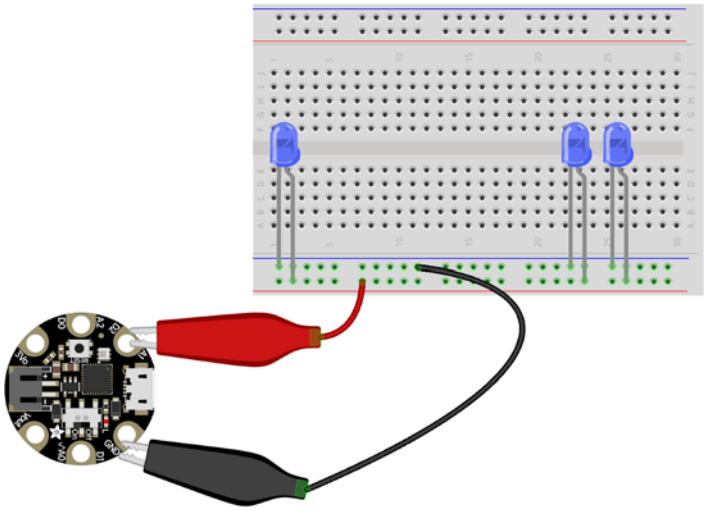
Add more LEDs

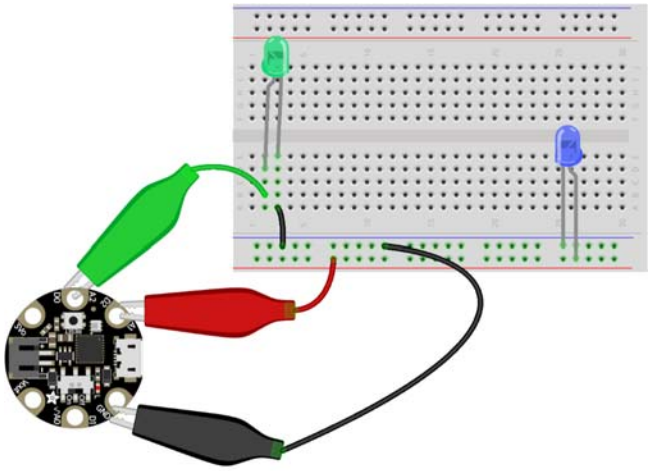
Without making changes to the code, add more LEDs to the breadboard rails.

Is the red D13 LED on the Gemma still blinking?

Which pin on the Gemma is communicating with the LEDs on the breadboard? Because the cathodes (positive legs) of these LEDs are all in the same breadboard rail, they all receive the same information from the Gemma and they all blink the same way.

Change the **time.sleep** values to change the blinking rates.





Add an LED to another area of the breadboard away from the rails.

In this diagram anodes (shorter negative legs) of both LEDs connect to the blue rail ground. A jumper connects the blue rail to the anode of the green LED.

The cathode (longer positive leg) of the blue LED in the red rail is still connected to pin D2.

The green alligator to jumper wire connects pin D0 to the cathode of the green LED.

The code at the right is similar to the **main.py** code from the previous activity that is already on your Gemma. We changed the variable names to ledblue and ledgreen to help remember which is which. You can name the variables however makes sense to you.

Change the output pin in line 8 to D0 and save. What happens?

```
1  import board
2  import digitalio
3  import time
4
5  ledblue = digitalio.DigitalInOut(board.D2)
6  ledblue.direction = digitalio.Direction.OUTPUT
7
8  ledgreen = digitalio.DigitalInOut(board.D0)
9  ledgreen.direction = digitalio.Direction.OUTPUT
10
11 while True:
12     ledblue.value = True
13     ledgreen.value = True
14     time.sleep(2)
15     ledblue.value = False
16     ledgreen.value = False
17     time.sleep(0.5)
```

CHALLENGE : Change the code in the **while True:** loop so that the blue LED is on when the green LED is off and the green LED is on when the blue LED is off.

Eject the CIRCUITPY disk, remove the data cable, and connect the battery to use your Gemma away from your computer. The main.py code remains on the Gemma .

This material is based upon work supported by the Department of Energy and the Department of Homeland Security

Copper Tape Circuits

Connect the Gemma to a Copper Tape Circuit

Attach 2 strips of copper tape to an index card. Copper tape is sticky on one side (peel off the white backing) and conductive on the other. Turn the end of the tape over to the other side of the card.

Tape the anode (longer leg) of an LED to the shiny side of one strip of copper tape and label this strip **+**. Tape the other leg (the cathode) to the shiny side of the other strip of copper tape and label this strip **-**.

Keep the Gemma connected to your computer, but remove the alligator to jumper wires and the breadboard. There may already be a main.py program on your Gemma that looks similar to code at the right. If not, save this code to your Gemma.

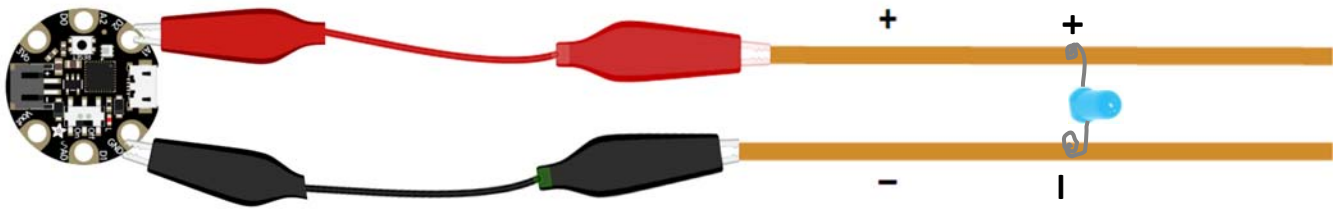
```
import board
import digitalio
import time

ledblue = digitalio.DigitalInOut(board.D2)
ledblue.direction = digitalio.Direction.OUTPUT

ledgreen = digitalio.DigitalInOut(board.D0)
ledgreen.direction = digitalio.Direction.OUTPUT

while True:
    ledblue.value = True
    ledgreen.value = True
    time.sleep(2)
    ledblue.value = False
    ledgreen.value = False
    time.sleep(0.5)
```

The lines of code that assign ledgreen to pin D0 don't do anything in this circuit because nothing is yet connected to D0.

Use an alligator to alligator wire to attach the Gemma GND (ground) pin to the negative copper tape strip and another to attach the D2 pin to the positive copper tape strip.

If all your connections are tight, you should see a blinking LED.

Add more LEDs and see them all light. What do you see?



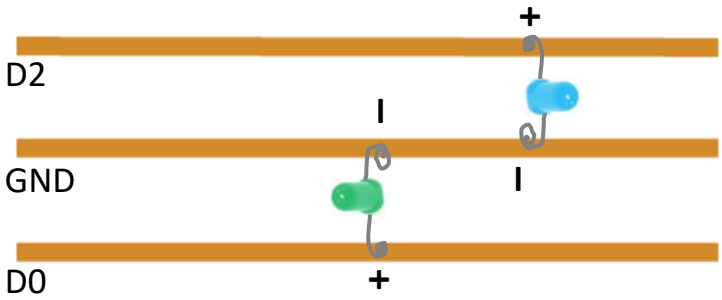
CHALLENGE :

Attach three alligator to alligator clips to three strips of copper tape with LEDs arranged like the diagram below. Attach the negative copper strip to ground on the Gemma and the other two positive strips to pins D0 and D2.

Now pin D0 is connected and lines 8 and 9 in the main.py code send output data to the green LED.

Change the code in the while True: loop so the two LEDs blink independently of each other.

Add more LEDs and see them all light. What do you see?



This material is based upon work supported by the Department of Energy and the Department of Homeland Security

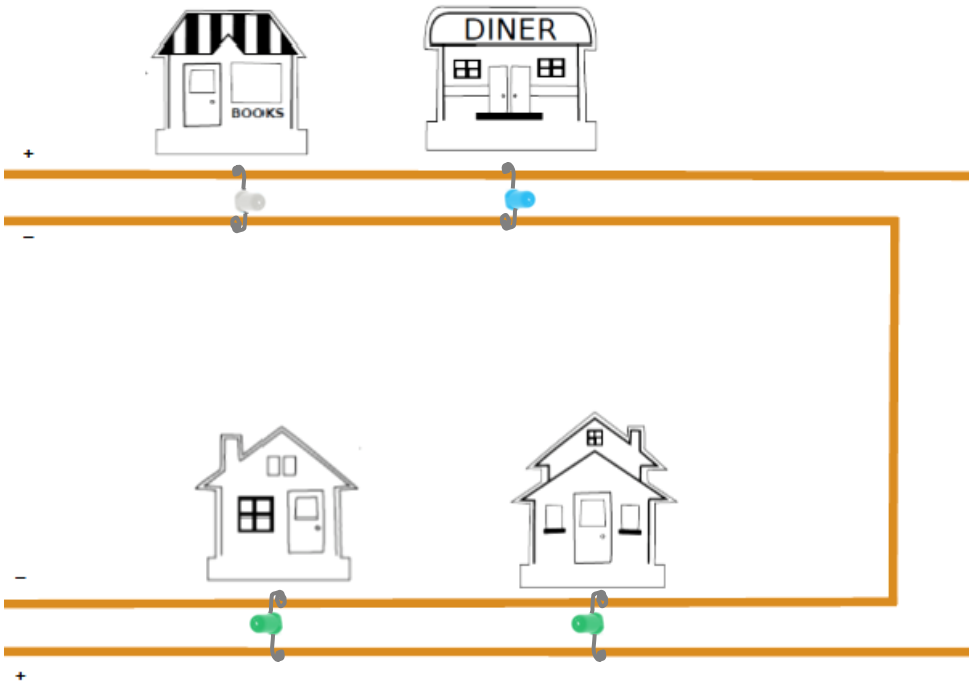
Copper Tape Circuits

Design a neighborhood using copper tape and stickers.

In this diagram the copper tape making the inner U-shape is the ground. Fold the copper tape to make a corner. Cutting it breaks the connection. The two positive copper tape strips are separate so you can connect each to its own pin.

Add LEDs for each building.

Save the code on the previous page to your Gemma and use alligator to alligator wires to connect the Gemma to your neighborhood.



CHALLENGE : Edit the code so that the lights in the stores behave differently than the lights in the residential neighborhood. Add more LEDs. Add another "street."

Use what you've learned about LEDs and circuits to light the insides of your houses with as many lights as possible.

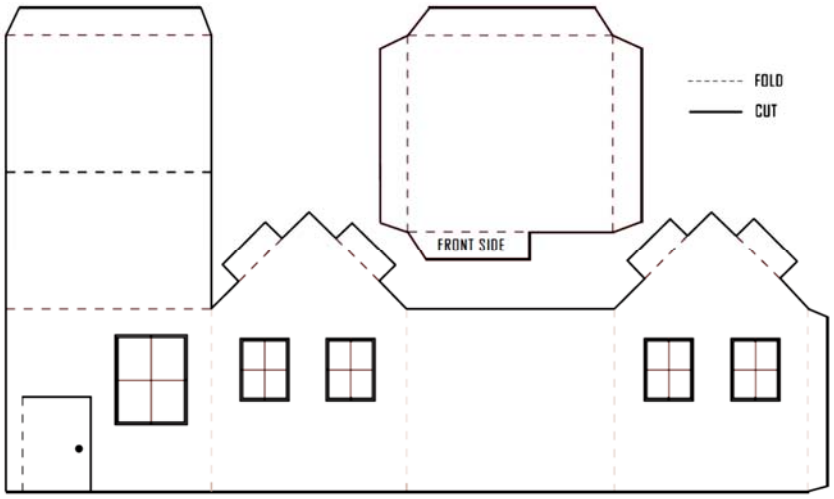
Cut out and assemble each small house.

Design a system that supplies power to each house in your community. Make more buildings to complete the neighborhood.

Use copper tape to connect the parts of the system.

Add foil, paper clips or other conductors if you need them.

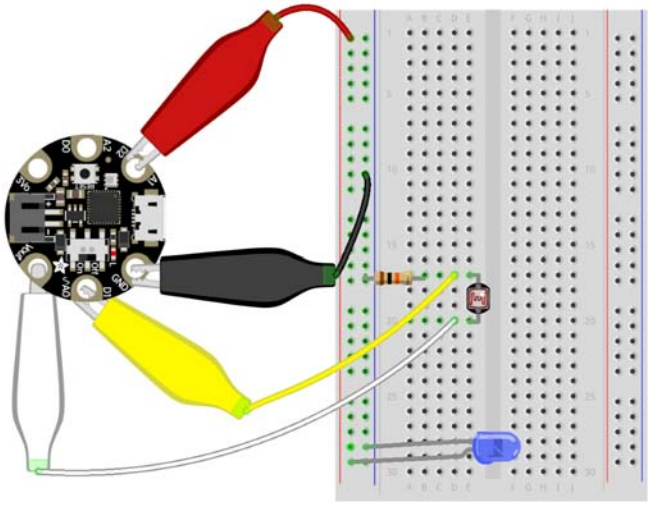
Use the Gemma to control the lighting.



This material is based upon work supported by the Department of Energy and the Department of Homeland Security

Circuits with a Sensor

PART 1 - ADD A SENSOR



Build the circuit

In the diagram at the right the LED is still connected to pin D2.

We want the LED to light when the photo sensor detects light.

Add a 10KΩ resistor and a photo sensor to the circuit. The resistor is between the photo sensor and ground. The photo sensor is connected to Vout and A0.

Create code to use the sensor

Click the New button to begin a new program in Mu or edit the main.py code on your Gemma.

Save the code below to the CIRCUITPY disk as main.py. If Mu says main.py already exists, choose to replace it.

```
1  import time
2  import board
3  import analogio
4  import digitalio
5
6  sensor = analogio.AnalogIn(board.A0)
7
8  led = digitalio.DigitalInOut(board.D2)
9  led.direction = digitalio.Direction.OUTPUT
10
11
12
13 while True:
14
15     sensorVoltage = sensor.value/65536*sensor.reference_voltage
16
17     if(sensorVoltage > 2.5):
18         led.value = True
19
20     else:
21         time.sleep(1)
22         led.value = False
23
```

The photo sensor senses light. When there is enough light, the LED connected to pin D2 is on. When you cover the sensor or take it into a dark room, the LED stays on for one more second and then turns off.

Line 17 **if(sensorVoltage > 2.5):** sets the photo sensor value at which the LED turns on. How does your circuit act when you change this value? Try larger or smaller values. What happens when you change **>** to **<** ?

READ THE CODE

The first four lines import libraries. The board library gives access to the physical pins of the microcontroller. The analogio board allows control of analog pins on the Gemma, and the digital input output library allows control of the digital pins. The time library allows control of how long the LED is on or off.

Line 6 assigns pin A0 to detect analog data from the photo sensor.

Lines 8 and 9 send data to the LED that is connected to pin D2

The while True loop reads the data from the photo sensor and does some math to make a voltage number. If the light sensor detects a high light level (more than 3), the LED turns on and stays on as long as it is light and 1 second more.

If the sensor detects a lower level of light the LED remains off.

This material is based upon work supported by the Department of Energy and the Department of Homeland Security

Circuits with a Sensor

Investigate the photo sensor

Circuit python and the Mu editor can allow you to see the voltage number detected by the photo sensor.

Add line 16, **print(sensorVoltage).**

Click the **Serial** button in the top menu to open a connection to the Gemma. Then save.

The numbers in the REPL (read evaluate print loop) window are the sensorVoltage values. How do these numbers change as the light on the sensor changes? Adjust the value in the line 17 **if(sensorVoltage >2.5):** for varying light situations.

untitled | main.py

```
1  import time
2  import board
3  import analogio
4  import digitalio
5
6  sensor = analogio.AnalogIn(board.A0)
7
8  led = digitalio.DigitalInOut(board.D2)
9  led.direction = digitalio.Direction.OUTPUT
10
11
12
13 while True:
14
15     sensorVoltage = sensor.value/65536*sensor.reference_voltage
16     print(sensorVoltage)
17     if(sensorVoltage > 2.5):
18         led.value = True
19
20     else:
21         time.sleep(1)
22         led.value = False
23
24
25
```

Adafruit CircuitPython REPL

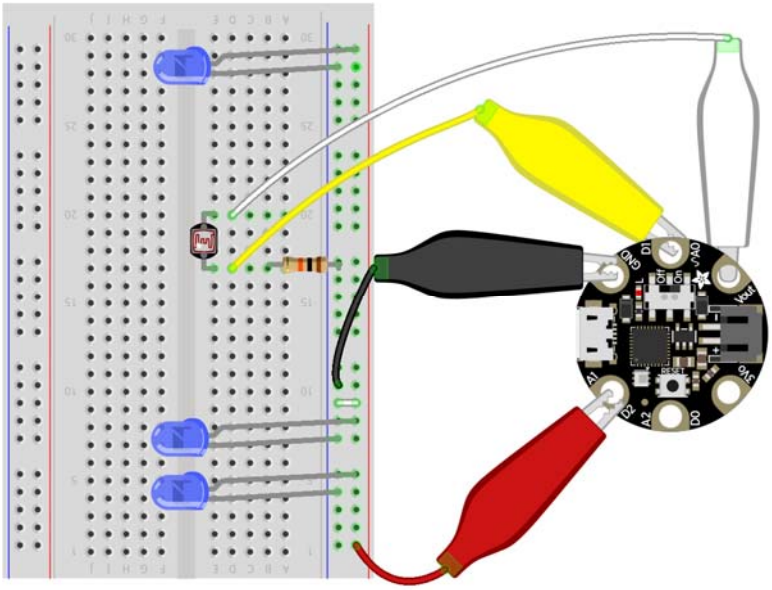
```
3.29517
3.29597
3.29355
3.29275
3.29517
```

CHALLENGE :

Edit the code so that the LED is on in darkness and off when there is light.

Add lines of code to make the LED blink when darkness is detected.

Add more code to make the LED blink at one rate when light is detected and at a different rate when it's darker.



Add more LEDs

What happens when you add more LEDs to the red and blue rails?

All of the LEDs receive data from pin D2 so they all act the same.

If we want two LEDs to each do something different, each needs to get data from separate pins.

This material is based upon work supported by the Department of Energy and the Department of Homeland Security

PART 2 - ADDRESS ANOTHER PIN

Build the circuit

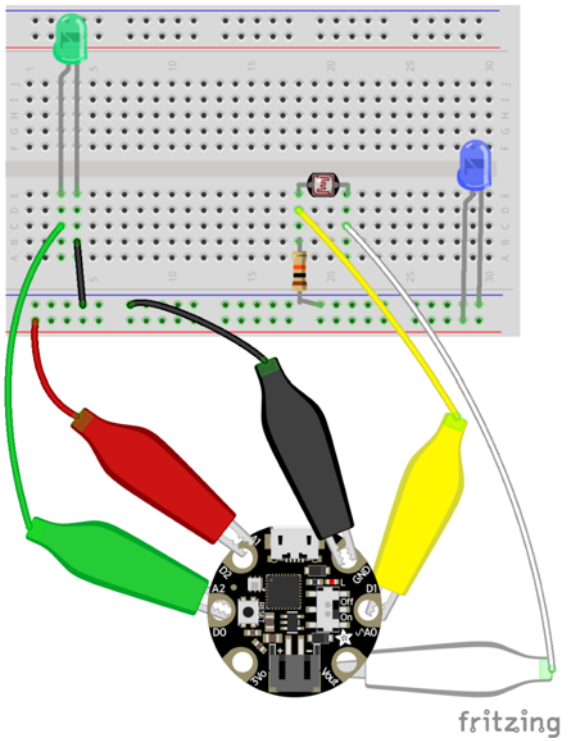
Add a green LED to an area of the breadboard away from the rails.

This circuit has the green and blue LEDs configured in the same way as the circuit on page 7 and the sensor in the same way as the circuit on page 10.

In the diagram cathodes (shorter negative legs) of both LEDs are connected to ground (the blue rail connected to the GND pin). A jumper connects the blue rail to the cathode of the green LED.

The anode (longer positive leg) of the blue LED in the red rail is still connected to pin D2.

The green alligator to jumper wire connects pin D0 to the anode of the green LED.



Be careful not to connect the LED to the 3Vo pin. It may output too much voltage and burn out the LED.

Edit the code

The code below is similar to the **main.py** code from the previous activity that is already on your Gemma. Load the code into Mu and add more code to address pin D0.

Remember, we changed the variable names to ledblue and ledgreen to help remember which is which. You can name the variables however makes sense to you.

The code in the **while True:** loop turns the blue LED on and the green LED off when the sensor detects light. Otherwise the blue LED is off and the green LED is on.

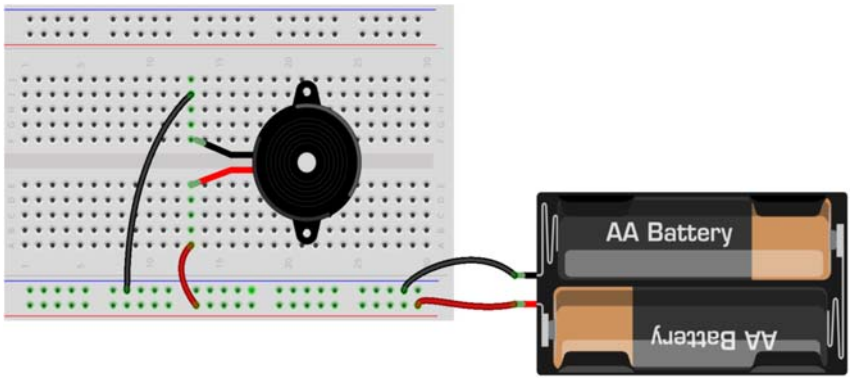
```
import time
import board
import analogio
import digitalio

sensor = analogio.AnalogIn(board.A0)

ledblue = digitalio.DigitalInOut(board.D2)
ledblue.direction = digitalio.Direction.OUTPUT

ledgreen = digitalio.DigitalInOut(board.D0)
ledgreen.direction = digitalio.Direction.OUTPUT

while True:

    sensorVoltage = sensor.value/65536*sensor.reference_voltage

    if(sensorVoltage > 2.5):
        ledblue.value = True
        ledgreen.value = False

    else:
        ledblue.value = False
        ledgreen.value = True
```

CHALLENGE : Edit the code so that the lights in the stores behave differently than the lights in the residential neighborhood. Add more LEDs. Add another "street."

This material is based upon work supported by the Department of Energy and the Department of Homeland Security

Circuits with a Piezo Buzzer

Build a circuit



Insert the piezo buzzer across the ravine. (The one in your kit has legs instead of wires like the diagram. It fits perfectly across the ravine.)

Use a jumper wire to connect one leg of the buzzer to ground (the blue rail that's connected to the negative battery wire). Use a second jumper wire to connect the other leg of the buzzer to positive.

Connect the battery holder to ties on the breadboard. Connect red to red and dark to dark.

Do you hear the buzzer?

Add the Gemma and create code to control the piezo buzzer

Click the **New** button to begin a new program in Mu or edit the main.py code on your Gemma.

Save the code below to the CIRCUITPY disk as **main.py**. If Mu says main.py already exists, choose to replace it.

```
1  import time
2  import board
3  import pulseio
4
5  piezo = pulseio.PWMOut(board.A1, duty_cycle=0, frequency=440, variable_frequency=True)
6
7
8
9  tempo = 0.3
10 twinkle_twinkle_little_star = [ (261, tempo), #c
11                                 (261, tempo), #c
12                                 (392, tempo), #g
13                                 (392, tempo), #g
14                                 (440, tempo), #a
15                                 (440, tempo), #a
16                                 (392, tempo * 2), #g
17                                 (0,   tempo), #rest
18                                 ]
19
20
21 while True:
22
23     for f, d in twinkle_twinkle_little_star:
24         if(f != 0):
25             piezo.frequency = f
26             piezo.duty_cycle = 65536 // 2  # On 50%
27             time.sleep(d)  # On for 1/4 second
28             piezo.duty_cycle = 0  # Off
29             time.sleep(d)  # Pause between notes
30         else:
31             time.sleep(d)
32     time.sleep(tempo/2)
```

Read the code

The first three lines import libraries. The pulse input output library allows control of pulsed input and output

Line 6 assigns the piezo buzzer to pin A1 sending pulsed output with variable frequency

In line 9 the tempo variable controls the playback speed.

The notes of the song are in the array of pairs, where the first item in the pair is the frequency of the note, and the second item is the duration in seconds.

The while True loop reads the data in the array of notes and sends it to the piezo buzzer

Each frequency value in the array represents a musical note. Line 23 sets the frequency of the piezo buzzer to the note frequency and keeps the buzzer on for the duration of the note before moving on to the next note.

This material is based upon work supported by the Department of Energy and the Department of Homeland Security

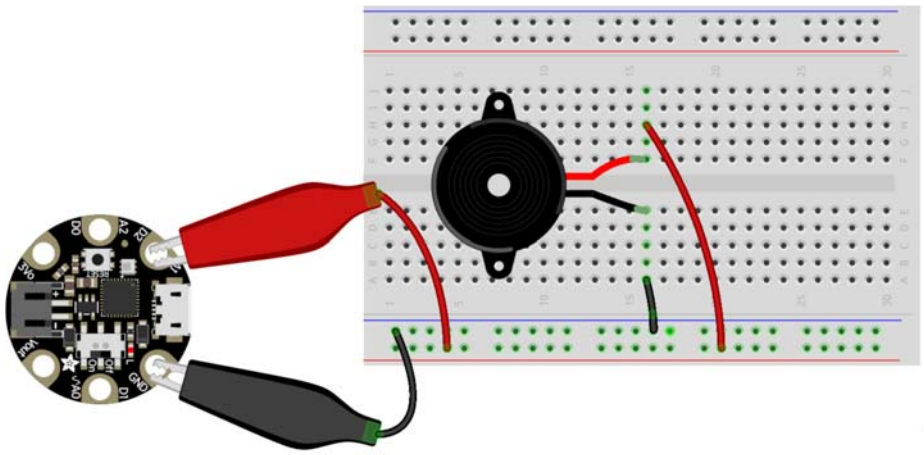
Circuits with a Piezo Buzzer

Build the circuit

Use alligator to jumper wires to attach the GND (ground) pin to any tie in the blue rail and the A1 pin to any tie in the red rail.

Does the piezo buzzer play a few notes of Twinkle, Twinkle Little Star?

Add the rest of the notes for Twinkle, Twinkle Little Star to your code The array pairs for the notes are on the next page.



CHALLENGES :

Add the photo sensor and 10KΩ resistor to your circuit. Place the 10KΩ resistor between ground and a connection to pin A0 (A2 also works) and in series with one leg of the photo sensor. Connect the other leg to Vout. This circuit is similar to the ADD A SENSOR circuit on page 10 and the diagram below.

Edit your code so that the music plays when the sensor detects light. Remember to import the analogio library so your program can read the data from the photo sensor and set the sensor variable to receive data from the pin (A0 or A2) you connect to one leg of the photo sensor. Use a while True loop to read the sensorVoltage and play music if the sensorVoltage is high.

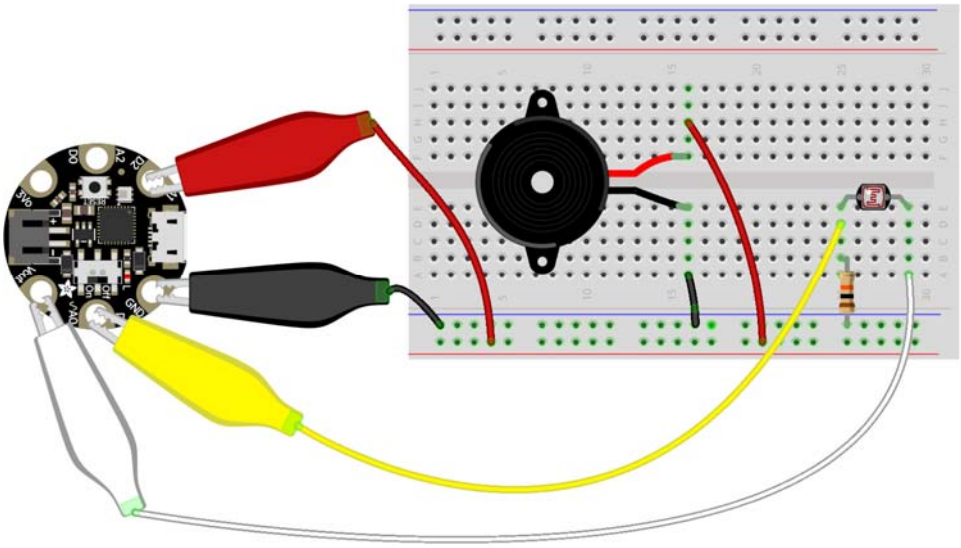
Add an LED and create code to make it light when the sensor detects darkness.

Connect your circuit to the LEDs in your copper tape neighborhood. When do you your LEDs to light?

The components of your kit can be used for all kinds of creative projects. There are ideas on the Adafruit website and there are tips on the CREDC Education website.

Alligator clips and breadboards are quick and easy for testing your designs, but you may want to try more copper tape circuitry or a conductive thread project.

Contact us through our website. Send us pictures of your projects!



This material is based upon work supported by the Department of Energy and the Department of Homeland Security

Each frequency value in this array represents a musical note. Tempo tells the buzzer how long to play the note.

```
tempo = 0.3
twinkle_twinkle_little_star = [ (261, tempo), #c
                                (261, tempo), #c
                                (392, tempo), #g
                                (392, tempo), #g
                                (440, tempo), #a
                                (440, tempo), #a
                                (392, tempo * 2), #g
                                (0,   tempo), #rest
                                (349, tempo), #f
                                (349, tempo), #f
                                (329, tempo), #e
                                (329, tempo), #e
                                (294, tempo), #d
                                (294, tempo), #d
                                (261, tempo * 2), #c
                                (0,   tempo * 4), #rest
                                (392, tempo), #g
                                (392, tempo), #g
                                (349, tempo), #f
                                (349, tempo), #f
                                (329, tempo), #e
                                (329, tempo), #e
                                (294, tempo), #d
                                (0,   tempo), #rest
                                (392, tempo), #g
                                (392, tempo), #g
                                (349, tempo), #f
                                (349, tempo), #f
                                (329, tempo), #e
                                (329, tempo), #e
                                (294, tempo), #d
                                (0,   tempo * 4), #rest
                                (261, tempo), #c
                                (261, tempo), #c
                                (392, tempo), #g
                                (392, tempo), #g
                                (440, tempo), #a
                                (440, tempo), #a
                                (392, tempo * 2), #g
                                (0,   tempo), #rest
                                (349, tempo), #f
                                (349, tempo), #f
                                (329, tempo), #e
                                (329, tempo), #e
                                (294, tempo), #d
                                (294, tempo), #d
                                (261, tempo * 2), #c
                                ]
#ccggaag ffeeddc ggffeed ggffeed ccggaag ffeeddc ;
```


This material is based upon work supported by the Department of Energy and the Department of Homeland Security