ENG 10: Lecture 5

Spring 2021

Engineering of the week

1. Rajvir Logani “Bone conduction hearing devices”

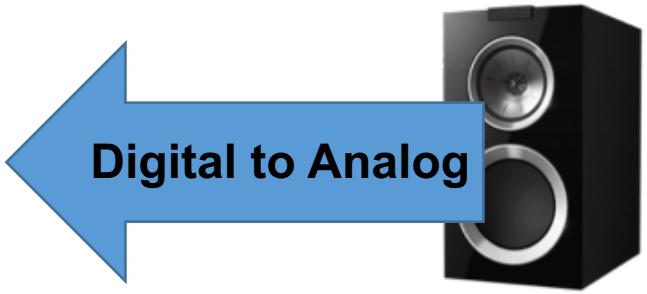
2. Evan Parez “Space X's reusable rocket boosters”

Review: Signals

Signals

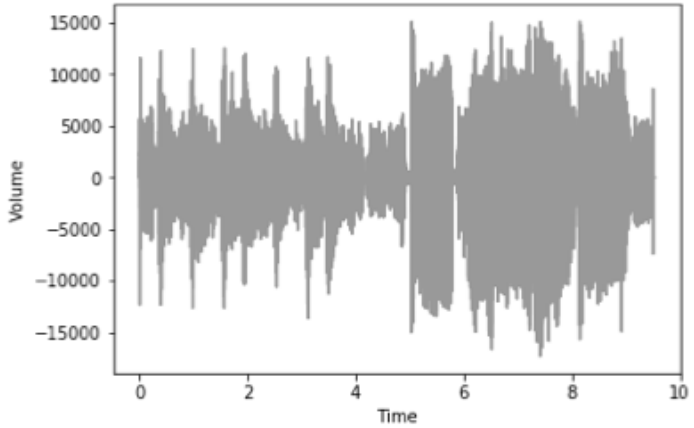
- **Signal is used to convey information in a certain method.**
- **Continuous signal to a sequence of numbers (discrete form)**

Analog



Digital

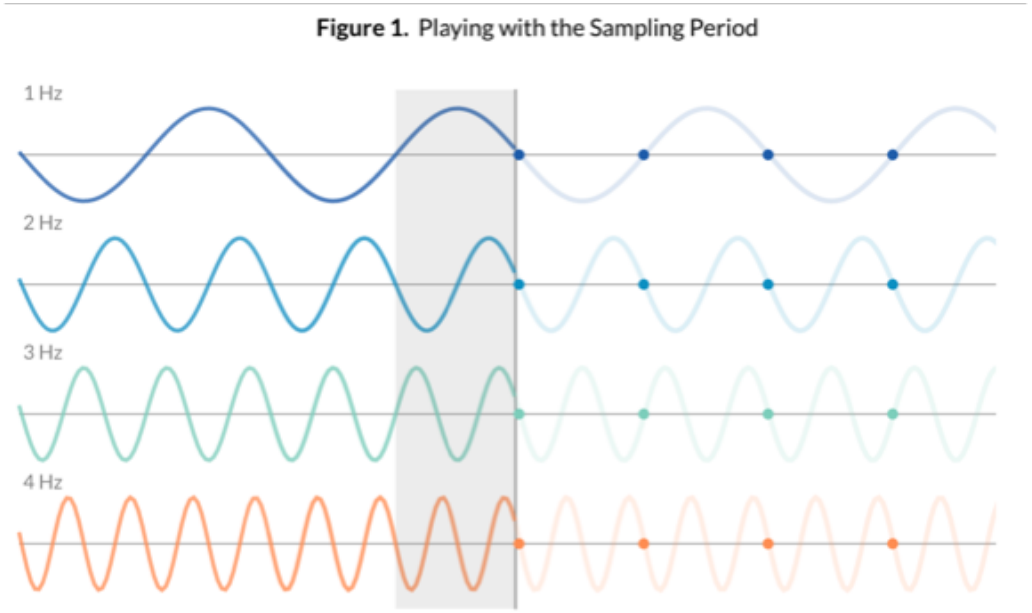
- Sound data: **[0, 0, 0, 0, 123, 982, 1012, -229, -980…]**
- Time vector (is given with **sampling frequency [Hz]**)
  → Time = # of samples/sampling rate



Continuous signal ≈ More dense “discrete” signal

Importance of Sampling

- **Failing in choosing an appropriate sampling rate will distort the result or lose information**



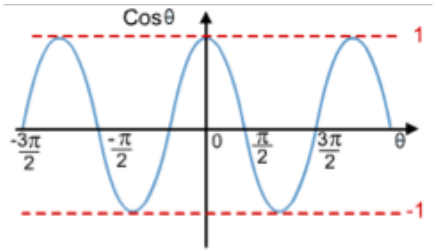
- **Nyquist Sampling Theorem: Need to sample at least twice as fast as the event occurs**

$$f_s > 2B$$

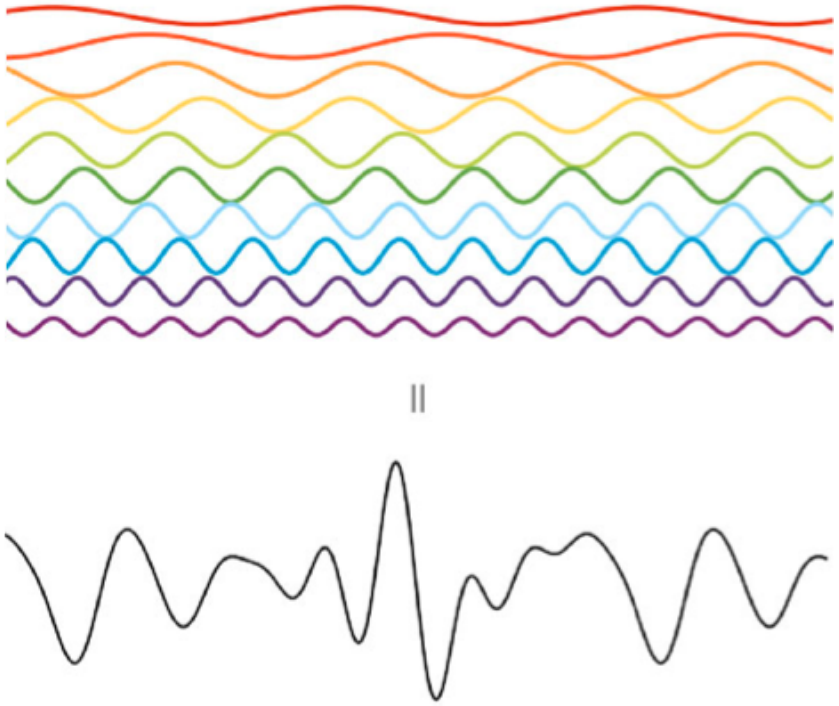
Common sampling frequency for digital songs: **44.1 kHz** → Signals lower than **22.05 kHz**

Modeling Signal

- **Modeling a signal using trig. functions:** $x(t) = Acos(2\pi ft)$



- **Complex wave signals = many simple wave signals**



$$A_1 cos(2\pi f_1 t)$$
$$A_2 cos(2\pi f_2 t)$$
$$A_3 cos(2\pi f_3 t)$$
$$A_4 cos(2\pi f_4 t)$$
$$\dots$$
$$\dots$$
$$A_n cos(2\pi f_n t)$$

$$x(t) = A_1 cos(2\pi f_1 t) + A_2 cos(2\pi f_2 t) + \dots + A_n\, cos(2\pi f_n t)$$
$$= \sum_{j=1}^{n} A_j cos(2\pi f_j t)$$

We can model this complex signal as a *mixture of independent, simple* sines *and* cosines

Deconstruction and Reconstruction of Complex Signals

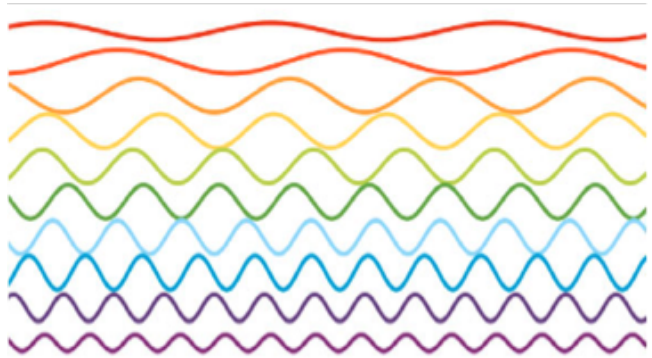
- **Modification of complex wave signals can be processed by using Fast Fourier Transform (FFT) and inverse FFT**

- **Complex wave signals = many simple wave signals**

$x(t)$

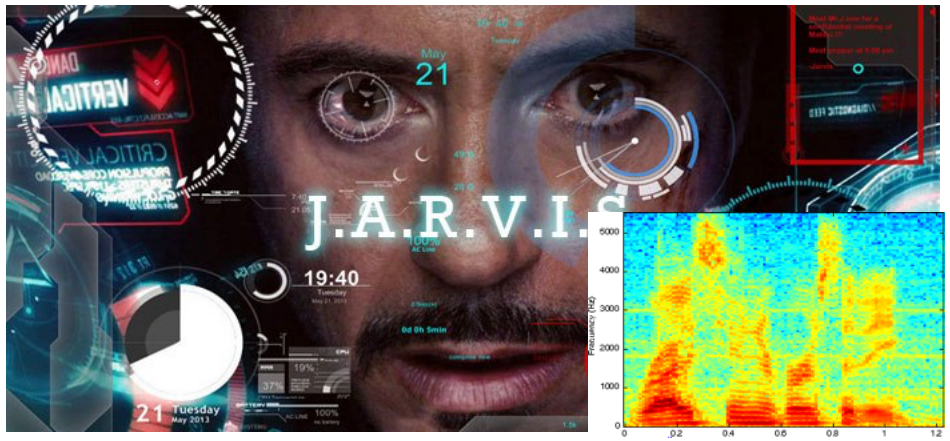
FFT

Inverse FFT

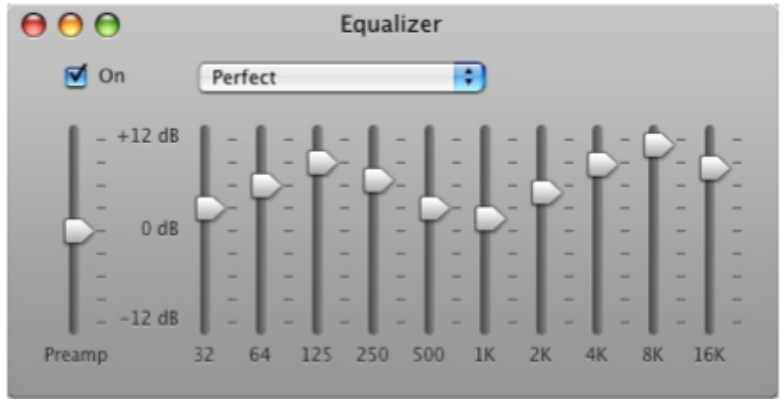


$A_1 cos(2\pi f_1 t)$
$A_2 cos(2\pi f_2 t)$
...
...
$A_n cos(2\pi f_n t)$

Applications of FFT



Artificial Intelligence Assistant



$$A_1 cos(2\pi f_1 t)$$
$$A_2 cos(2\pi f_2 t)$$
$$A_3 cos(2\pi f_3 t)$$
$$A_4 cos(2\pi f_4 t)$$
$$\dots$$
$$\dots$$
$$A_n cos(2\pi f_n t)$$

Equalizer (Filtering)

Alexa

Google Home

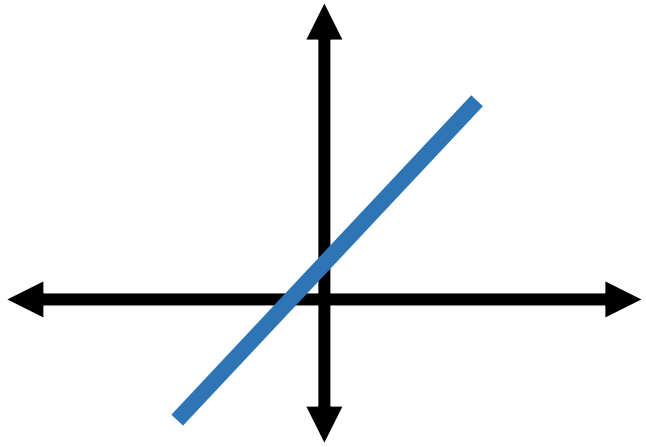
Noise-canceling

Math:
Systems of Linear Equations

What is Linear Algebra?

- A branch of mathematics
- Covers topics such as vectors, matrices, and systems of linear equations
- Relevant for all fields of engineering

$$y = mx + b$$



Why Linear Algebra is important?

Let’s deal with a **real-world** problem:

SpaceX Falcon 9 Rocket Re-entry and Landing

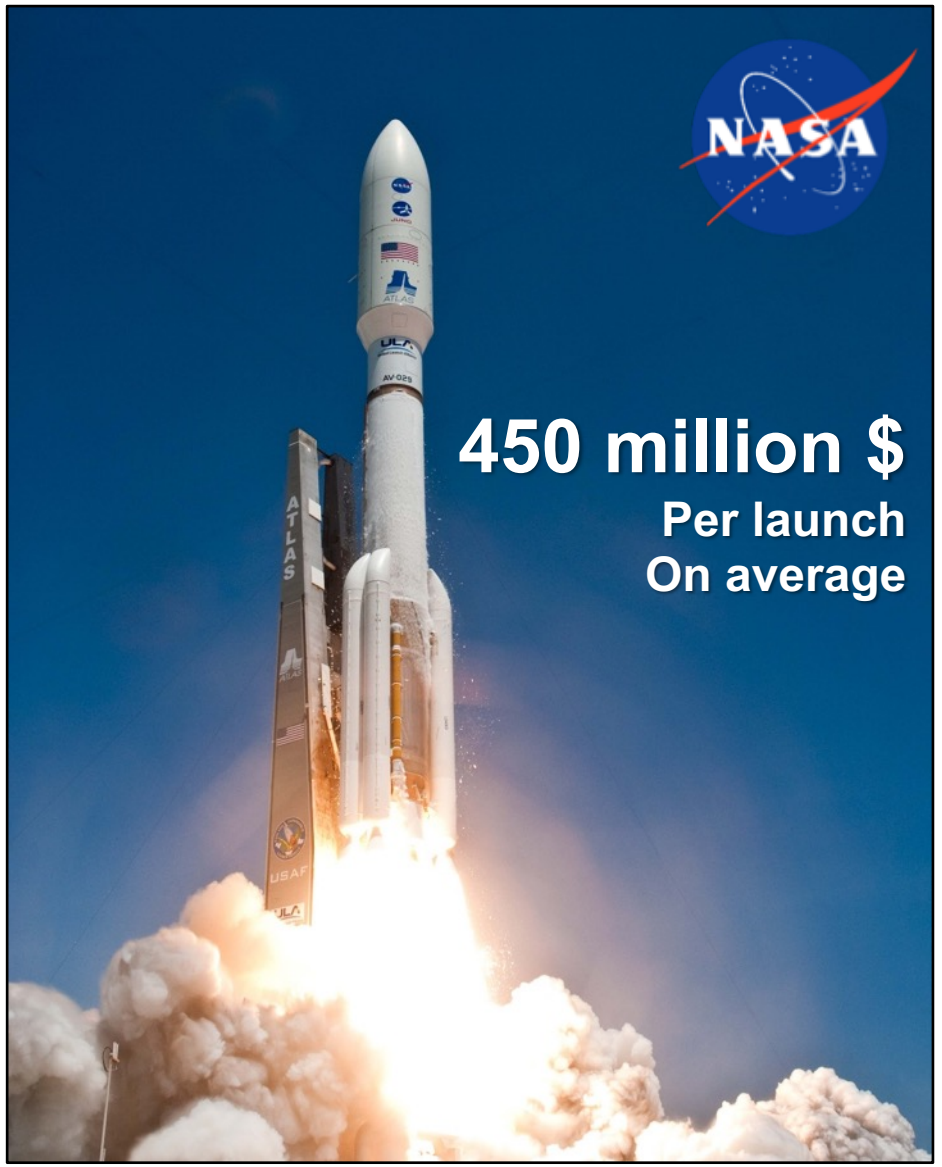
Why Linear Algebra is important?

- **How many variables** does this landing system need?
- **How many equations** does this landing system need?
- All the computations have to be done **in real-time** for a high accuracy of landing.

Cost of One Launch: SpaceX vs NASA

13.5 % of the Nasa cost of one launch

VS

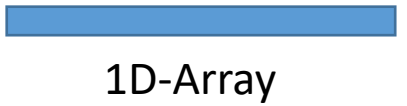


Data Structure: Scalars, Vectors, and Matrices

- Scalar $a = 5$

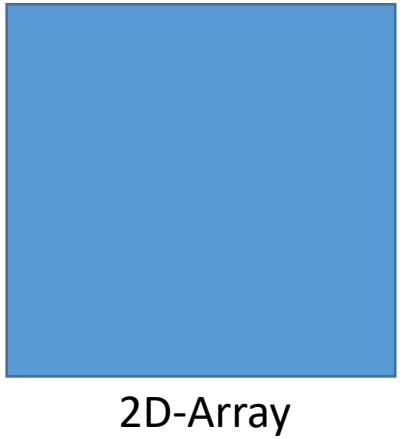
$a \in \mathbf{R}$

- Vector $x = (0.1, 2.7, -1.8)$



$x \in \mathbf{R}^3$

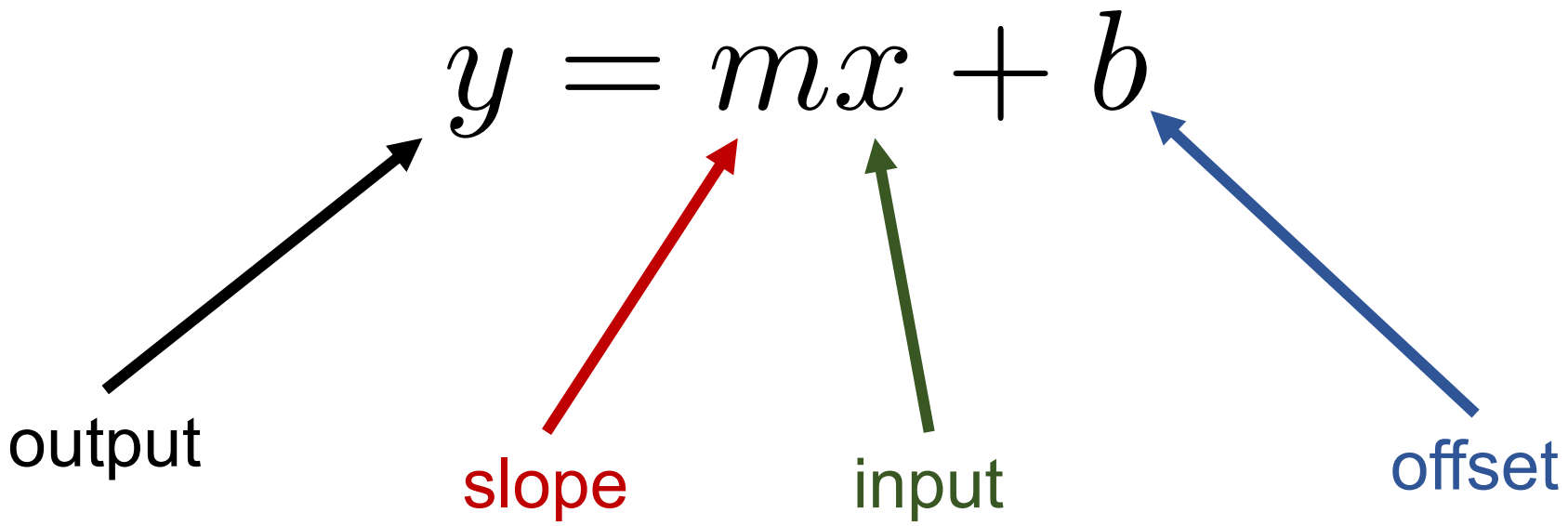
- Matrix $C = \begin{bmatrix} 1 & 0 & 3 \\ 0 & 5 & 7 \\ 8 & 2 & 1 \end{bmatrix}$



$C \in \mathbf{R}^{3\times 3}$

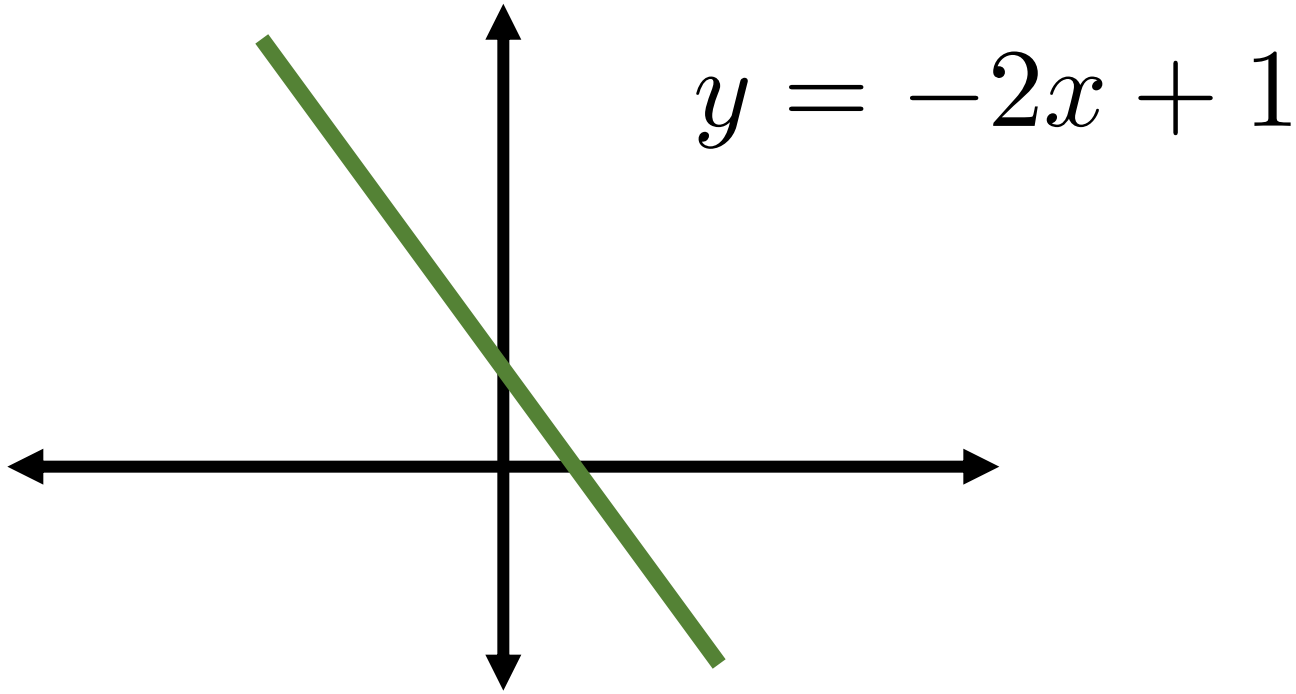
Linear Equations

Start with a single linear equation:



Linear Equations

What do linear equations look like?



Linear Equations

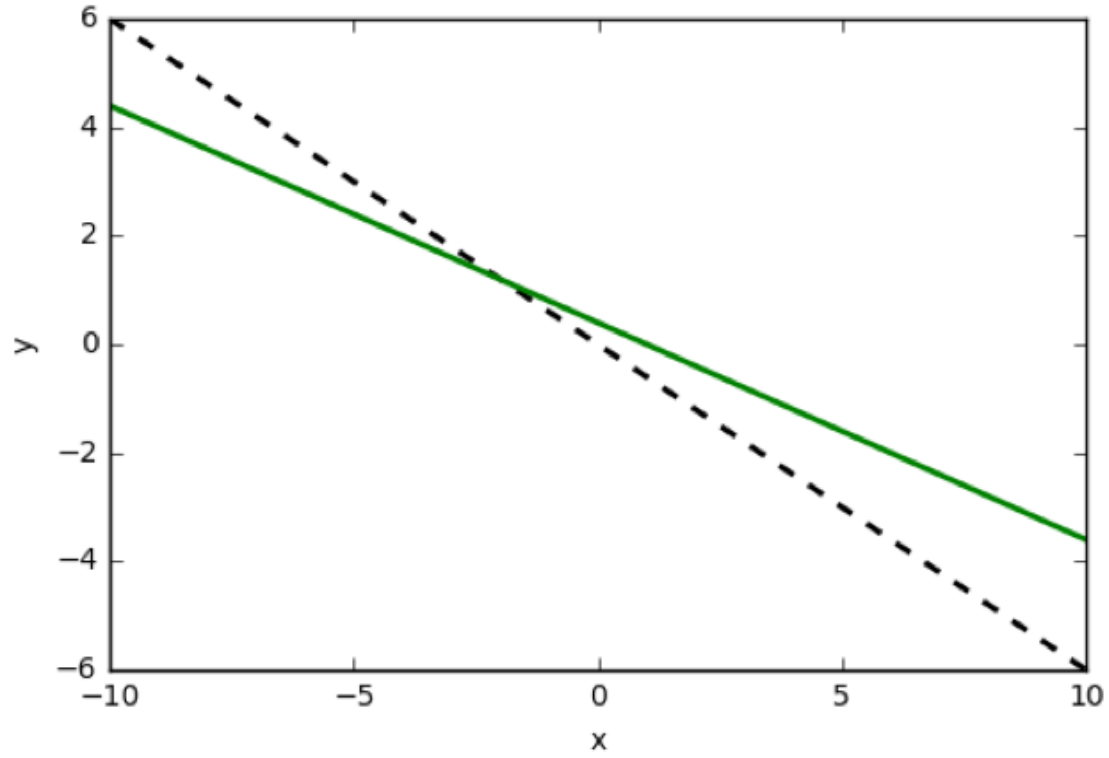
What about two linear equations?

$$3x + 5y = 0$$

$$2x + 1y = 2$$

Linear Equations

What do they look like?



Linear Equations

Can you solve these two equations by hand?

$$3x + 5y = 0$$

$$2x + 1y = 2$$

Linear Equations

Did you notice any patterns in the two equations?

$$a_1x + b_1y = c_1$$

$$a_2x + b_2y = c_2$$

Linear Equations

Now let's try grouping the terms:

$$\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix} \quad \begin{bmatrix} x \\ y \end{bmatrix} \quad \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$$

coefficients parameters constants

Linear Equations

Do these groupings remind you of anything?

$$\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix} \qquad \begin{bmatrix} x \\ y \end{bmatrix} \qquad \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$$

coefficients — matrix

parameters — vector

constants — vector

Systems of Linear Equations

- The examples we just looked at are **systems of linear equations.**
- Specifically, they are 2x2 systems of equations:
    - 2 equations
    - 2 variables/unknowns
- Now let's solve a 1000 x 1000 system by hand ☺?!

Systems of Linear Equations

- The examples we just looked at are **systems of linear equations.**
- Specifically, they are 2x2 systems of equations:
    - 2 equations
    - 2 variables
- Now let's solve a ~~1000x1000~~ **3x3** system ~~by hand~~ **computationally**!

3x3 System of Linear Equations

Can you solve this by hand?

$$3x + 2y - z = 1$$

$$2x - 2y + 4z = -2$$

$$-x + 1/2y - z = 0$$

3x3 System of Linear Equations

Do you **want to** solve this by hand?

$$3x + 2y - z = 1$$

$$2x - 2y + 4z = -2$$

$$-x + 1/2y - z = 0$$

3x3 System of Linear Equations

Let's try rewriting the equations:

$$a_1x + a_2y + a_3z = b_1$$

$$a_4x + a_5y + a_6z = b_2$$

$$a_7x + a_8y + a_9z = b_3$$

3x3 System of Linear Equations

Now let's group the components:

$$\begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix} \quad \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

3x3 System of Linear Equations

Question: Is there another way we can write this?

$$\begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix} \quad \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

3x3 System of Linear Equations

Answer: YES!

$$a_{1,1}x_1 + a_{1,2}x_2 + a_{1,3}x_3 = b_1$$

$$a_{2,1}x_1 + a_{2,2}x_2 + a_{2,3}x_3 = b_2$$

$$a_{3,1}x_1 + a_{3,2}x_2 + a_{3,3}x_3 = b_3$$

3x3 System of Linear Equations

Answer: YES!

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \quad \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

A **x** **b**

3x3 System of Linear Equations

Answer: YES!

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \quad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

System of Linear Equations

What about bigger systems (e.g. 5x5, 10x10, 10^6x10^6)?

Same ideas apply:

1) Group the components

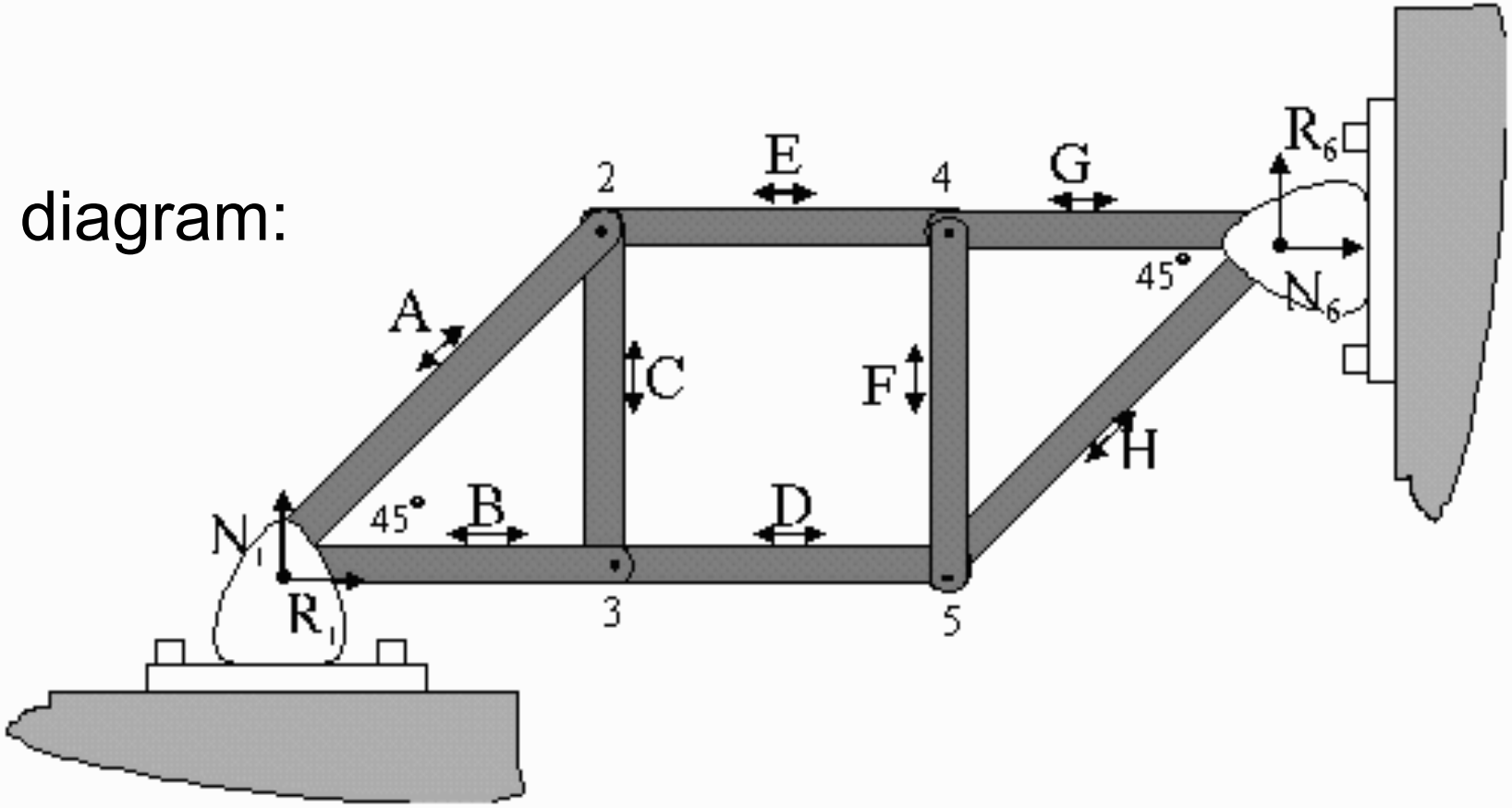
2) Rewrite as Ax = b

3) Solve

More Complex Problem 1: Truss bridge

Different types of forces exist along the beams: Tension, Shear, Bending, and Compression

Force diagram:



More Complex Problem 1: Truss bridge

- 12 x 12 system

Equations:

- 12 equations
- 12 unknowns

1) $\rightarrow \quad R_1 - \frac{1}{\sqrt{2}} A - B = F_{1x}$

$\uparrow \quad N_1 - \frac{1}{\sqrt{2}} A = F_{1y}$

2) $\rightarrow \quad \frac{1}{\sqrt{2}} A - E = F_{2x}$

$\uparrow \quad \frac{1}{\sqrt{2}} A + C = F_{2y}$

3) $\rightarrow \quad B - D = F_{3x}$

$\uparrow \quad -C = F_{3y}$

4) $\rightarrow \quad E - G = F_{4x}$

$\uparrow \quad F = F_{4y}$

5) $\rightarrow \quad D - \frac{1}{\sqrt{2}} H = F_{5x}$

$\uparrow \quad -F - \frac{1}{\sqrt{2}} H = F_{5y}$

6) $\rightarrow \quad N_6 + G + \frac{1}{\sqrt{2}} H = F_{6x}$

$\uparrow \quad R_6 + \frac{1}{\sqrt{2}} H = F_{6y}$

More Complex Problem 1: Truss bridge

- 12 x 12 system

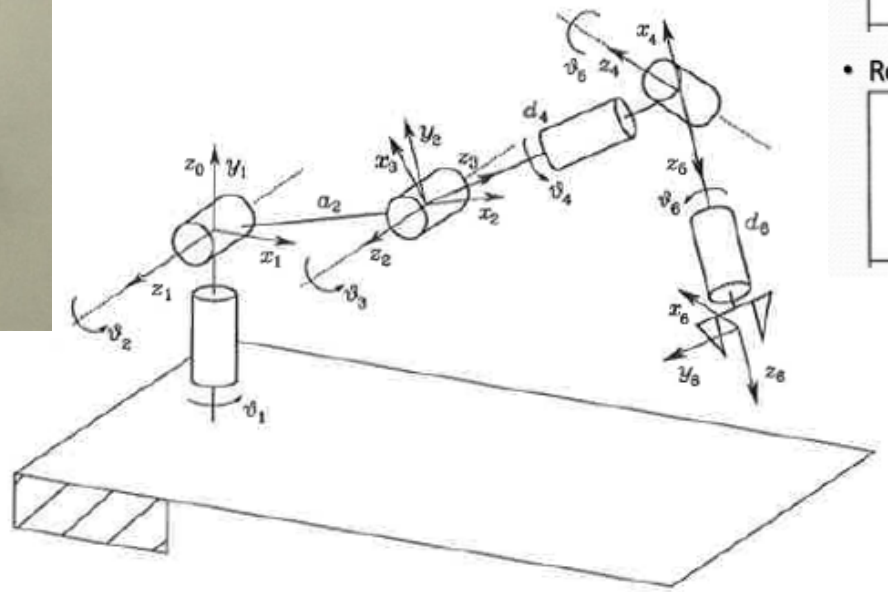
Ax = b form:

$$\begin{bmatrix} 1 & 0 & -\frac{1}{\sqrt{2}} & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -\frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -\frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \frac{1}{\sqrt{2}} & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} R_1 \\ N_1 \\ A \\ B \\ C \\ D \\ E \\ F \\ G \\ H \\ N_6 \\ R_6 \end{bmatrix} = \begin{bmatrix} F_{1x} \\ F_{1y} \\ F_{2x} \\ F_{2y} \\ F_{3x} \\ F_{3y} \\ F_{4x} \\ F_{4y} \\ F_{5x} \\ F_{5y} \\ F_{6x} \\ F_{6y} \end{bmatrix}$$

A x b

What about an example of real-time calculations?

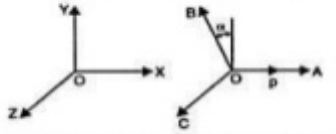
More Complex Problem 2: Robotic arms



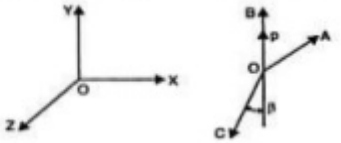
Anthropomorphic arm with spherical wrist

Geometric Interpretation of Rotation Matrix

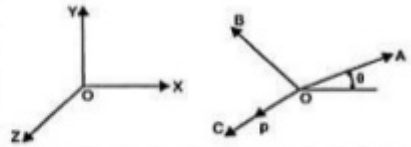
- Rotation about X-axis

Rotation	Rotation Matrix	p_{abc}	Geometric Representation
R(x, α)	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}$	$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$	

- Rotation about Y-axis

R(y, β)	$\begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\alpha \end{bmatrix}$	$\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$	

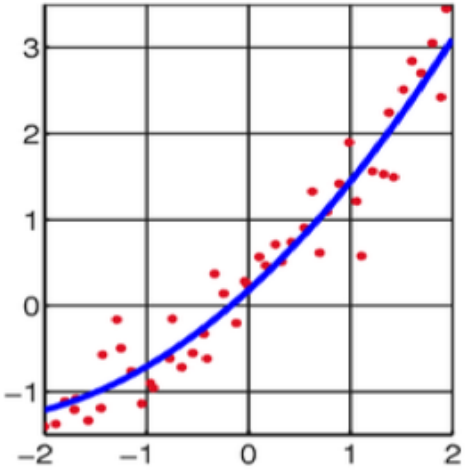
- Rotation about Z-axis

R(z, θ)	$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$	

System of Linear Equations

- But how do you solve it???
- We are unable to calculate the values in real time.
- … We can quickly compute Ax = b problems in Python with the following NumPy Module: ``x = np.linalg.solve (A, b)``

Least-Squares

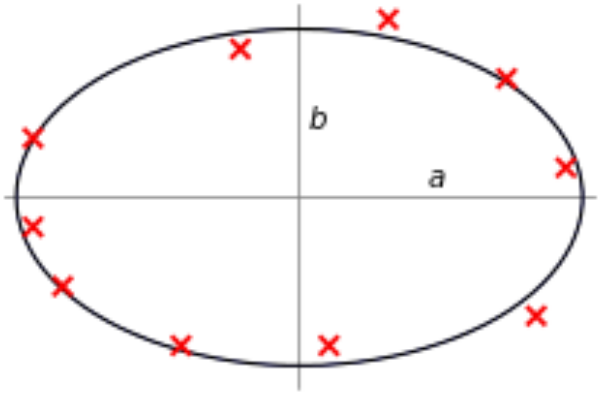


- **A method** to approximate the solution of **overdetermined** systems (i.e. systems with more equations than unknowns)

- Method dates back to Gauss (~1800s)

Carl Frederick Gauss

- **Useful for fitting data** (where you have more samples than variables)

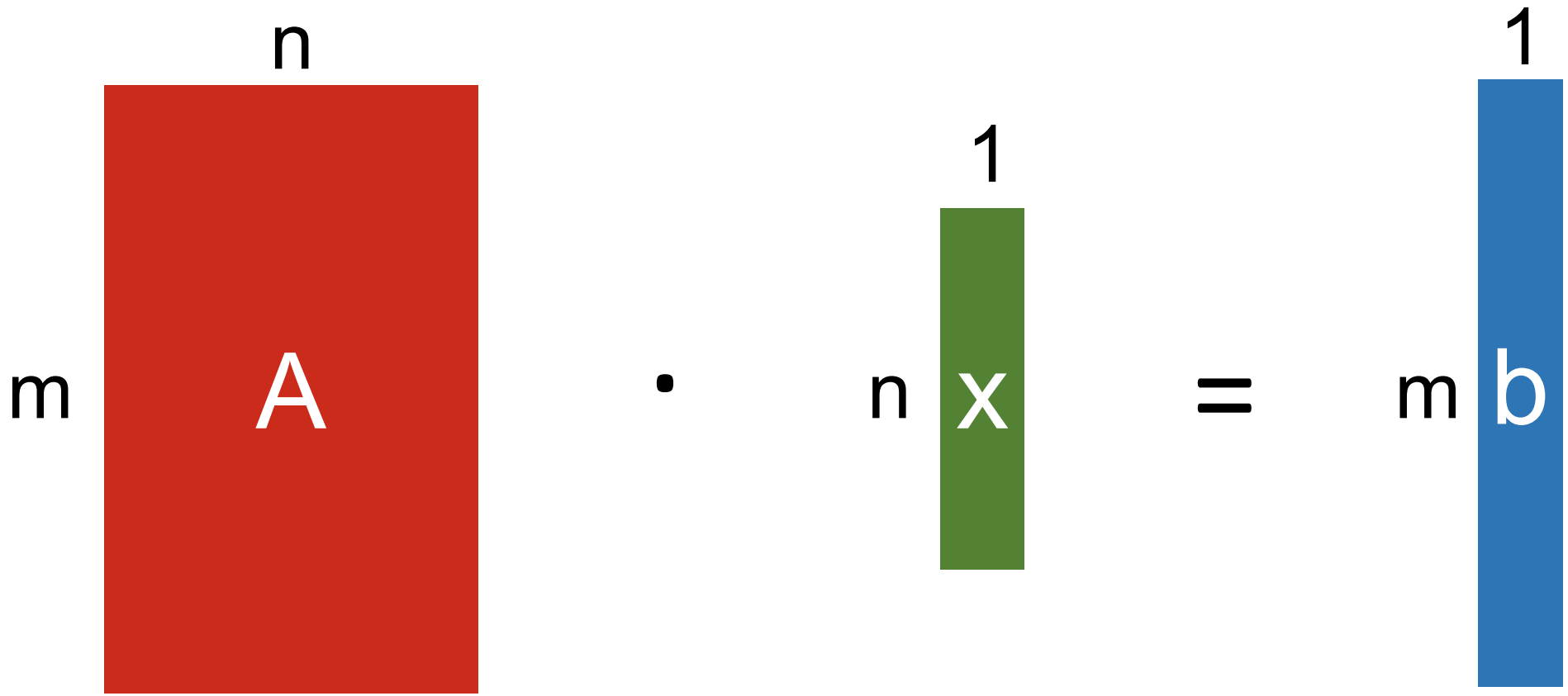


Fitting data with Least-Squares

- Given:
  - **A**: data with **m** samples, **n** features $A \in \mathbf{R}^{m \times n}$
  - **b**: corresponding target value for **m** samples $b \in \mathbf{R}^{m}$
- Find **x**: parameters for the **n** features $x \in \mathbf{R}^{n}$

$$y \approx a_1 x_1 + a_2 x_3 + \ldots a_n x_n$$

Fitting data with Least-Squares



We have to find vector **x** in such a way that the “distance” between A**x** and **b** would be minimal

Examples of uses of Least-Squares

- **Chemistry:** Learn model that relates temperature and pressure from sensor array in tank to chemical reaction rate.
- **Medicine:** Predict risk of cancer (%) based on height, weight, age, blood pressure, smoking habits, etc.
- **Environment:** Estimate power produced (kW) by a solar farm from measurements of solar irradiance, temperature, wind speed, etc.

HW 4 Submission

- Download and unzip the homework files from Canvas
- **Due: 2:00 pm May 5th** (1 week from today)
- **Complete all codes and answer questions** (inside the notebook)
- Rename the notebook file with your name: hw4_firstname_lastname.ipynb
- **Submit via Canvas** (canvas.ucsd.edu) to HW 4 assignment
- **NOTE:** make sure to submit in **ipynb** format!

- In order to get the description for Python function, you have to place your cursor over your function and press **shift+tab**!
- Link for Python functions: https://www.w3schools.com/python/python_functions.asp

Next Class

- **Probability and Statistics**

- **Project Design Challenges**

List of Teams (available on Canvas)

Team	#	Name	E-mail address
1	1	Bhattacharya, Anika	a2bhatta@ucsd.edu
	2	Rico, Fernando	frico@ucsd.edu
	3	Stevens, Tyler	trstevens@ucsd.edu
2	1	Perez, Evan	egperez@ucsd.edu
	2	Aghavali, Reza	reaghava@ucsd.edu
	3	Wang, Yiwen	yiw576@ucsd.edu
	4	Jo, U	j0jo@ucsd.edu
3	1	Tojima, Naohiro	ntojima@ucsd.edu
	2	Soebroto, Tiffany	tsoebrot@ucsd.edu
	3	Fujimoto, Takumi	tafujimoto@ucsd.edu
	4	Logani, Rajvir	rlogani@ucsd.edu

Deliverables:

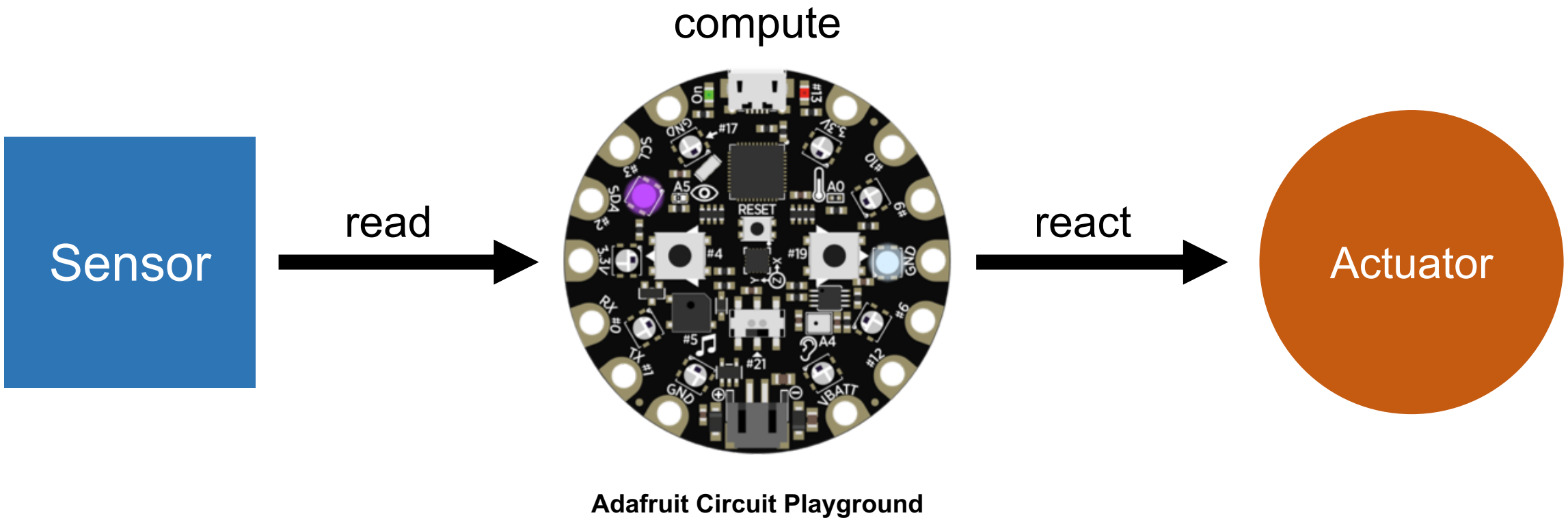
- **Proof-of-concept**: Demonstrate a working prototype (due **Week 9th, May 26th**).

- **Final presentations**: ~ 15 - 20 min. per team, in-class (**Week 10th, June 2nd**)

Notes:

- You **can draw inspiration** from other projects, but you **cannot just plagiarize** a completed project (e.g. no copying a tutorial project from Adafruit)
- Start searching a variety of Arduino projects on YouTube or Google
- You **cannot** use outside hardware without prior approval

Arduino Microcontrollers



Adafruit Circuit Playground

Input Options (Sensors)

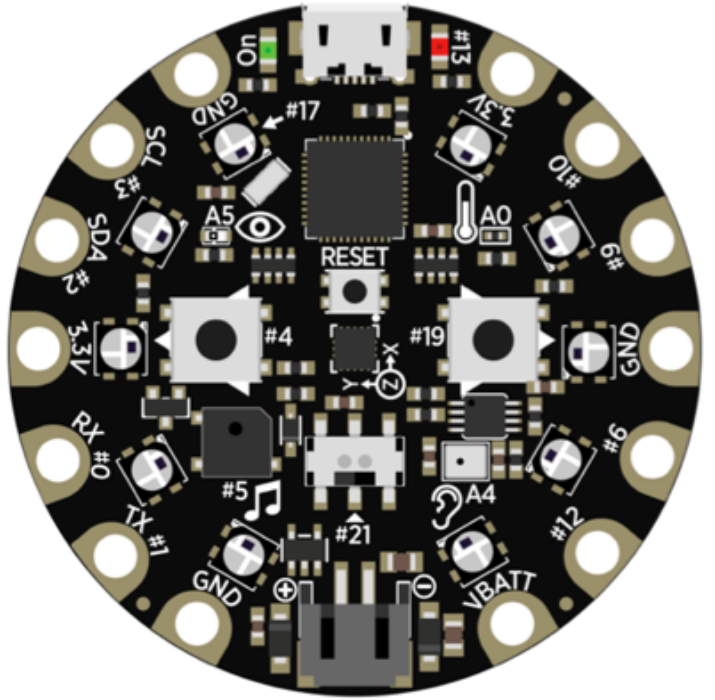
Built-in sensors on the Circuit Playground:

Sound: Noise levels, sound signals, frequencies

Temperature

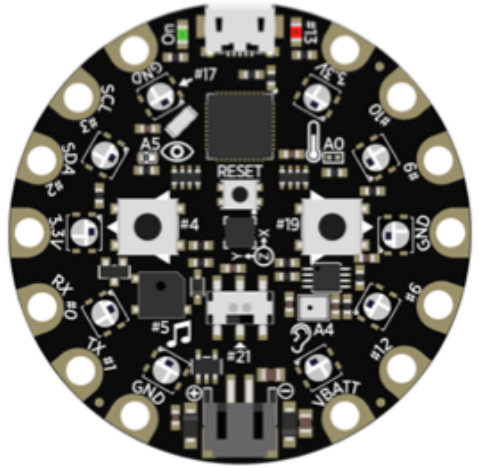
Light: Brightness, color (using LEDs)

Accelerometer: Acceleration, tilt



Output Options (Actuators)

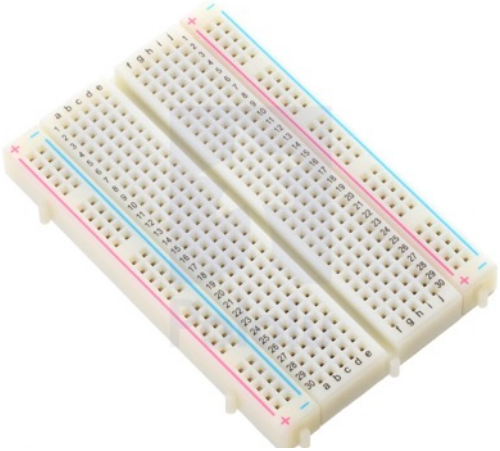
- **Built-in devices on the Circuit Playground**
  - Speaker
  - LEDs
- **Today's external actuators**

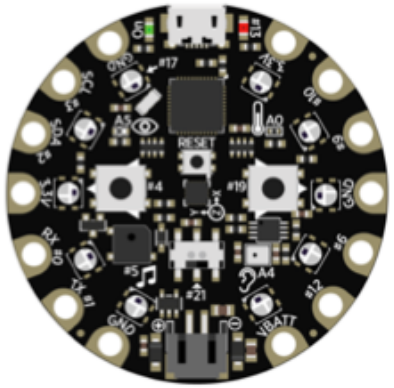


  - LEDs
  - Piezoelectric Buzzer
  - Servo motor

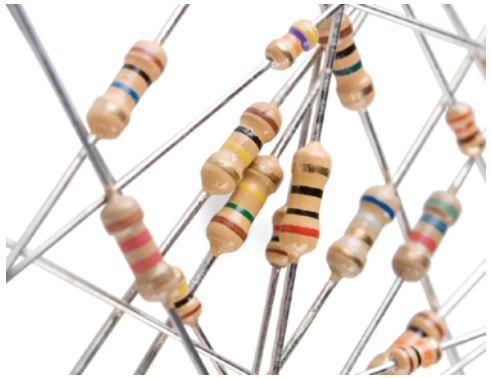
Other Circuit Building **Tools**

Breadboard

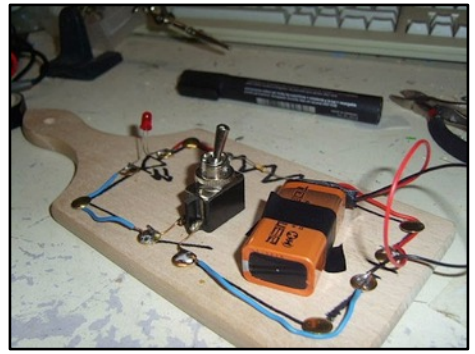




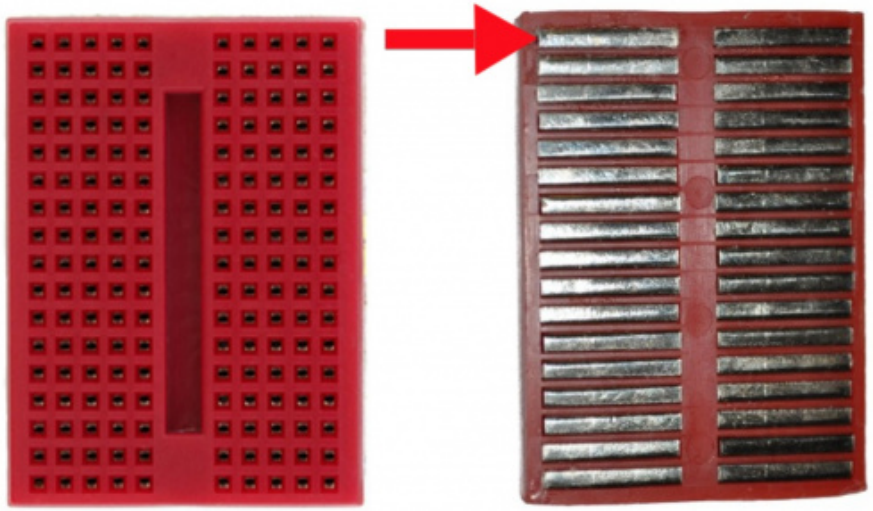
- CPX
- Breadboard
- Resistors
- **Jumper wires** and **alligator clips**



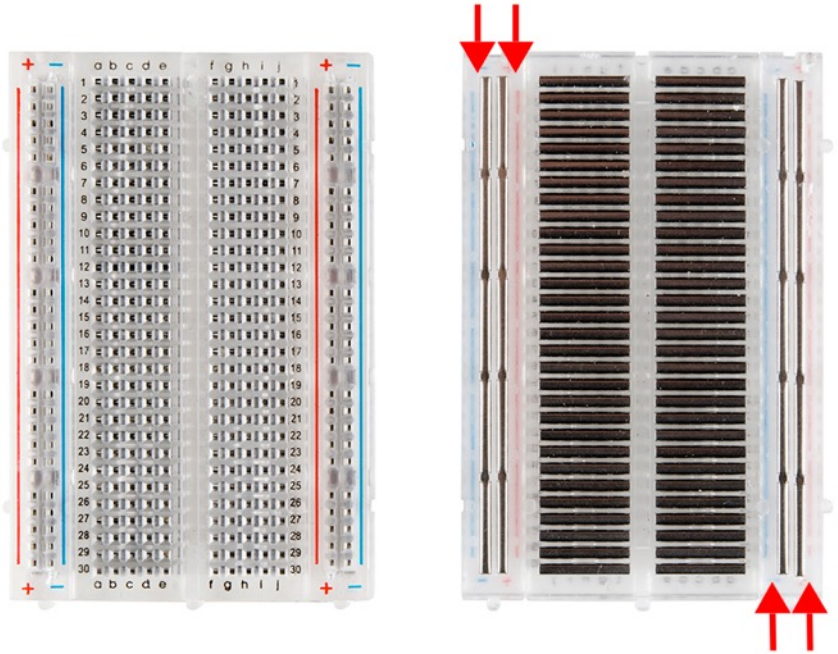
Breadboards



Fundamental tool to build circuits by allowing multiple connections between devices



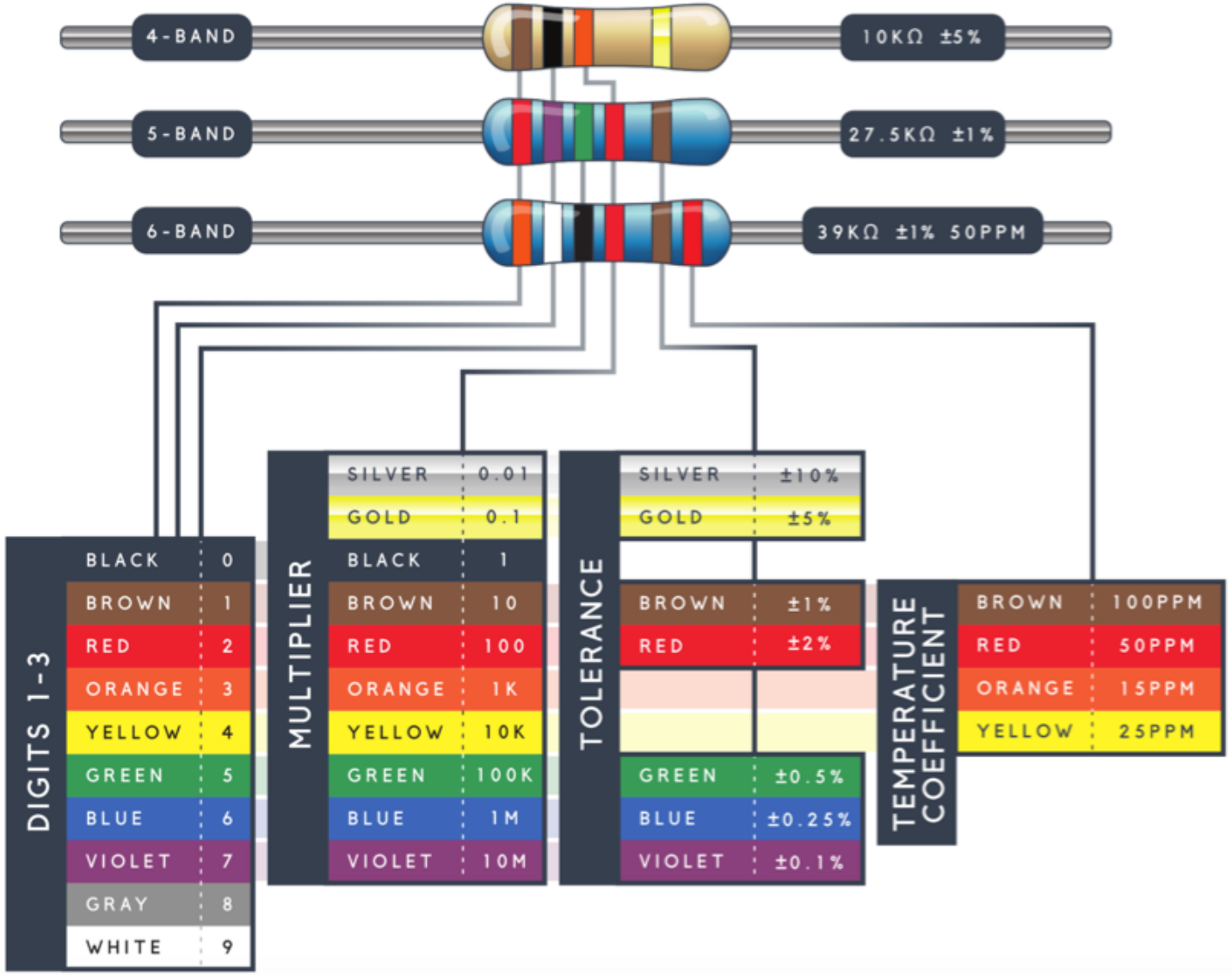
Metal connects each row beneath the plastic which allows for connections between each element plugged into the ***terminal strips*** (rows)



Most breadboards also **have *power rails*** which are two columns on each

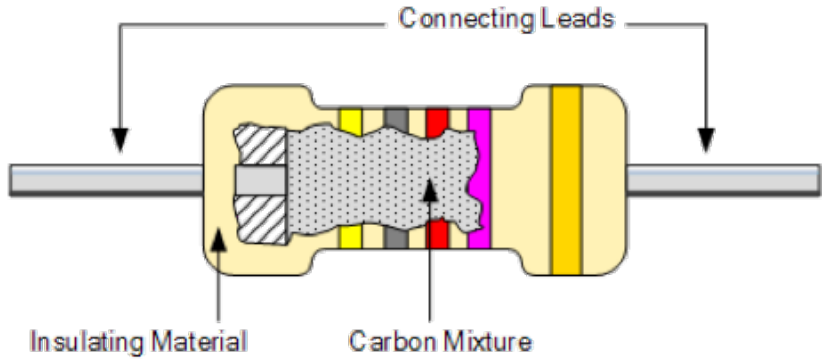
https://learn.sparkfun.com/tutorials/how-to-use-a-breadboard/all

Resistors



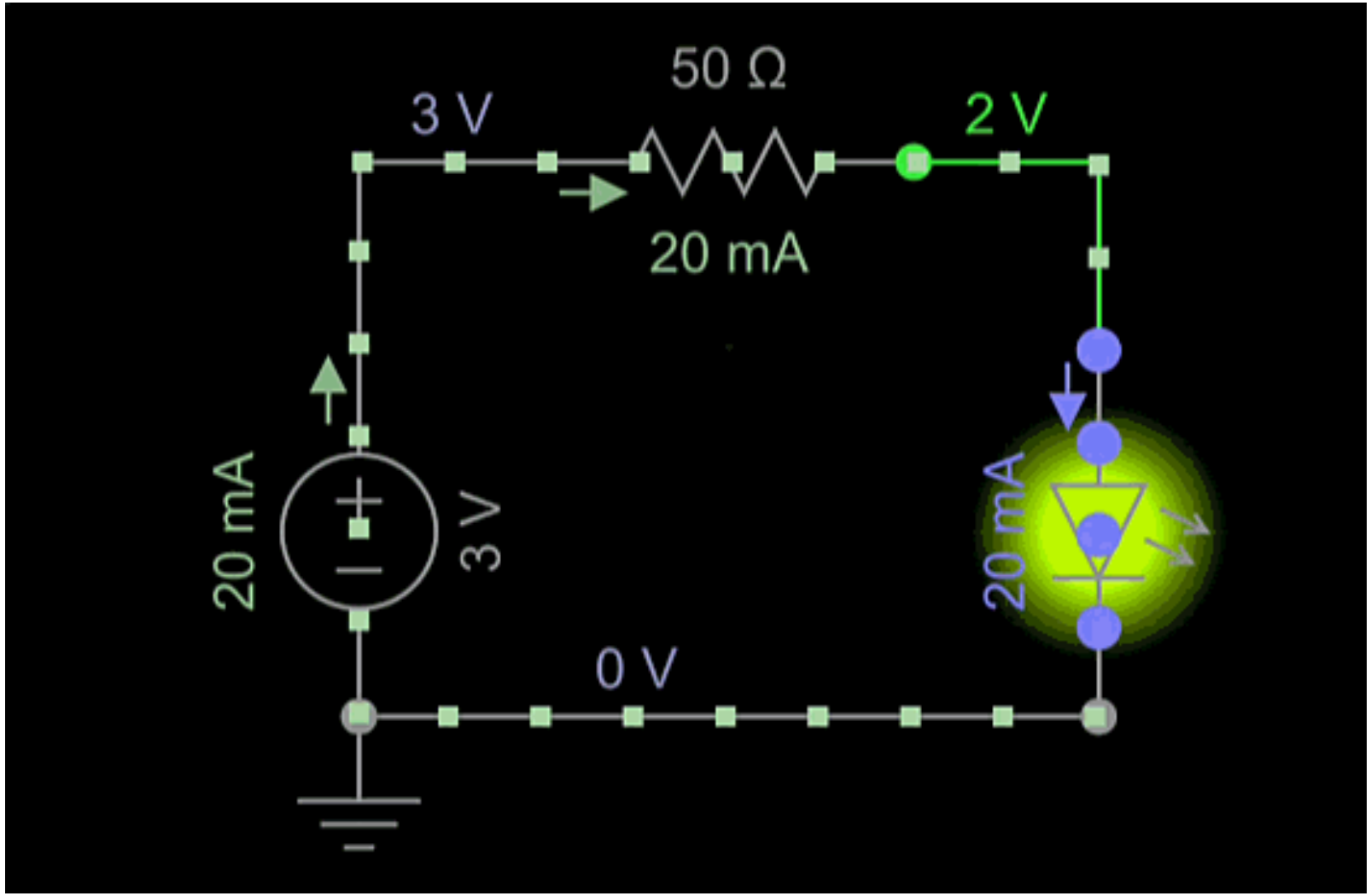
Electronic component that limits the flow of electrons through a circuit.

They are used to *step down* a power source that is too high for a given device



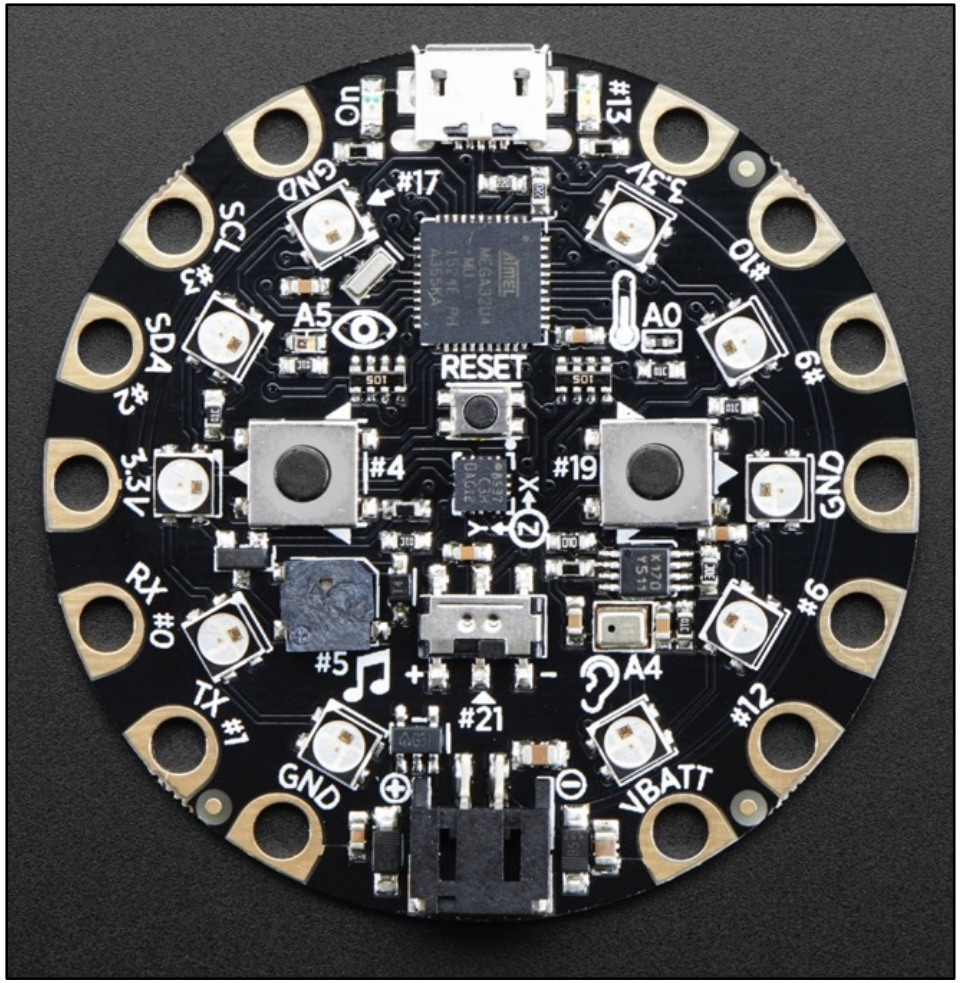
https://learn.sparkfun.com/tutorials/resistors/all

A drop of the voltage needed for a LED

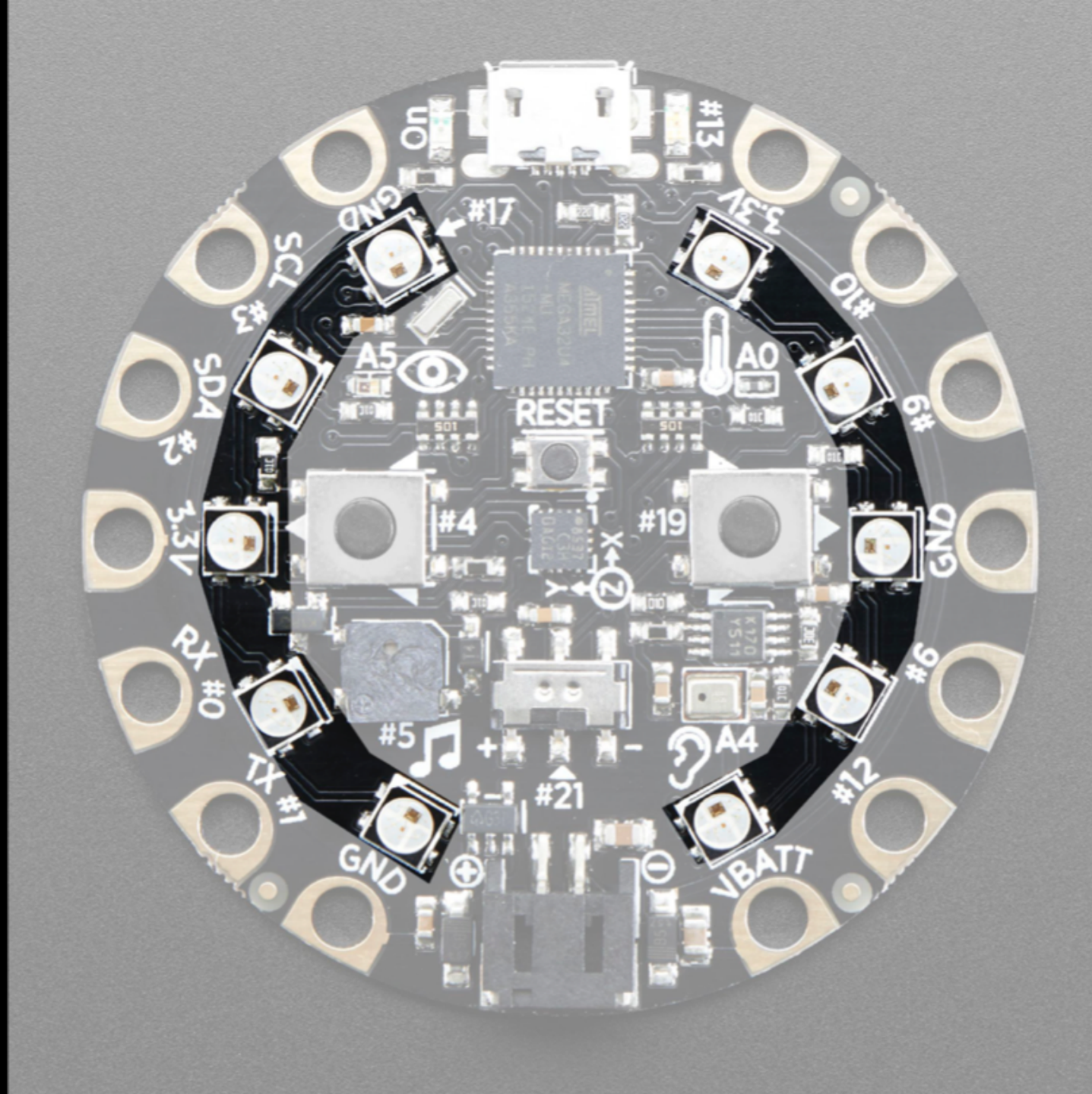


Adafruit Circuit Playground

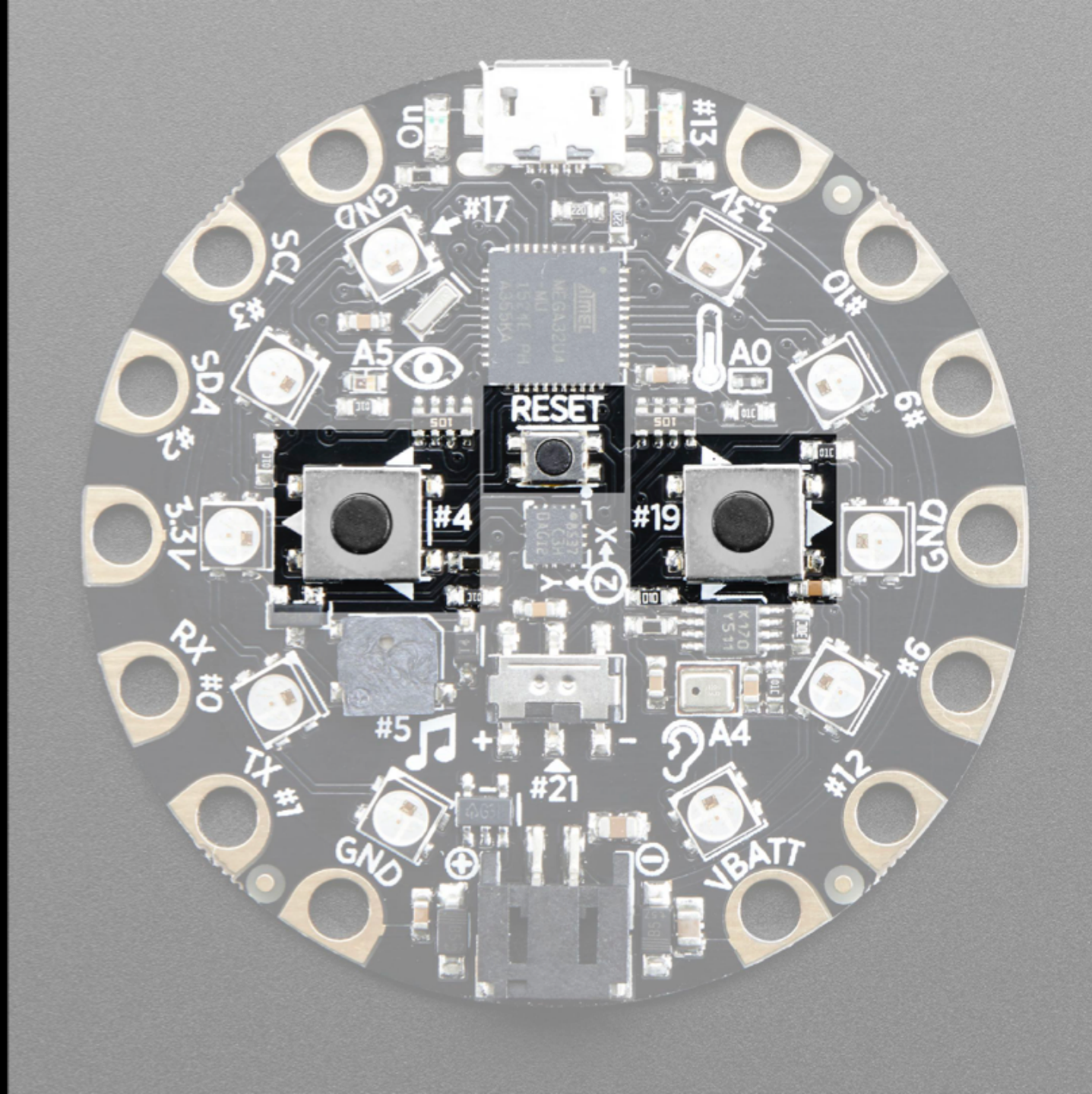
- (10) Mini NeoPixels (RGB controllable LEDs)
- (1) Sound sensor
- (1) Triple-axis accelerometer
- (1) Light sensor
- (1) Temperature sensors
- (1) Speaker (Buzzer)
- (2) Buttons
- (1) Slide switch



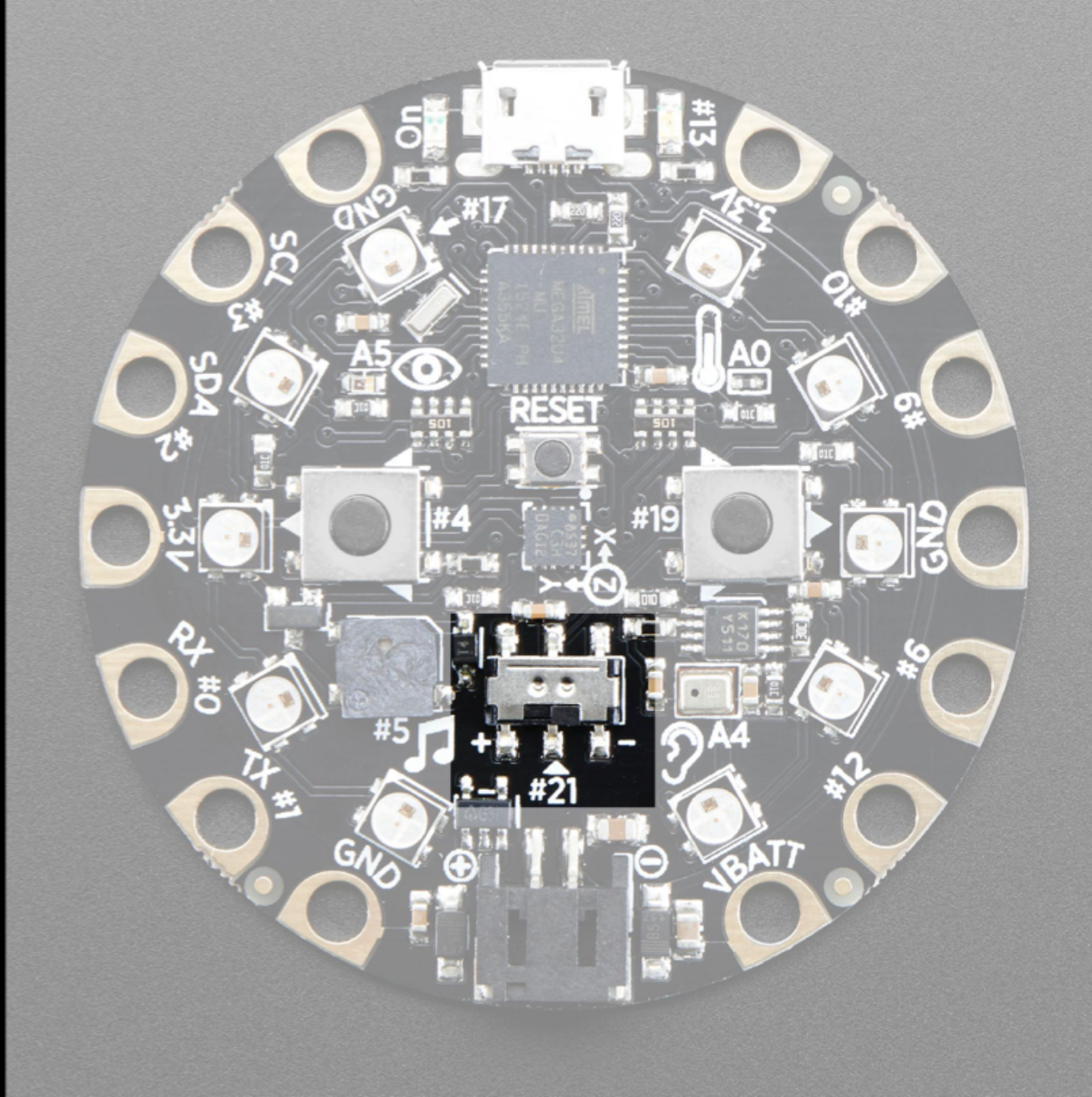
NeoPixels (RGB LEDs)



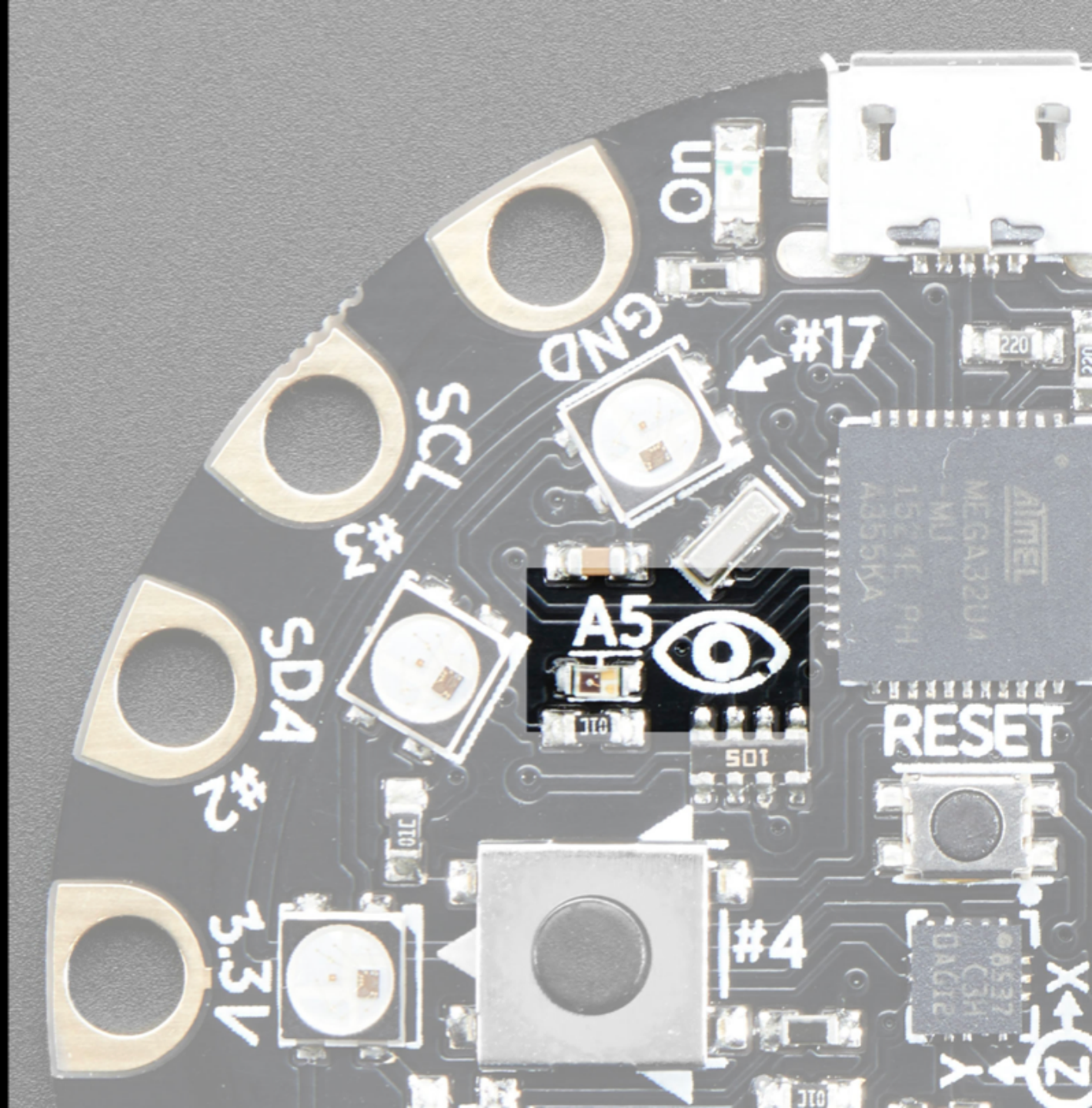
Pushbuttons



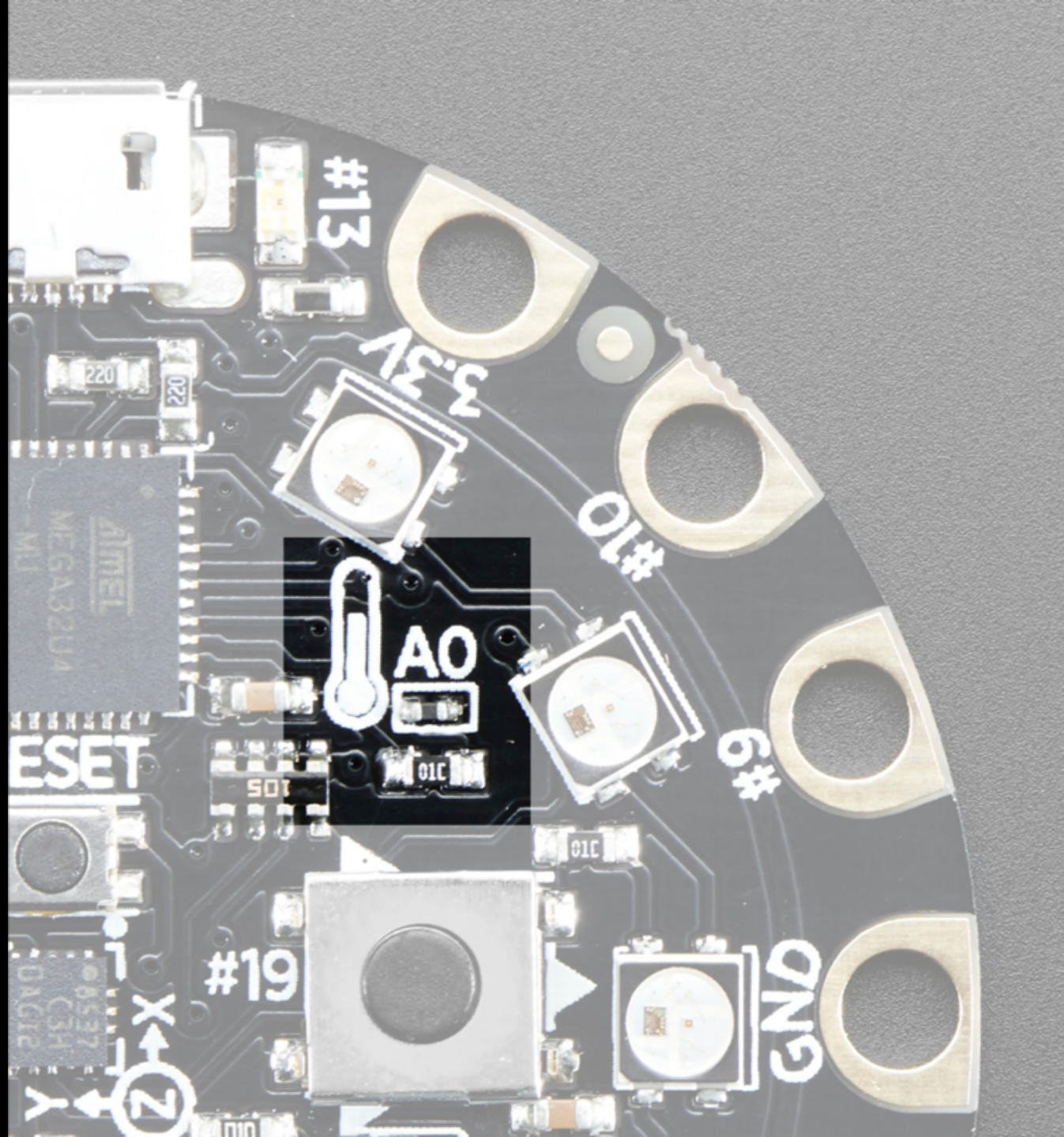
Slide Switch



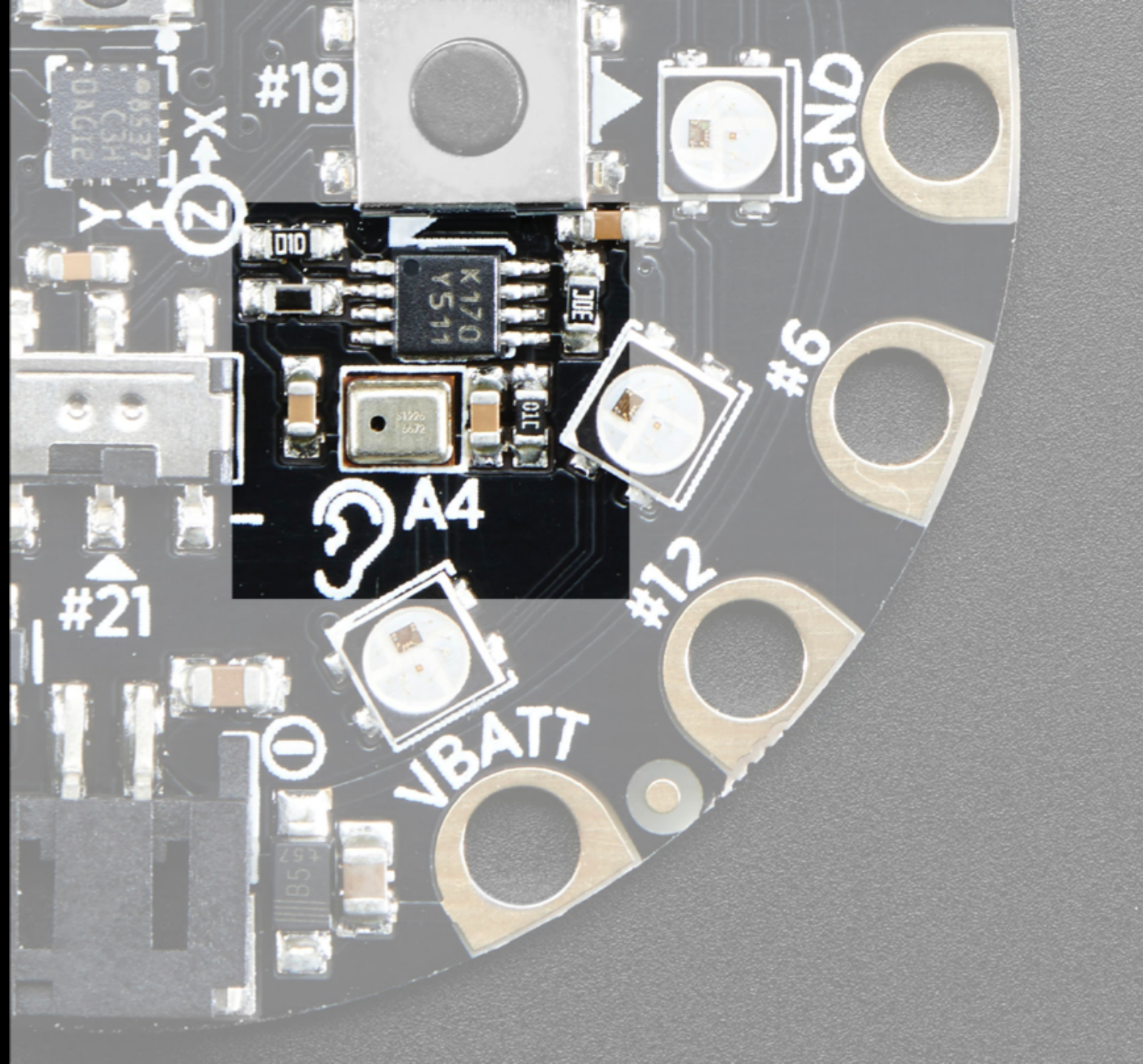
Light Sensor



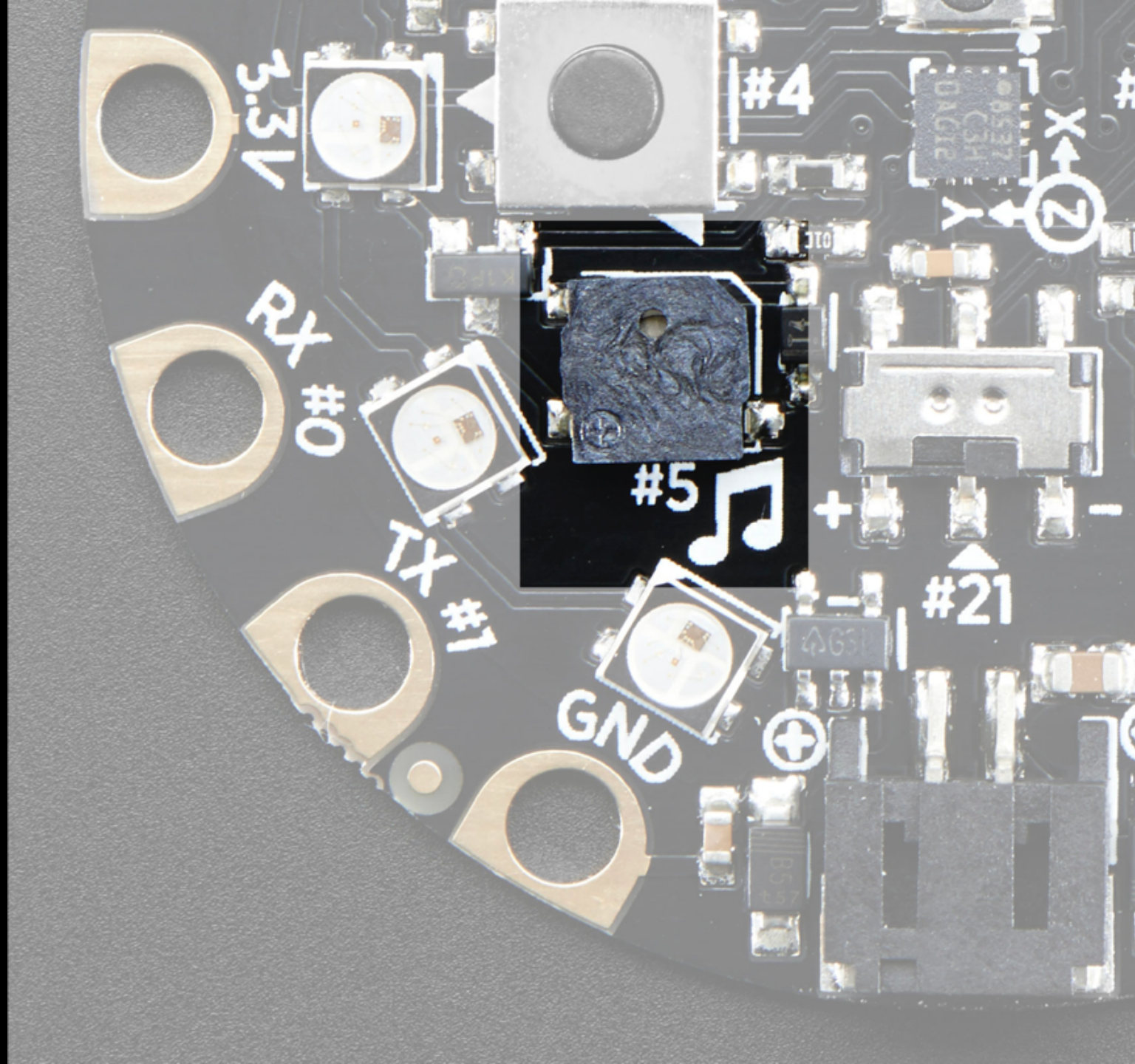
Temperature Sensor



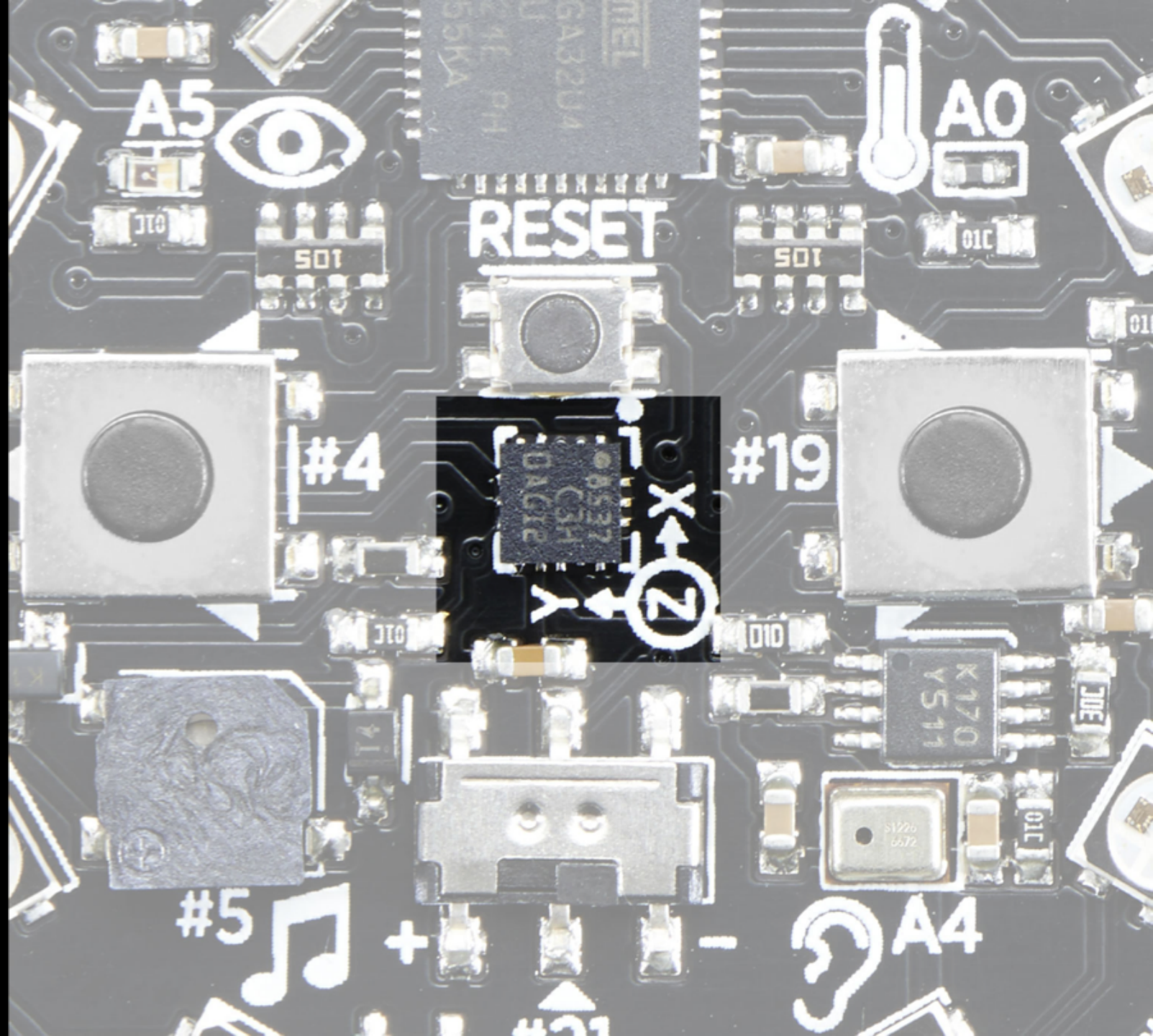
Sound Sensor



Mini Speaker

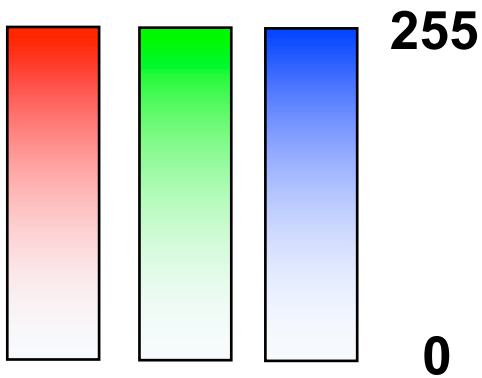


Accelerometer



Color Code: How to create colors in your image

- Combine parts of the three primary colors (red, green, and blue).
- Each of the primary colors has a value in the range from 0 to 255.



RGB Scale

Color	Color HEX	Color RGB
	#000000	rgb(0,0,0)
	#FF0000	rgb(255,0,0)
	#00FF00	rgb(0,255,0)
	#0000FF	rgb(0,0,255)
	#FFFF00	rgb(255,255,0)
	#00FFFF	rgb(0,255,255)
	#FF00FF	rgb(255,0,255)
	#C0C0C0	rgb(192,192,192)
	#FFFFFF	rgb(255,255,255)

Arduino Tutorial: External Actuators

Lab 2: External Actuators for Arduino with CircuitPython

A. Trahan, P.E.

September 19, 2019

Introduction

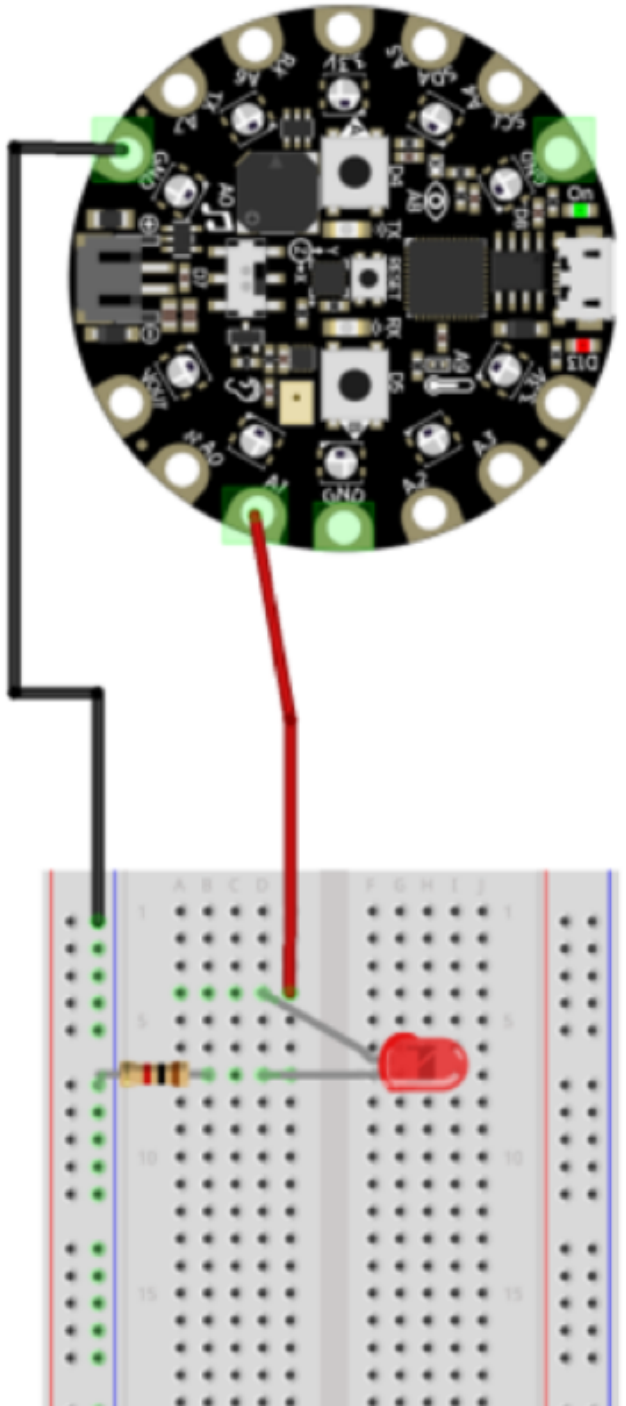
This lab provides an overview of using external actuators (output devices) on Arduino with Circuit Python. While there are many built-in actuators on the CPX, there may be additional actuators (like servos) that need to be attached for a given project. In addition, most other Arduino devices don't have as many built-in actuators, so all actuators for a project must be attached externally.

The CPX has a variety of "pins" for inputs and outputs. Instead of traditional pins, the CPX uses alligator clip mounts around the edge of the device. Since these are mapped to pins on the SAMD21 chip on the CPX (the "brain"), and they're actually pins on some Arduino devices, so references will often call them pins. The clip mounts on this device are generally called pads, so we will use that nomenclature. Some pin definitions from the SAMD21 are connected to on-board sensors and actuators, but this lab will focus on the externally accessible pads.

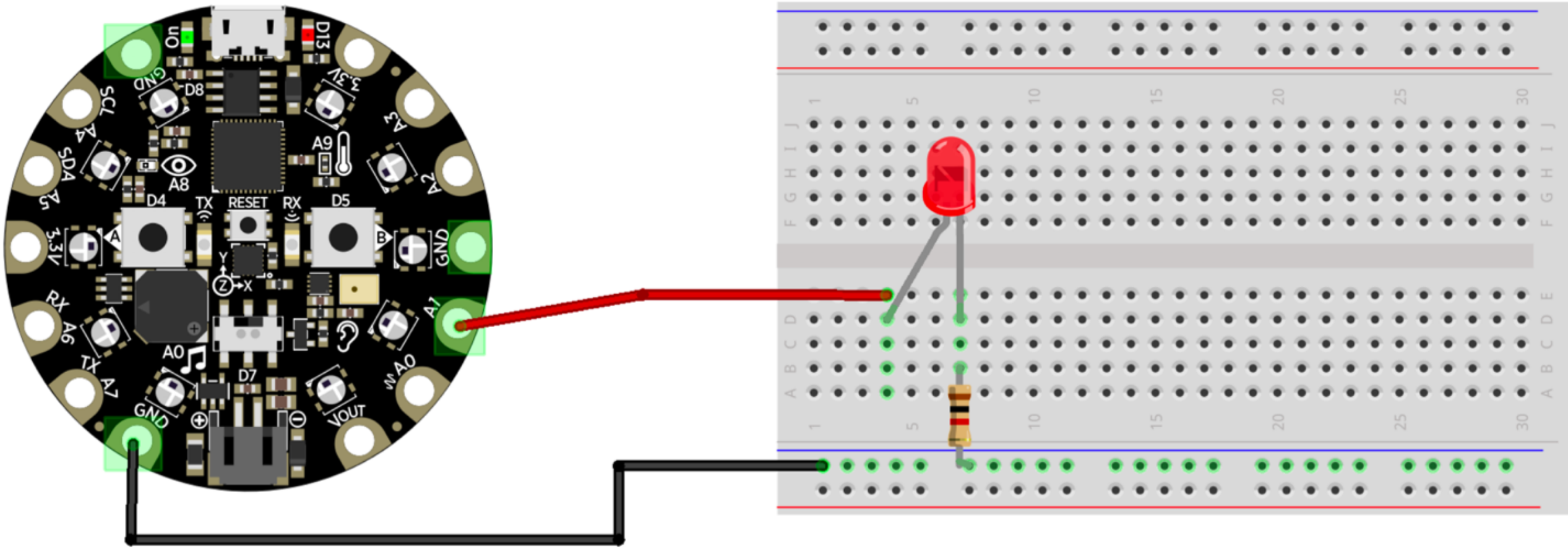
Full Arduino tutorial is available on Canvas in Tutorials Module **"Arduino CPX Tutorials" zip** Folder

1. Connecting Pads/Pins on CPX

1. Connect LED (**long leg**) to alligator clip, LED (**short leg**) in the main section of the breadboard (will connect to ground through resistor)
2. Connect same alligator clip to **CPX A1**
3. Connect second alligator clip to **GND** on **CPX** and to ground column (+ or -) of the breadboard (see figure on the left)
4. Add 100 ohm (**brown**) resistor to breadboard with one end in the ground column and other **in the same row as the LED**
5. Add a code from tutorial (**see page 4**) to Mu Interpreter and save as **code.py**
   Make sure you are using THE same indentations for the program!
6. Once saved (as **code.py**), the LED should start blinking
7. Try changing parameters in the code to see how it changes the actuation of the LED
8. Opening Serial and using CTRL+C will cancel the action

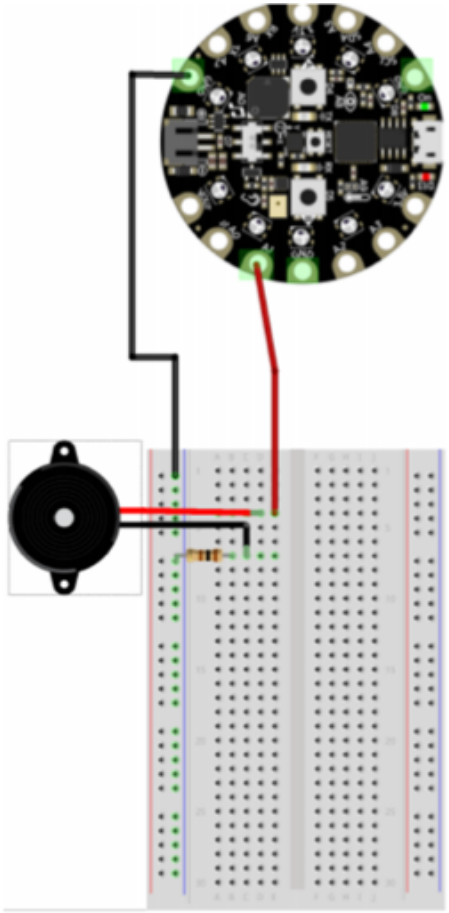


1. Connecting Pads/Pins on CPX (larger picture from the previous slide)



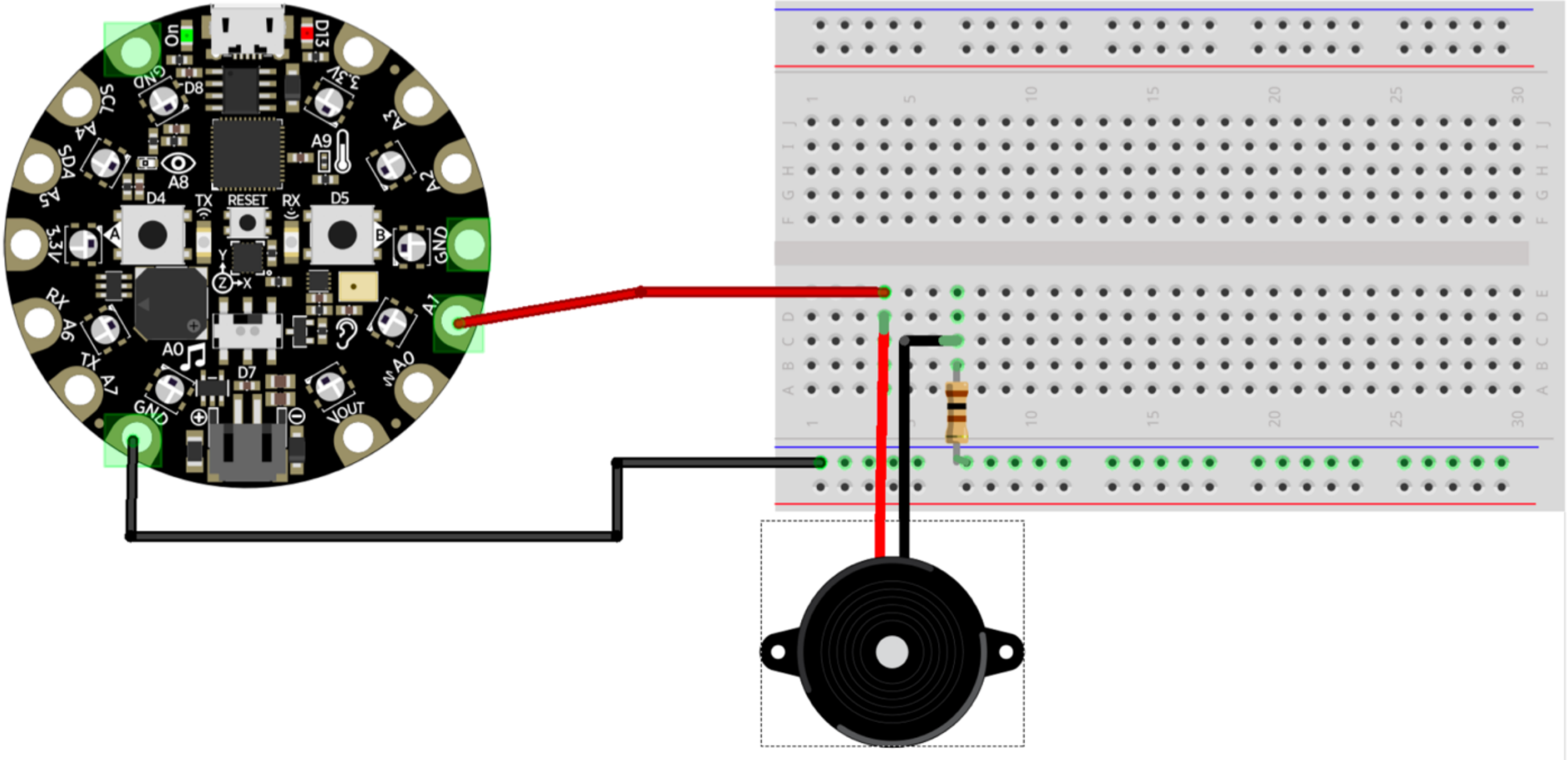
2.2. Using a Buzzer via Direct Pin Control

For buzzer tutorial part one we will essentially repeat the steps above. The buzzer has shorter pins though, so you likely want to use additional wires to get a good connection



1. **Disconnect the LED and replace this with the buzzer**
    1. Connect buzzer **red** end to alligator clip to A1, and **black** end to breadboard
    2. OR connect buzzer **long leg** to alligator clip to A1, and **short leg** end to breadboard
    3. Resistor connected in same row as buzzer **black** end
2. Add code from tutorial (**see page 7**) to the Mu Interpreter and save as **code.py**
3. Open Serial and CTRL+C to cancel the action (open then close serial if it isn't responding)

2.2 Using a Buzzer via Direct Pin Control (larger image from the previous slide)



2.3. Using a Buzzer via *SimpleIO (Optional)*

We can also use **a different library** to accomplish the same buzzer actuation

1. Open Serial and hit CTRL+C to find the version of your software (**see figure below**)
2. Download the corresponding ***bundle*** *(***Bundle version 6.X***)* from https://circuitpython.org/libraries
3. **Save bundle to desktop** then unzip folder (**extract all**)
   Note: You can put this folder in other areas but may run into "pathway too long" errors
4. In unzipped folder > lib > search: **simple.io** → copy and paste in **CIRCUITPY > lib**
5. Add new code from tutorial using **simple.io** and save as ***code.py***

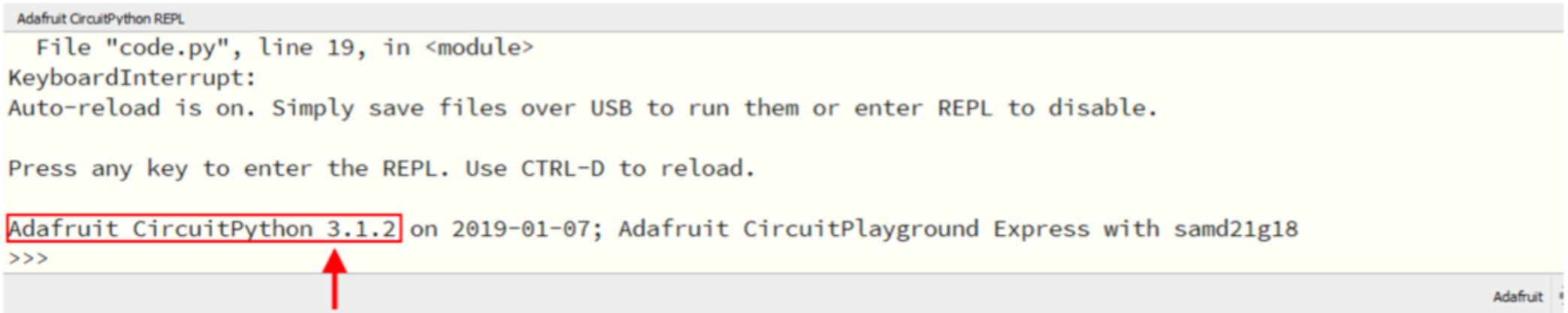


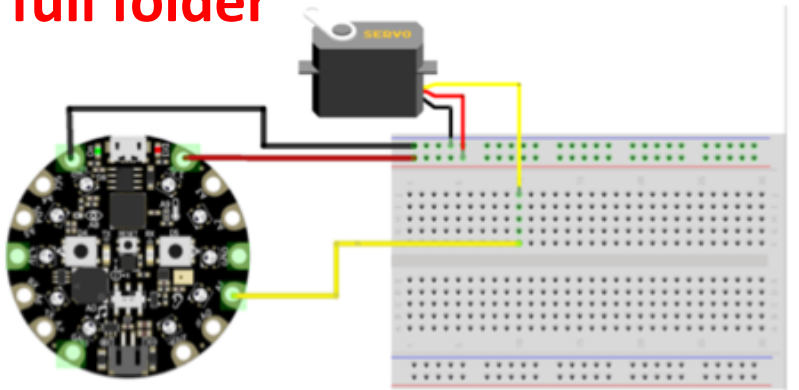
Figure 3: Finding the CircuitPython version with Mu

3.1 Responding to Sensors on the CPX (Optional)

This example demonstrates how **the amount of light changes the position of servo**

Our first several examples relied on set schedules for actuator behavior. Here we will learn how we can use inputs from sensors to control external actuators

1. **First, we will need more software from the bundle—copy the full folder *adafruit_motor* into CIRCUITPY > lib**
2. Add **a fin** to the servo motor top
3. Connect the servo motor to CPX using the following:
    1. **Brown** is ground, **connect to GND**
    2. **Red** is power, connects to **3.3V**
    3. **Orange** is PWM (Pulse-Width Modulation, controls the motor), **connect to A1**
4. Input the new code from the tutorial and save as *code.py* on your CPX
    1. If not working, check Serial w/ Ctrl+D
    2. If it says not working because safe mode, eject CPX and reload



We can do this one without the breadboard!

https://learn.adafruit.com/adafruit-circuit-playground-express/circuitpython-servo

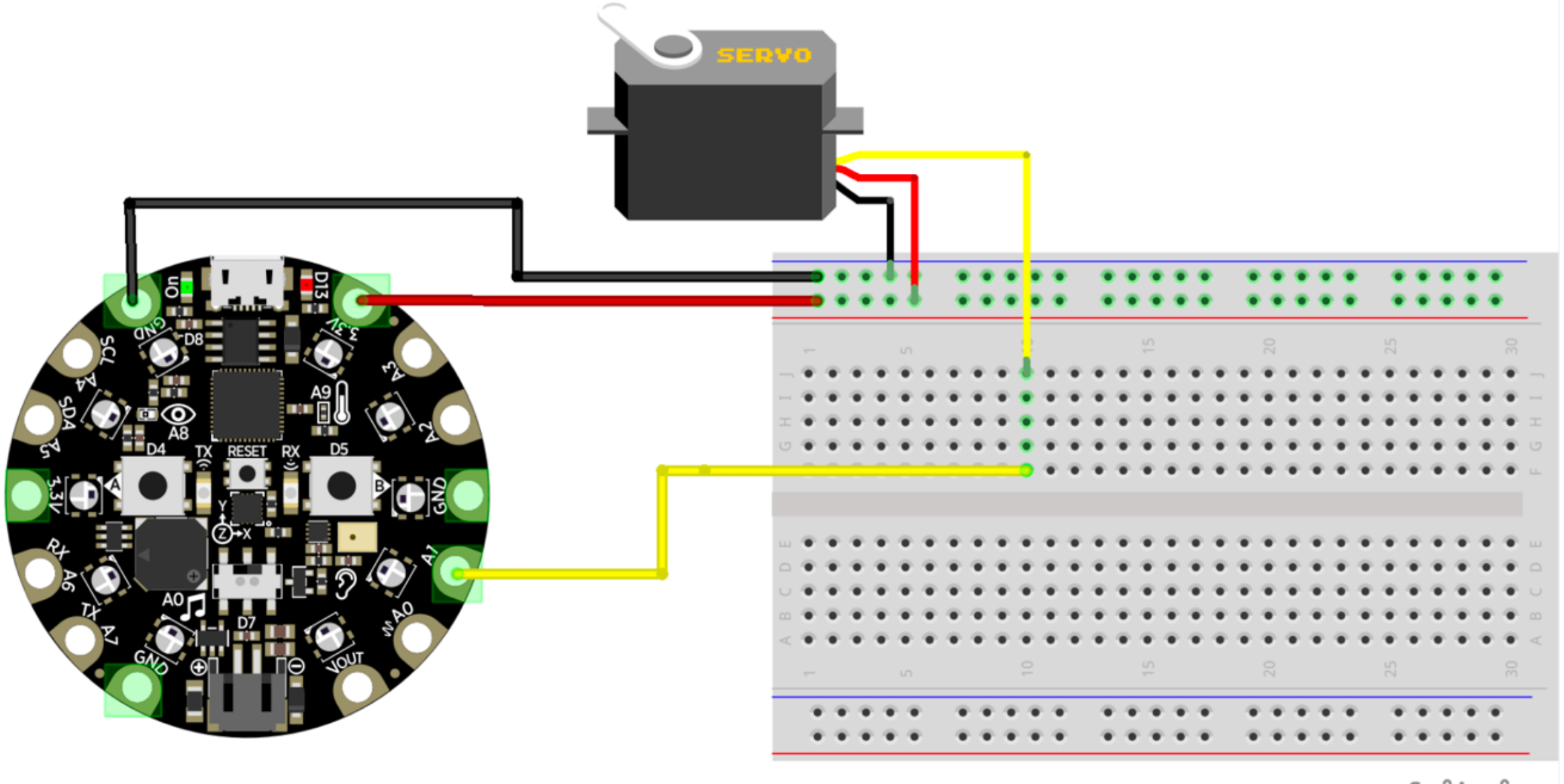
3.1 Responding to Sensors on the CPX (Optional)

This example demonstrates how **the amount of light changes the position of servo. There are minor changes into the Tutorial program. Please consider to check this program!**

```
import board
import pulseio
from adafruit_circuitplayground.express import cpx
from adafruit_motor import servo
from time import sleep
# Setup Section
pwm = pulseio.PWMOut(board.A1, duty_cycle=2 ** 15, frequency=50)
servo = servo.Servo(pwm, min_pulse=750, max_pulse=2600)
# Function Section
def light_to_servo_pos(light):
    light_max = 320
    return 180 - ((light/light_max) * 180)
# Loop Section
while True:
    servo.angle = light_to_servo_pos(cpx.light)
    print(servo.angle)
    sleep(0.5)
```

3.1 Responding to Sensors on the CPX (Optional)

This example demonstrates how **the amount of light changes the position of servo**



3.2 Another example that shows how **servo rotates between 0 and 90 degrees**

Our first examples all relied on set schedules for actuator behavior. Here we will learn how we can use inputs from sensors to control external actuators

1. **First, we will need more software from the *bundle*—copy the full folder *adafruit_motor* into CIRCUITPY > lib**
2. Add **a fin** to the servo motor top
3. Connect the servo motor to CPX using the following:
    1. **Brown** is ground, **connect to GND**
    2. **Red** is power, connects to **Vout**
    3. **Orange** is PWM (Pulse-Width Modulation, controls the motor), **connect to A2**
4. Input the new code (**that Fabian sent, see next slide**) and save as *code.py on your CPX*
    1. If not working, check Serial w/ Ctrl+D
    2. If it says not working because safe mode, eject CPX and reload

We can do this one without the breadboard!

https://learn.adafruit.com/adafruit-circuit-playground-express/circuitpython-servo

3.2 Program for Servo Motor to show **how servo rotates between 0 and 90 degrees**

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

# create a PWMOut object on Pin A2.
pwm = pulseio.PWMOut(board.A2, duty_cycle=2 ** 15, frequency=50)

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

while True:
    for angle in range(0, 90, 5):  # 0 - 180 degrees, 5 degrees at a time.
        my_servo.angle = angle
        time.sleep(0.05)
    for angle in range(90, 0, -5): # 180 - 0 degrees, 5 degrees at a time.
        my_servo.angle = angle
        time.sleep(0.05)
```