MAKE THIS! ELECTRONICS PROTOTYPING using ARDUINO

INTRODUCTION

Welcome to the CHI course *Make This! Electronics Prototyping using Arduino.*

The course comprises two 80-minute sessions. In the first session, you'll learn about electronics prototyping by lighting LEDs and controlling them using a few digital and analog sensors.

In the second, you'll build a small paper robot to see how a microcontroller, software, sensors and actuators work together on a single project.

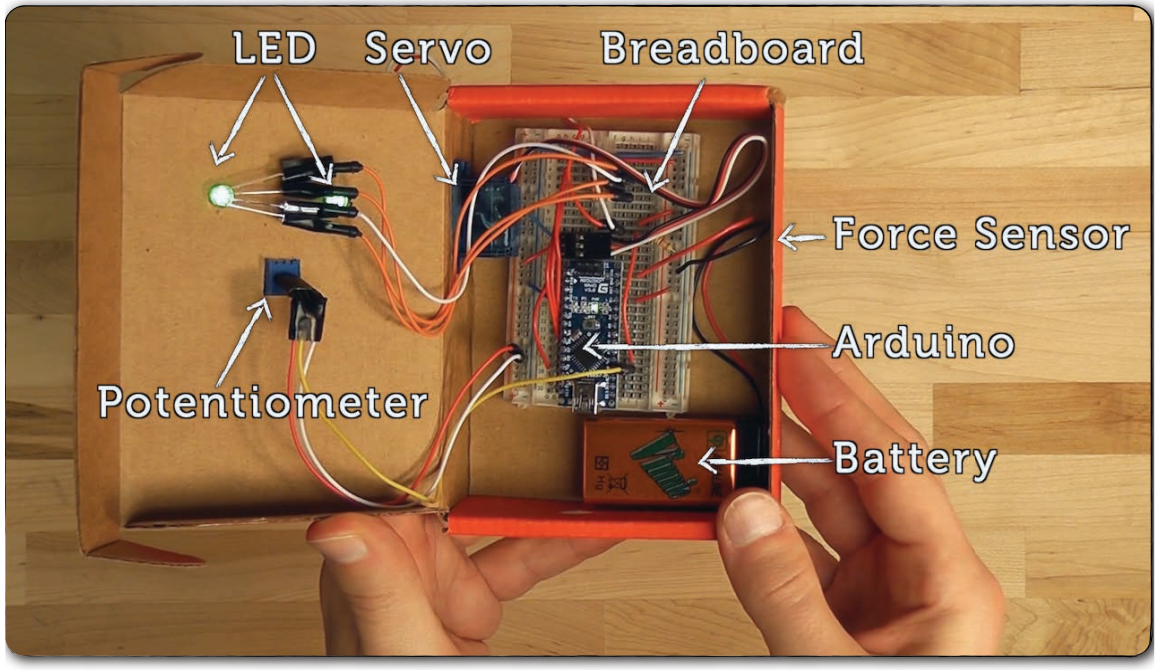
We'll cover Parts I & II of this guide today. Try Part III if you're experienced.

I: Electronics Prototyping
II: Paper Robot
III: The Advanced Section
IV: Make (More Than) This!

David Sirkin, Nik Martelaro, Wendy Ju
Stanford University, CA, USA

PAPER ROBOT

Here's the paper robot project that you'll be building in the second half of the session.



MAKE THIS! ELECTRONICS PROTOTYPING using ARDUINO

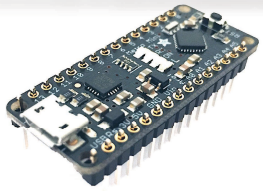
PART I — ELECTRONICS PROTOTYPING

1. Your Kit
2. Arduino
3. Breadboard
4. LEDs
5. Sensors
6. Voltage Dividers

ELECTRONICS PROTOTYPING KIT

To get started, take out the first 4 items listed below: breadboard, Arduino, USB cable and jumper wires.

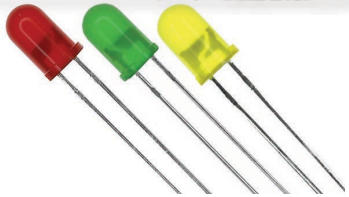
Breadboard: Wires plug into its sockets.



Arduino Metro Mini: Microcontroller.

USB Cable: A-type to micro-B connector.

Jumper Wires: To connect components.



LED: Electroluminescent Semiconductor.

RC Servo: Motor holds at a set position.

Button: Normally-open momentary type.

Potentiometer: Resistance of 10K Ohm.

FSR: 250-30K Ohm force-sense resistor.

Photocell: Changes resistance with light.

Battery & Clip: To power the Arduino.

Box: We use it for the body of the robot.

INSTALL THE ARDUINO SOFTWARE

If you haven't already done so, download the Arduino software now.

1 Install Arduino

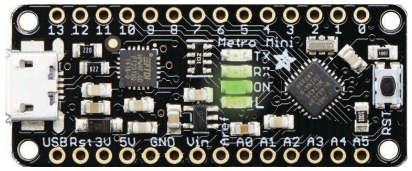
Visit the Arduino Software page and download version 1.6.8 for your OS.

http://arduino.cc/en/main/software

Unzip the download, making sure to preserve the structure of any folders.

Copy the application to a location on your system that you prefer.

Connect the Arduino to your laptop using the USB cable. The green power LED on the top of the Arduino should light up.



2 Install the Driver

Windows users have to install a device driver; OSX users shouldn't, and can move on to step 3.

If the device installer doesn't launch automatically, first try the steps at the Arduino driver install page.

Otherwise, open the Windows Device Manager.

Start > Control Panel > Hardware

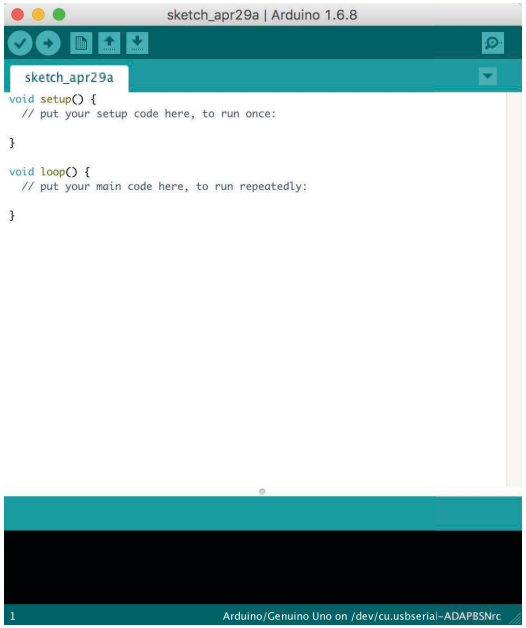
Find the Arduino Leonard listing, right click on it, and select Update Driver Software.

At the next screen, select the Browse My Computer alternative, and navigate to your new Arduino software.

Select the drivers folder, and click Next to complete the installation.

3 Launch Arduino

Launch the Arduino application. It will open to the template of a new, empty sketch. Here's what it looks like.



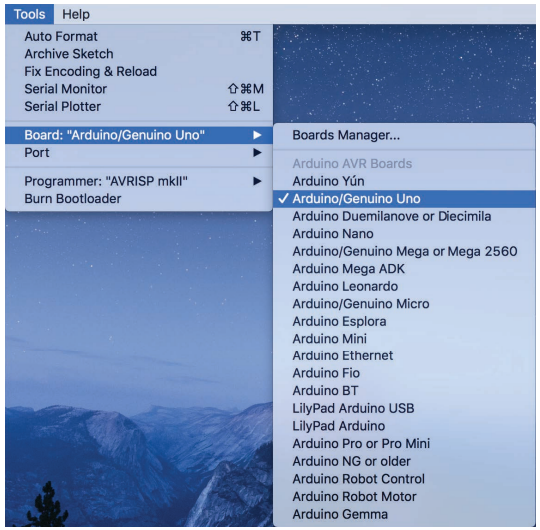
SETUP THE ARDUINO SOFTWARE

Now we'll set up the software to work with the Arduino Micro board.

1 Set the Board

The programmer needs to know what hardware you're using, so select the Arduino Micro using the menu bar.

Tools > Board > Arduino/Genuino Uno



2 Set the Port

Next, select a port for the serial device.

Tools > Serial Port

On Windows, the port to choose will typically be COM3 or higher.

On OSX, the port to choose will resemble /dev/cu.usbserial-ABCD, only the final sequence will include different characters.

To check that you've chosen the right port, disconnect the Arduino and reopen the Serial Port menu.

The entry that has disappeared is the correct port. Reconnect the board and select that serial port.

3 Interface

Do you see the 5 buttons at the top left of the interface?

Hover your mouse over each of these buttons to highlight its function.

You'll mostly use Verify (the check mark) to compile sketches, and Upload (the arrow) to first compile sketches, and then upload them to the Arduino.

ARDUINO'S "HELLO WORLD!"

Open, compile, upload, then edit the Blink sketch to start learning how to program Arduino.

1 Flash "Blink"

Open the Blink sketch example by navigating through the top menu bar.

File > Examples > 01.Basics > Blink

Compile and upload the sketch, using the arrow button. After a few seconds, the yellow RX and TX LEDs on the Arduino should flash quickly.

If the upload is successful, the message "Done uploading" will appear in the status bar at the bottom.

You should see the red LED on the top of the Arduino blink on for 1 second, then blink off for 1 second.

2 Update "Blink"

Try to modify the Blink sketch so that the LED flashes at twice its original rate.

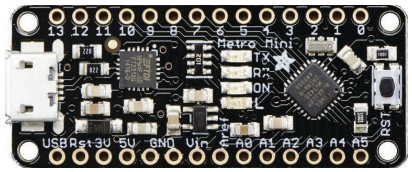
If you're not sure how to change the flashing rate, find the line below, and edit it so the delay is shorter. Then re-upload your modified sketch.

```
delay(1000); // wait for a second
```

When you can readily change the way that the LED flashes, you're ready to move on with the tutorial.

Keep the Blink sketch open for now.

The brain of the Arduino is its micro-controller. It has a similar role as the CPU in your laptop, only it's designed to be used with sensors and actuators. The Metro has a 16 MHz ATmega328, made by Atmel.



Arduino programs are called sketches. Sketches are just C++ programs. Every sketch includes 2 special functions: `setup()` and `loop()`.

`setup()` runs once at the start of the program, and `loop()` runs forever after that.

Blink

```
// the setup function runs once when you press reset or
power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off
  delay(1000);                       // wait for a second
}
```

BREADBOARD WIRING OVERVIEW

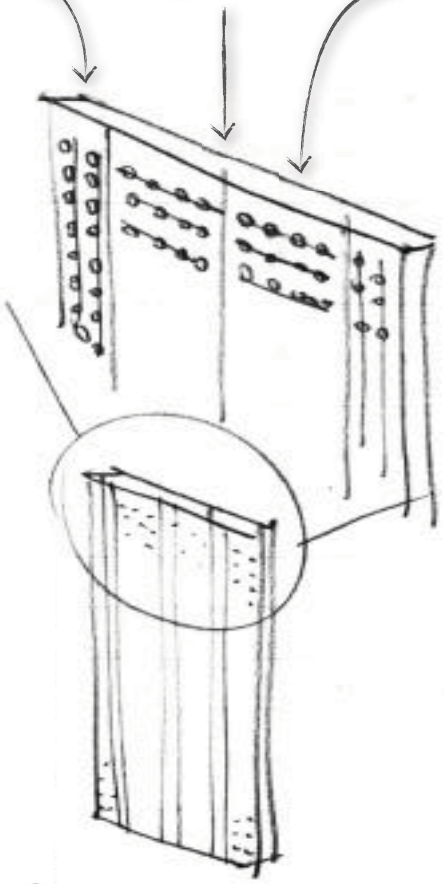
Your breadboard has a few regions: power and ground rails on the sides, and circuit-building areas in the center.

Power & Ground Rails

The holes along any **column** (either red or blue) are connected to every other hole in that column (only!).

The rails marked by **red** stripes are used for power, and the rails marked by **blue** are used for ground.

Separator



Circuit Building Area

The holes along any **row** are connected to every other hole in that row (only!).

The separation that runs down the middle of the board isolates the rows on the left side from the rows on the right.

The Arduino receives power from your laptop through the USB cable. It then makes that power available to circuits on your breadboard through its 5V, 3.3V and GND pins.

When changing any circuit on the breadboard, it's a good idea to unplug the USB cable from the Arduino, so you don't power something by mistake, so **unplug it for now**.

SETUP THE BREADBOARD

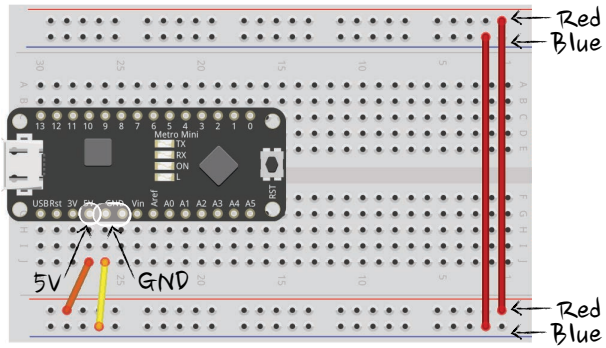
Now we'll configure the board for convenient access to power and ground through the rails.

1 Socket the Arduino

Unplug the USB cable from the Arduino, then socket it into your board as shown, with the USB port facing the outside.

Make sure it straddles the separation that runs down the middle of the board.

If it doesn't, then the pins on one side of the Arduino would connect to those on the other side, mixing their signals.



2 Wire the Board

A: Using 2 long jumper wires, connect the power and ground rails across opposite sides of the board, as shown on the right side of the image. Red connects to red, blue to blue.

Note: The wires may be red or some other color. Pay attention to the rails.

B: Using 2 short jumper wires, connect the Arduino's 5V pin to the red power rail, and either of the Arduino's 2 GND pins to the blue ground rail.

Together, these connections create a convenient power supply along both rails for your circuits.

Double-check your wiring against the image on the left to make sure that you got everything right.

3 Power the Board

Now plug the USB cable back into the Arduino. You should see the green power LED on the Arduino turn on.

If you don't, you have a short circuit, and should disconnect the USB cable and fix the circuit right away!

If everything looks good, you're up and running, and ready to light some LEDs.

A short circuit is when the GND and 5V pins of the Arduino connect together, and it can destroy the Arduino. :-(

We recommend you unplug the USB cable every time you change a circuit.

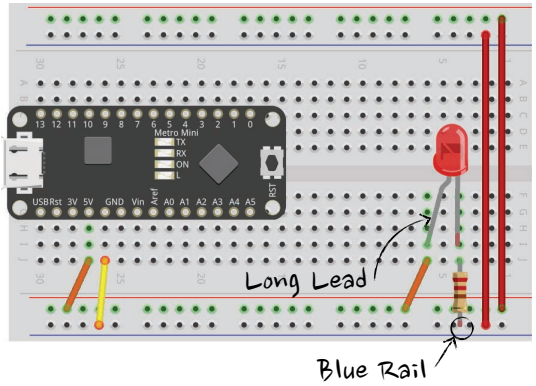
BASIC LED CIRCUITS 1: LED Always On

Let's build 3 LED circuits: **1) LED always on**, 2) LED control by button, and 3) LED control by program.

1 Add a Resistor

First, unplug the USB cable from the Arduino, then build the circuit shown below. Use a 220 Ohm resistor.

Resistors have 4 color bands to indicate their resistance. For 220 Ohm resistors, the bands are: **red**, **red**, **brown** and (farther away) **gold**.

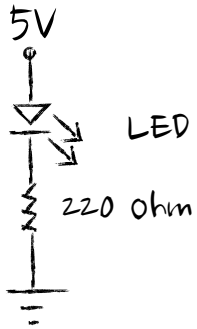


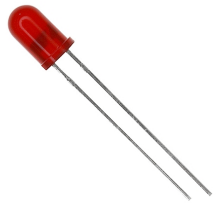
2 Add an LED

LEDs are diodes, and diodes only work (as intended) when oriented one way.

Connect the longer lead (the anode) toward power, and the shorter lead (the cathode) toward ground. In the diagram at left, the longer lead has a bent knee.

Note: **The long arm reaches for power!**





Diffuse Red LED

3 Power the Arduino

Now plug the USB cable back into the Arduino. Your new LED should light up.

Notice that the onboard LED is still flashing. That's because the Arduino is still running the Blink program from earlier.

It's important to always place a resistor in series with any LED, to limit the current in the LED to a safe value.

We include a circuit diagram alongside each drawing. Circuit diagrams are readable and portable, so it's worth being (somewhat) familiar with them.

BASIC LED CIRCUITS 2: LED Control by Button

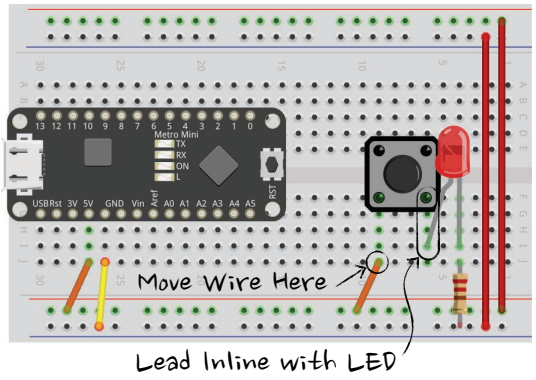
Let's build 3 LED circuits: 1) LED always on, **2) LED control by button**, and 3) LED control by program.

1 Add a Button

Unplug the USB cable from the Arduino, then insert a pushbutton into the circuit, as shown below. Press the button's leads fully into the board.

Make sure that the button crosses the center separator, as shown on the right.

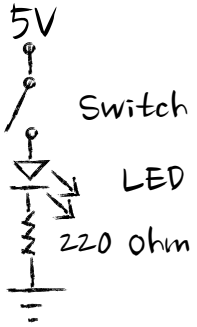
Note that you have to move the short wire that connected the LED to power.



2 Power the Arduino

Now plug the USB cable back into the Arduino, and check that the button works: the LED should light only when you press the button.

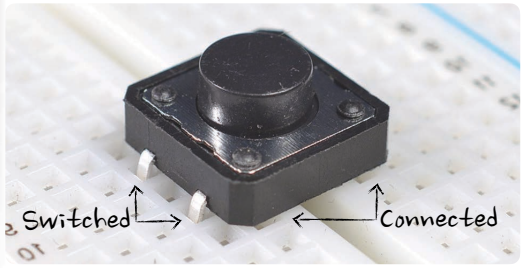
Congratulations, you've made a light switch.



12mm Pushbutton

If your button doesn't work, check that it's oriented as shown below. You might need to rotate it by 90 degrees.

A typical pushbutton, when pressed, connects the 2 pins on one side to the 2 pins on the other side.



BASIC LED CIRCUITS 3: LED Control by Program

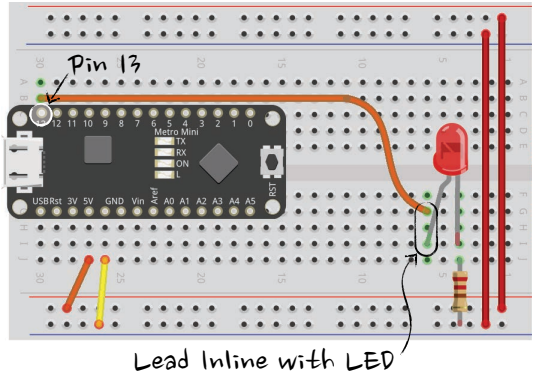
Let's build 3 LED circuits: 1) LED always on, 2) LED control by button, and **3) LED control by program**.

1 Rewire the Board

Unplug the USB cable from the Arduino, then rewire the board shown below.

First remove the button and the short wire that powered the circuit before. Then connect a long jumper wire to Arduino pin 13. This pin will now power the LED.

Reconnect the USB cable. The external LED should flash with the onboard LED.



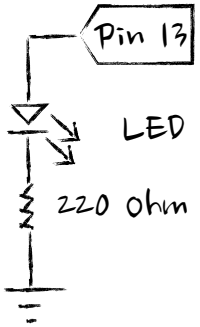
2 Revisit "Blink"

Return to the Blink sketch that's open on your laptop. The following line sets pin 13 to output, or "source," voltage.

```
pinMode(13, OUTPUT);
```

And the following lines tell the Arduino to send 5V, and later OV, out that pin.

```
digitalWrite(13, HIGH);
digitalWrite(13, LOW);
```



The Metro Mini's red onboard LED is internally connected to pin 13. You can also connect an external LED to the same pin: they'll behave the same.

Why is Blink still running? The most recently loaded sketch is stored in flash memory, so it's remembered even after you power down the Arduino.

Blink (pin 13)

```
// the setup function runs once when you press reset or
power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(13);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(13);     // turn the LED on
  delay(1000);          // wait for a second
  digitalWrite(13);     // turn the LED off
  delay(1000);          // wait for a second
}
```

FADING THE LED

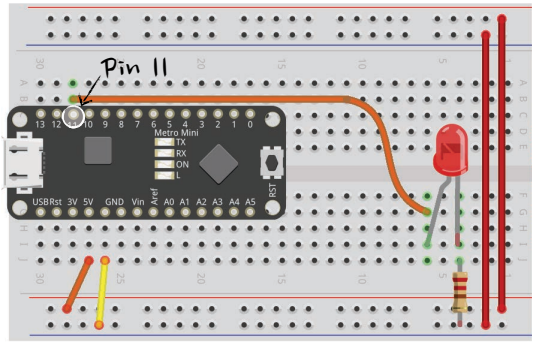
The last way to control an LED is to vary its brightness, using pulse width modulation, or PWM.

1 LED Brightness

What about those "breathing" LEDs during sleep mode on many laptops?

The fading light is done using pulse width modulation, or PWM. The LED is toggled on and off very quickly: say, 1,000 times per second. Much faster than your eye can follow.

The percentage of time that the LED is on (called the duty cycle) controls its apparent brightness.



2 Rewire the Board

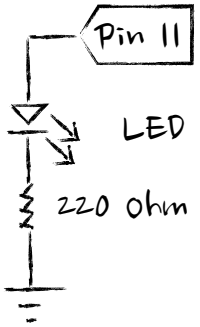
Update your circuit so the jumper wire connects to pin 11, as shown on the left. Then open the Fade sketch.

File > Examples > 01.Basics > Fade

The first line (after the comment) sets the variable named `led` to pin 9.

```
int led = 9;
```

But your LED is connected to pin 11. Edit the line above to set `led` to pin 11.



3 Flash "Fade"

Now upload your sketch and check that the LED fades on and off.

Try changing a few parameter values to better understand how things work. Start with the last line, for example.

```
delay(30);
```

To control an LED using PWM, you have to connect it to one of the pins that supports PWM (not all do!).

On the Metro Mini, these are pins 3, 5, 6, 9, 10 and 11.

You also have to use `analogWrite()` (which enables PWM) in your sketch, rather than `digitalWrite()`.

Fade

```
int led = 11;           // the PWM pin the LED is attached to
int brightness = 0;    // how bright the LED is
int fadeAmount = 5;    // how many points to fade the LED by

// the setup routine runs once when you press reset:
void setup() {
  // declare pin 9 to be an output:
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  // set the brightness of pin 9:
  analogWrite(led, brightness);

  // change the brightness for next time through the loop:
  brightness = brightness + fadeAmount;

  // reverse the direction of the fading
  if (brightness <= 0 || brightness >= 255) {
    fadeAmount = -fadeAmount;
  }
  // wait for 30 milliseconds to see the dimming effect
  delay(30);
}
```

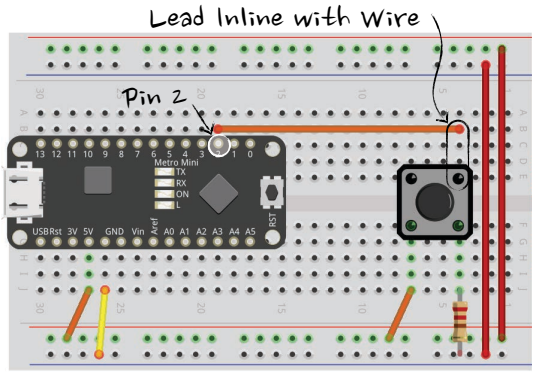
SENSING A BUTTON PRESS

The Arduino can detect different voltages, as well as provide them. Here's how.

1 Replace the Button

It's just as easy for your Arduino to sense input as it is to control output. Wire your board with the button circuit shown below.

Check that the wire that connects from the button to the Arduino is on pin 2.

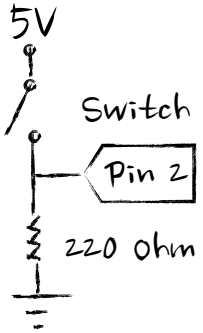


2 Flash "Button"

Open the Button sketch, look over how it works for a moment, and then run it.

File › Examples › 02.Digital › Button

Make sure that the Arduino's onboard LED lights when you press the button.



It's okay to connect the jumper wire to the "opposite" side of the button: recall that the 2 leads on the same side are connected to each other.

Pins as Input or Output

In the earlier circuit, the LED turned on because the button directly switched a supply of 5V to it.

Here, pushing the button sends a 5V signal to pin 2. The sketch reads this, and sends 5V to pin 13, which is attached to the onboard LED.

Any Arduino pin can be made an input (or an output) by setting its mode in the `setup()` function, like the following.

```
pinMode(buttonPin, INPUT);
```

The resistor is required for the button circuit to work. Without it, closing the switch would create a short circuit (!).

Button

```
const int buttonPin = 2;
const int ledPin =  13;

int buttonState = 0;

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
}

void loop() {
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed.
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  } else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

ANALOG SENSORS: Potentiometer

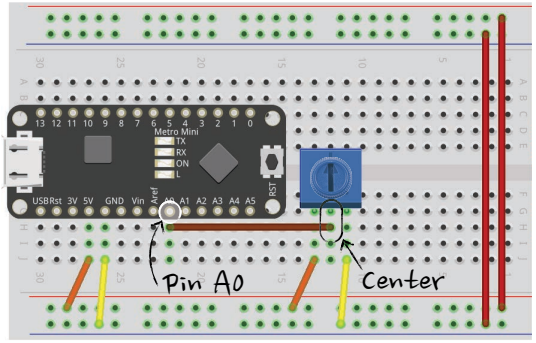
Buttons have only 2 states: open and closed. A potentiometer provides a continuous range of values.

1 Add a Potentiometer

A potentiometer (or "pot") is really just a variable resistor. It includes 3 pins.

Find the potentiometer in your kit and rewire your board as shown below.

Using a long jumper wire, connect the potentiometer's center pin to the Arduino's analog input pin labeled A0. Connect 1 outer pin to the power rail, and the other to the ground rail.

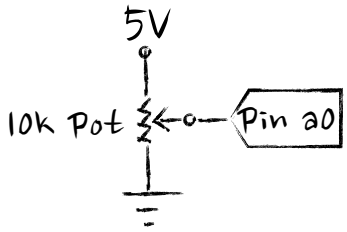


2 Flash "AnalogInput"

Open the AnalogInput sketch, look over how it works for a moment, and then upload it.

File > Examples > 03.Analog > AnalogInput

At first, the onboard LED should flash. As you rotate the potentiometer's cap, the LED should flash at different rates.



For this potentiometer, the resistance between the 2 outer pins is a constant 10K Ohms.

However, the resistance between either of the 2 outer pins and the center pin varies as you turn the cap.

For more about what input the Arduino can sense (only voltage), and how a potentiometer (which varies resistance) works with it, read the page on Voltage Dividers at the end of Part I.

Analog Input

```
int sensorPin = A0;    // select the input pin for the
potentiometer
int ledPin = 13;      // select the pin for the LED
int sensorValue = 0;  // variable to store the value coming from
the sensor

void setup() {
  // declare the ledPin as an OUTPUT:
  pinMode(ledPin, OUTPUT);
}

void loop() {
  // read the value from the sensor:
  sensorValue = analogRead(sensorPin);
  // turn the ledPin on
  digitalWrite(ledPin, HIGH);
  // stop the program for <sensorValue> milliseconds:
  delay(sensorValue);
  // turn the ledPin off:
  digitalWrite(ledPin, LOW);
  // stop the program for for <sensorValue> milliseconds:
  delay(sensorValue);
}
```

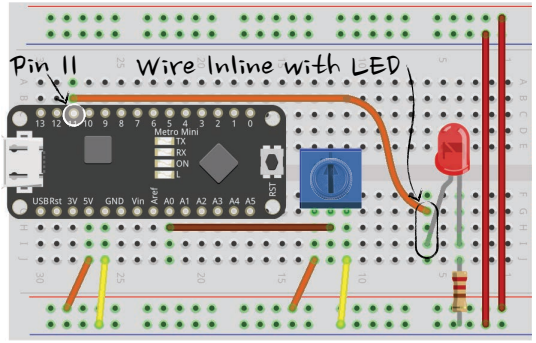
ANALOG SENSORS: Potentiometer & LED

Now that you can read a signal from a potentiometer, lets use it to control an external LED.

1 Add an LED

Without removing the potentiometer circuit, connect a long jumper wire from an external LED to Arduino pin 11, as shown below.

Notice that we're just combining the circuits from a) Fading the LED and b) Analog Sensors: Potentiometer.

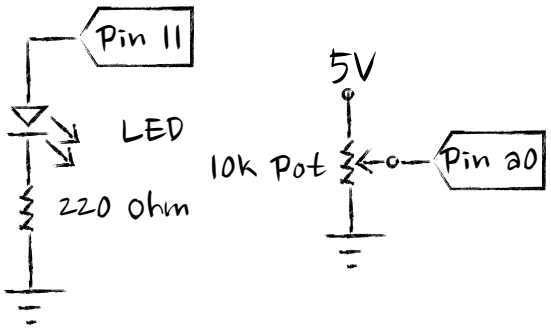


2 Update "AnalogInput"

Update the AnalogInput sketch to flash the external LED that you just wired to pin 11, rather than the onboard LED, which is connected to pin 13, then upload the modified sketch.

If you need a hand, here's the line that you should edit.

```
int ledPin = 13; // select the pin
```



Analog sensors are among the most useful in electronics. Other common types include photocells, proximity sensors, force-sensing resistors (FSRs), accelerometers, gryoscopes and ultra-sonic rangefinders.

Analog Input (ledPin 13 → 11)

```
int sensorPin = A0;    // select the input pin for the
potentiometer
int ledPin = 11;      // select the pin for the LED
int sensorValue = 0;  // variable to store the value coming from
the sensor

void setup() {
  // declare the ledPin as an OUTPUT:
  pinMode(ledPin, OUTPUT);
}

void loop() {
  // read the value from the sensor:
  sensorValue = analogRead(sensorPin);
  // turn the ledPin on
  digitalWrite(ledPin, HIGH);
  // stop the program for <sensorValue> milliseconds:
  delay(sensorValue);
  // turn the ledPin off:
  digitalWrite(ledPin, LOW);
  // stop the program for for <sensorValue> milliseconds:
  delay(sensorValue);
}
```

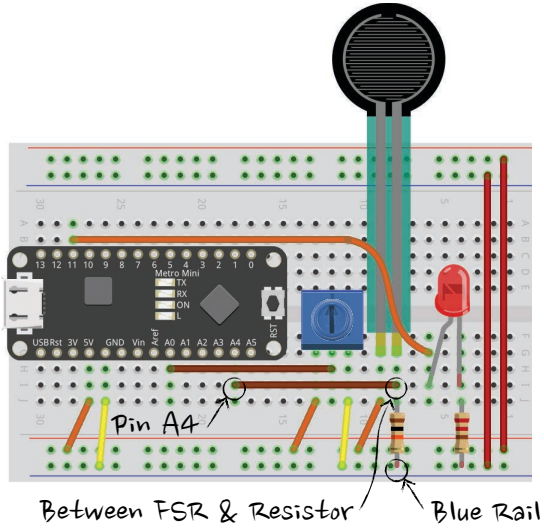
ANALOG SENSORS: FSR & Photocell

Let's explore 2 other analog sensors, both of which are used on real-world robots.

1 Try an FSR

Now is a good time to try the force-sensing resistor (FSR). It looks like this.

Leave the potentiometer and LED in place, and add a 10K Ohm resistor and jumper wires as below. 10K Ohm color bands are: **brown**, **black**, **orange** and (farther away) **gold**.



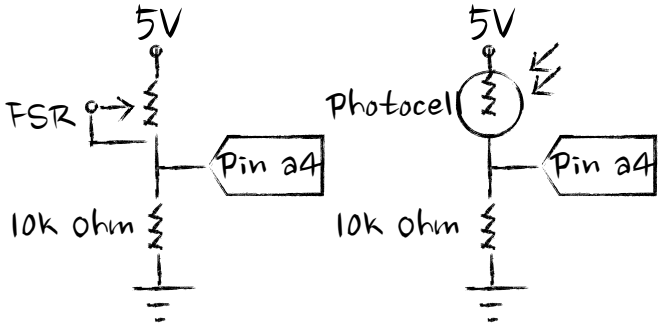
2 Update "AnalogInput"

Update the AnalogInput input sketch to read the value from the FSR, which you just connected to pin A4.

If you still need a hand, edit this line.

```
int sensorPin = A0; // select the pin
```

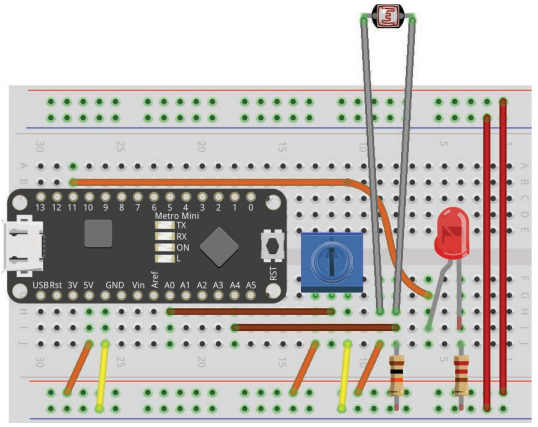
Like a potentiometer, the FSR and photo cell change resistance, just when the pressure or ambient light changes.



3 Try a Photocell

Find the photocell (also called a photo resistor) in your kit. It looks like this.

Just replace the FSR with the photocell. You can continue to use the same 10K Ohm resistor and jumper wires.



Analog Input (sensorPin A0 → A4)

```
int sensorPin = A4;    // select the input pin for the
potentiometer
int ledPin = 11;       // select the pin for the LED
int sensorValue = 0;  // variable to store the value coming from
the sensor

void setup() {
  // declare the ledPin as an OUTPUT:
  pinMode(ledPin, OUTPUT);
}

void loop() {
  // read the value from the sensor:
  sensorValue = analogRead(sensorPin);
  // turn the ledPin on
  digitalWrite(ledPin, HIGH);
  // stop the program for <sensorValue> milliseconds:
  delay(sensorValue);
  // turn the ledPin off:
  digitalWrite(ledPin, LOW);
  // stop the program for for <sensorValue> milliseconds:
  delay(sensorValue);
}
```

FSR, photocell 모두 같은 코드로 수행

VOLTAGE DIVIDERS

One of the most fundamental and useful circuits in electronics!

Why You Need Them

The Arduino's pins can only input or output voltages.

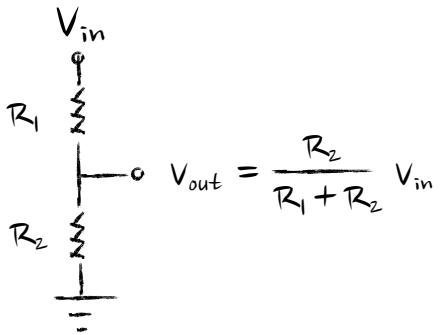
So how did we use sensors, such as a potentiometer, FSR and photocell, which only vary resistance?

Because it's easy to transform varying resistance into varying voltage, using a voltage divider circuit. Here's how...

How Do They Work?

A basic voltage divider has 2 resistors, connected in series, between power and ground, as shown below.

One of these (which is your sensor) has varying resistance, while the other has a fixed resistance.

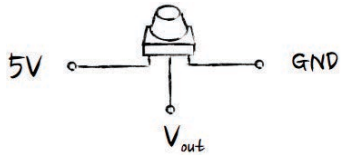


At the point between the 2 resistors (V_{out} on the left), you can measure how much the voltage dropped as it passed through the first resistor (R_1).

The voltage at V_{out} changes as the ratio between the 2 resistances changes (by the equation shown at left).

You can then connect V_{out} to one of your Arduino's analog input pins. On the Metro Mini, these are pins A0–A5.

A potentiometer includes **both** R_1 and R_2, so the connection will look like this.



MAKE THIS! ELECTRONICS PROTOTYPING using ARDUINO

PART II – PAPER ROBOT

1. Servos
2. Robot Face
3. Batteries
4. Arms & Body

SERVOS 1: Making Things Move

When you're ready to move on from flashing lights to moving things.

1 What's a Servo?

A servo is a DC motor, geartrain, potentiometer and feedback circuit, all in a single housing.

By sending a PWM signal from your Arduino to the servo, you're telling it what angular position you'd like it go to.

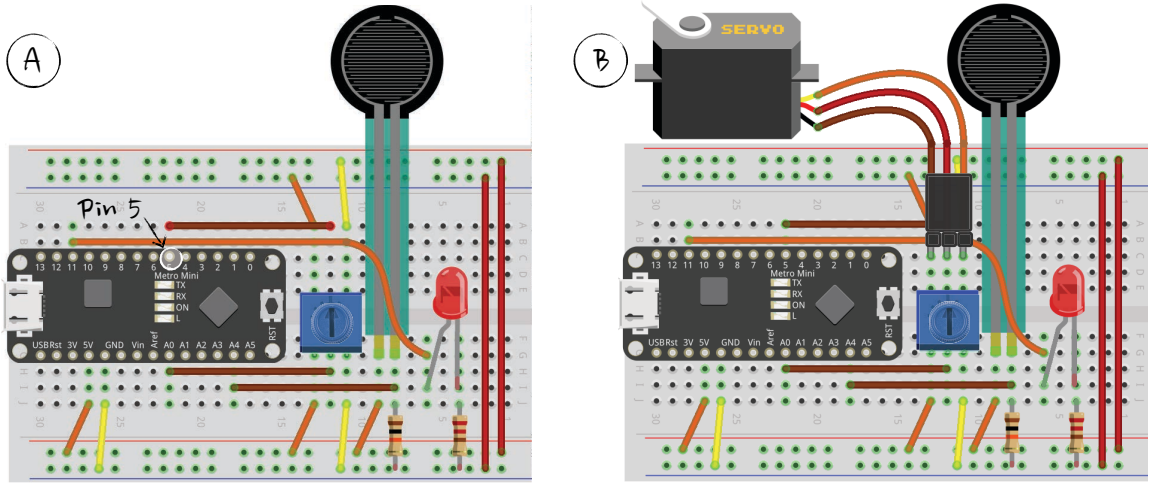
The potentiometer tells the feedback circuit the servo's current position, and the circuit drives the motor to match the desired position.

2 Add a Servo

A: Connect 3 jumper wires as shown on the left, with the long middle wire to pin 5: this wire carries the signal.

B: Connect the servo to your bread board, by snapping off a 3-pin segment from a right-angle breakaway header.

Socket the header as shown below. If it's turned 90 degrees, the wires will short!



Brown: (-) ground
Red: (+)
Orange: signal

3 Flash "Sweep"

Now open the Sweep sketch.

File > Examples > Servo > Sweep

Edit the following line to attach the servo to pin 5, instead of pin 9.

```
myservo.attach(9); // attach the servo
```

When you upload the sketch, your servo should be sweeping back and forth, by about 180 degrees.

Change some parameters in the sketch to make the servo sweep slower, or over a smaller angle.

For example, editing the following line will alter the sweep speed.

```
delay(15); // waits 15ms
```

Sweep (attach 9 → 5)

```
#include <Servo.h>

Servo myservo;  // create servo object to control a servo
int pos = 0;    // variable to store the servo position

void setup() {
  myservo.attach(5);  // attaches the servo on pin 9 to the servo object
}

void loop() {
  for (pos = 0; pos <= 180; pos += 1) { // goes from 0 degrees to 180
degrees
    // in steps of 1 degree
    myservo.write(pos);              // tell servo to go to position in
variable 'pos'
    delay(15);                       // waits 15ms for the servo to reach
the position
  }
  for (pos = 180; pos >= 0; pos -= 1) { // goes from 180 degrees to 0
degrees
    myservo.write(pos);              // tell servo to go to position in
variable 'pos'
    delay(15);                       // waits 15ms for the servo to reach
the position
  }
}
```

SERVOS 2: Controlling Movement

We really want to control servos using sensors. Let's start with a potentiometer.

1 Update "Knob"

Open the Knob sketch.

File > Examples > Servo > Knob

We'll be changing 2 lines here, because we used different pins for our circuit.

A. Edit the following line to set the `potPin` variable to A0, rather than 0.

```
int potpin = 0; // analog pin
```

B. Edit the following line to attach the servo to pin 5, rather than pin 9.

```
myservo.attach(9); // attach the servo
```

2 Flash "Knob"

Now upload and run the Knob sketch, then turn your potentiometer's cap. If all goes well, the servo should turn in sync with the potentiometer's cap.

You can turn the servo the opposite way by updating the following line.

```
val = map(val, 0, 1023, 0, 180);
```

This line maps possible analog input values (which range from 0–1,023) to possible servo angles (which range from 0–180). Try the following instead.

```
val = map(val, 0, 1023, 179, 0);
```

Why aren't we using the default pins in the example sketches?

Arduino's current servo library disables PWM on pins 9 and 10. However, these are the pins that the examples use.

As a result, we've changed our circuit to use other pins instead.

There are 2 common types of servo: standard and continuous. A continuous servo doesn't go to a desired position. Instead, the signal tells it what speed and direction to (continually) turn.

Knob (potPin 0 → A0, attach 9 → 5)

```
#include <Servo.h>

Servo myservo;  // create servo object to control a servo

int potpin = A0;  // analog pin used to connect the potentiometer
int val;    // variable to read the value from the analog pin

void setup() {
  myservo.attach(5);  // attaches the servo on pin 9 to the servo object
}

void loop() {
  val = analogRead(potpin);            // reads the value of the
potentiometer (value between 0 and 1023)
  val = map(val, 0, 1023, 0, 180);     // scale it to use it with the
servo (value between 0 and 180)
  myservo.write(val);                  // sets the servo position
according to the scaled value
  delay(15);                           // waits for the servo to get there
}
```

THE ROBOT'S FACE 1: Using LEDs as Eyes

We recommend using 2 LEDs for the robot's eyes, and the potentiometer for its nose.

1 Add a 2nd LED

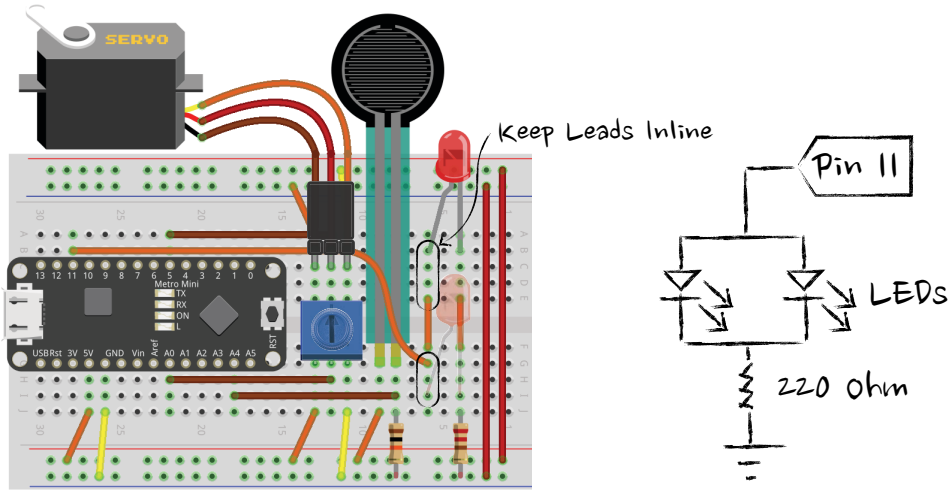
Let's add a second LED in parallel with the first.

First, add 2 short jumper wires to cross the separator down the center of the breadboard, as shown below.

Second, orient the second LED's leads the exact same way as the first LED's, and socket it into the same rows of the breadboard (just on the other side).

Since you've connected 2 rows across the separator, the LEDs will receive the same signal, and both should light up.

With the breadboard set at the bottom of your box, you should now be able to position the LEDs to poke out the top of the box, as the robot's eyes.



LEDs in Series & Parallel

LEDs can be connected in series or in parallel. Wired in series, the supply voltage gets divided among the LEDs, so each LED will appear dimmer.

Wired in parallel, each LED draws the full supply voltage, so they appear the same brightness (but the power supply will drain faster as a result).

THE ROBOT'S FACE 2: Moving Sensors Off-Board

Sensors are often more useful located away from the main circuit board, such as a robot's face or arm.

1 Breadboard Layout

As you build projects with many lights, sensors and actuators, planning the layout of your breadboard beforehand can be a great help.

But you may eventually need to move LEDs, potentiometers, FSRs and photocells off the breadboard, so you can attach them to your project wherever they're needed.

2 Long Jumper Wires

The shorter, right-angle jumper wires have a core of solid wire. They hold their set shape well, but aren't flexible.

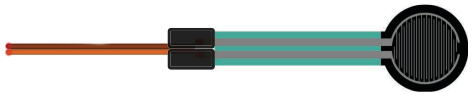
The longer jumper wires have a core of flexible, stranded wire, making them great for connecting to components off the breadboard.

Find the male-female jumper wires in your kit. They look like this.

3 Move the Sensors

Split off a 2-wire pair and a 3-wire triple. Try to keep each of them joined over its length, to keep things tidy.

Remove the FSR from your bread board. Attach one end to the 2-wire pair, and socket the other end back into the breadboard (in the same place as before). Do the same for the potentiometer and the 3 wires.



You may want to use some Blu-Tack adhesive, or electrical tape, to hold the sensors in place against the wires.

POWER: USB & Batteries

The electronics are almost done. Let's untether the electronics from your laptop.

About the USB Cable

Up to this point, your Arduino has been connected to your laptop through a USB cable.

The cable does 2 things: 1) it allows the laptop and Arduino to communicate, and 2) it powers the Arduino.

Many do-it-yourself (DIY) projects are meant to run standalone, without being connected to a laptop. For that, we use batteries.

Arduino sketches reside in flash memory, so once a sketch is uploaded, it can run without communicating with your laptop..

1 Add a Battery

We've had many students injure their Arduino when connecting a battery (!). Please follow these 3 steps carefully.

A: Find the 9V battery in your kit and snap on the cap with red & black leads.

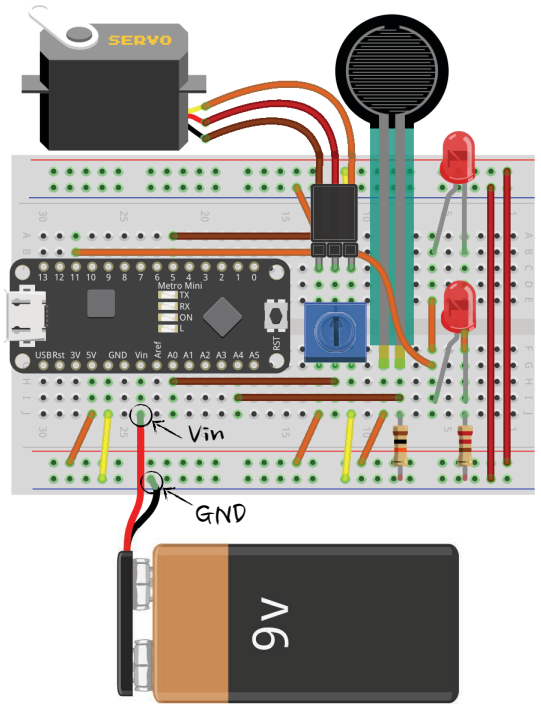
B: Connect the black (ground) lead to either the ground rail or one of the 2 GND pins on the Arduino.

C: Connect the red (power) lead to the Vin (voltage in) pin. Only this pin has a regulator, which safely translates 9V down to 5V.

Your project should start running again, with no tether to the laptop.

Final Breadboard Layout

Here's the final layout of your robot's electronics.



THE ROBOT'S BODY

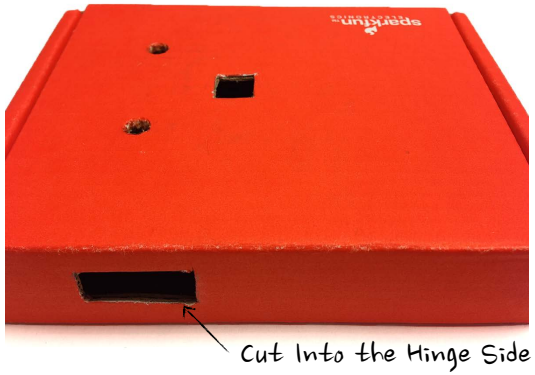
It will house the breadboard circuit, off-board components and battery.

1 Place Openings

Trace out the shape of the 2 LEDs, potentiometer and servo on the box's top and side, and cut them out with a utility knife.

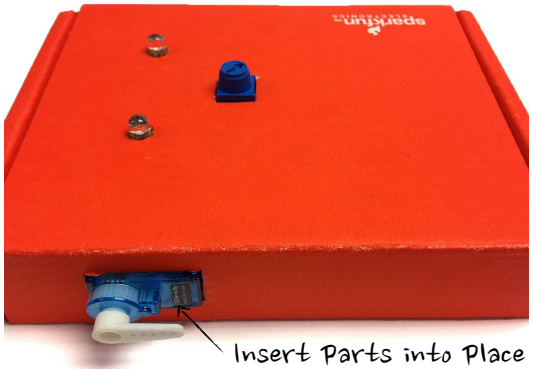
2 Cut Openings

Cut the servo into the hinge side of the box! Otherwise you have to cut through 2 layers of cardboard (including the flap) instead of 1.



3 Attach Parts

Position and tape or glue components in place. Attach them from inside of the box to keep the outside clean.



THE ROBOT'S ARMS

Both arms work together: one supports the FSR, the other attaches to the servo.

1 Cut Out 2 Arms

The robot's arms can be cut from construction paper, cardboard or foam sheets. Here's a sample pattern you might follow.

To attach the cutout arm to the servo, snap a white plastic arm (from the clear plastic bag) onto the servo's hub, then tape or glue the cutout arm to that.

2a Make an Arm Move

There are 2 approaches to making an arm move. The quickest approach is to reuse the Sweep or Knob sketches from the start of this section.

Keep in mind that Sweep constantly moves the arm (unless you edit it), and Knob requires the potentiometer.

The fancier approach is to interactively shake hands, described next...

2b Shake Hands

If you want the robot to shake hands, follow these 3 easy steps.

A: Bring the FSR to the outside of the box, either through the top of the lid, or an opening you cut along one side.

B: Affix (tape or glue) the FSR to the arm that **is not** attached to the servo, positioning the round pad at the hand.

You can affix the FSR to the arm that **is** attached to the servo, but wiring can be a challenge, since the arm moves.

C: Write a program that controls the robot utilizing what we have learned today and upload it!

Debugging Tip

```
void setup() {
  Serial.begin(9600); // open the serial p ort at 9600 bps
}

void loop() {
  Serial.print("print without a newline");
  Serial.println("print with a newline");
}
```