INTERNET OF THINGS CLASS

6 Lessons Intermediate Level

This intermediate level Arduino class guides you through creating your first internet-connected electronics projects using a wifi breakout board. Learn the elaborate workflow of hardware and software that makes smart objects succeed through basic step-by-step examples of the most common things you want to build: a sensor that triggers an email (or tweet, etc.), a circuit that displays info fetched from online, and how to combine sample code to build your own project ideas.

Whether you're a software engineer just dipping a toe into hardware, or a novice who just finished the introductory Arduino Class (https://www.instructables.com/class/Arduino-Class/), this class will give you the skills to realize the IoT projects of your dreams. You'll explore cloud services to quickly and easily link your DIY circuits up with other IoT devices, social media sites, and more.

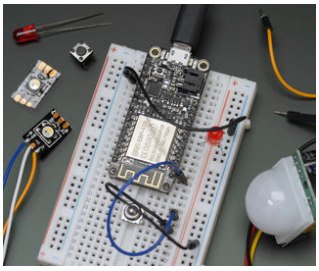
Enter an Instructables contest!

If you've used the knowledge from this class to create an awesome project, write an instructable about it and try entering it in one of our contests (https://www.instructables.com/contest/)!

Class Author:
bekathwia (/member/bekathwia/)

Becky Stern (https://www.instructables.com/member/bekathwia/) has authored hundreds of tutorials in everything from wearable electronics to knitting. Before joining Instructables as a content creator, Becky worked as a senior video producer for MAKE Magazine and then as the director of wearable electronics at Adafruit Industries. She lives in New York City and enjoys riding her motorcycle, making YouTube videos (https://www.youtube.com/c/beckystern), and collecting new hobbies. Her work has been featured by VICE, the BBC, The Late Show with Stephen Colbert, Engadget, CNN, Business Insider, Forbes, and Science Friday.

Lessons

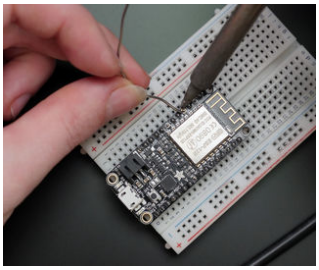


Lesson 1: Gather Your (Internet Of) Things

Find out what you'll learn, what you'll need to get started (with links to all supplies and tools used), and get a little more context on this class' approach.

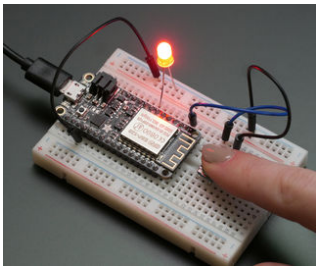
Lesson 2: Software Setup

You'll need to install some software to get working with the ESP8266, even if you've used Arduino before. These one-time steps will set you up for success in all your future lessons and projects!



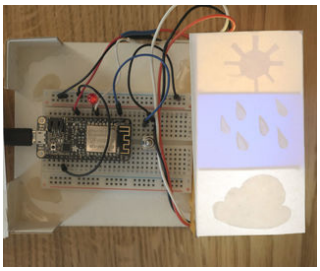
Lesson 3: Hardware Setup

Learn about your board assembly options, features, and limitations. Put together a basic solderless breadboard circuit using a pushbutton and an LED.



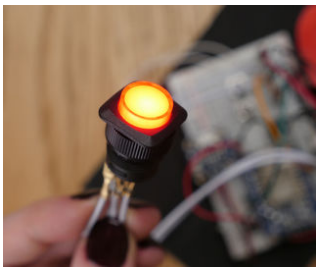
Lesson 4: Circuit Triggers Internet Action

Whip up a simple prototype that sends an email when you press a pushbutton! This lesson will walk you through the setup of your first Adafruit IO feed and IFTTT applet.



Lesson 5: Circuit Displays Internet Data

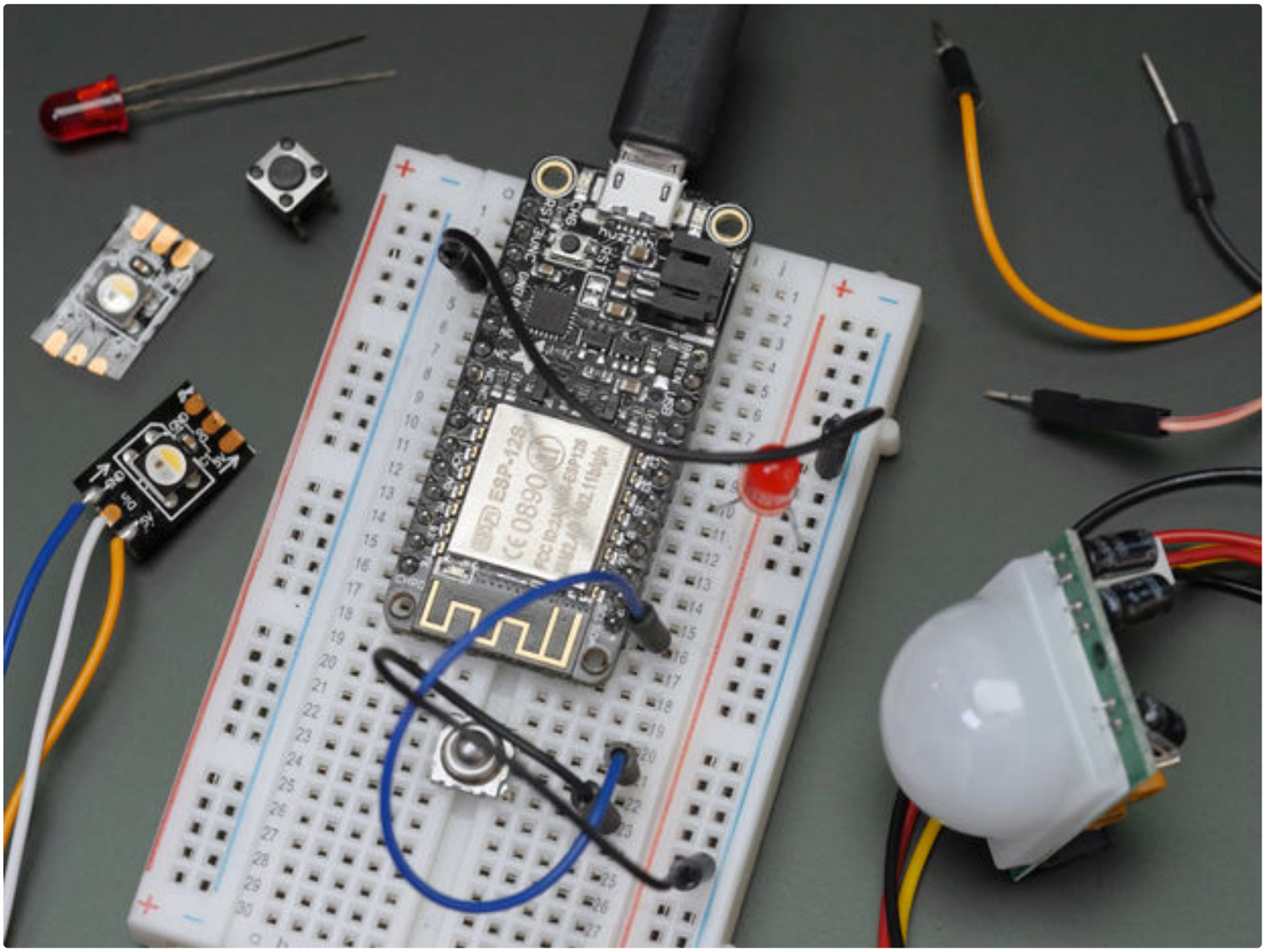
Level up to create colorful representations of weather conditions by collecting forecast data from IFTTT through Adafruit IO and displaying it with NeoPixel LEDs.



Lesson 6: Combining Inputs and Outputs

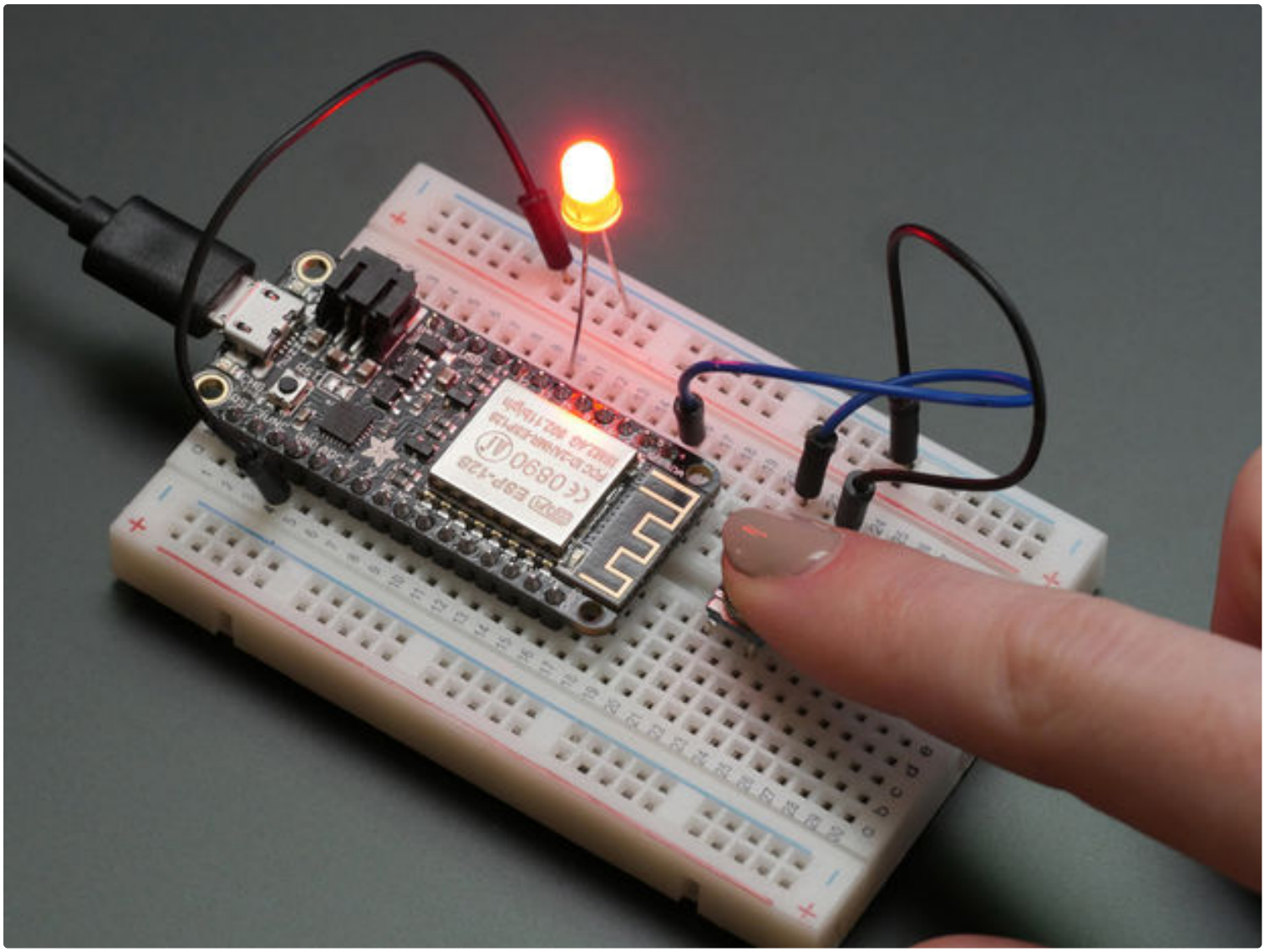
Learn to expand on the basic examples from this class to create your own projects, and see how two devices can interact with one another using the same feed and Arduino code, plus find links to more projects and learning resources.

LESSON 1: GATHER YOUR (INTERNET OF) THINGS



Welcome to class! Let's get your electronics projects talking online. This class will help you dip a toe in the IoT ocean by building and programming your own simple wireless device, even if you have no prior experience connecting your projects to the cloud. The class is built around free software tools and beginner-friendly hardware.

We'll be programming an ESP8266 wifi board using the Arduino software and programming language. If you're new to programming or microcontrollers, I highly recommend first completing my introductory Arduino Class (https://www.instructables.com/class/Arduino-Class/)— it will get you up to speed on the coding and wiring basics before adding the complexity of getting your circuit online. You may also be interested in these other related classes written by my smart and talented colleagues: Electronics (https://www.instructables.com/class/Electronics-Class/), Raspberry Pi (https://www.instructables.com/class/Raspberry-Pi-Class/), Wearable Electronics (https://www.instructables.com/class/Wearable-Electronics-Class/), Robots (https://www.instructables.com/class/Robots-Class/), and LEDs & Lighting (https://www.instructables.com/class/LEDs-and-Lighting-Class/).



What You'll Learn

The topic of connected devices is huge. There are undeniably huge sociological implications (https://motherboard.vice.com/en_us/article/internet-of-things-teddy-bear-leaked-2-million-parent-and-kids-message-recordings) of the proliferation of ever-present, ever-chatting circuits into the crevices of our existence. This class, however, tries to reign in any budding existential crises by focusing on the fun and easy process of creating your own wifi circuit that does your bidding (for now, anyway). We'll walk through a few "recipes" that can serve as the foundation for many common types of projects you may have in mind.

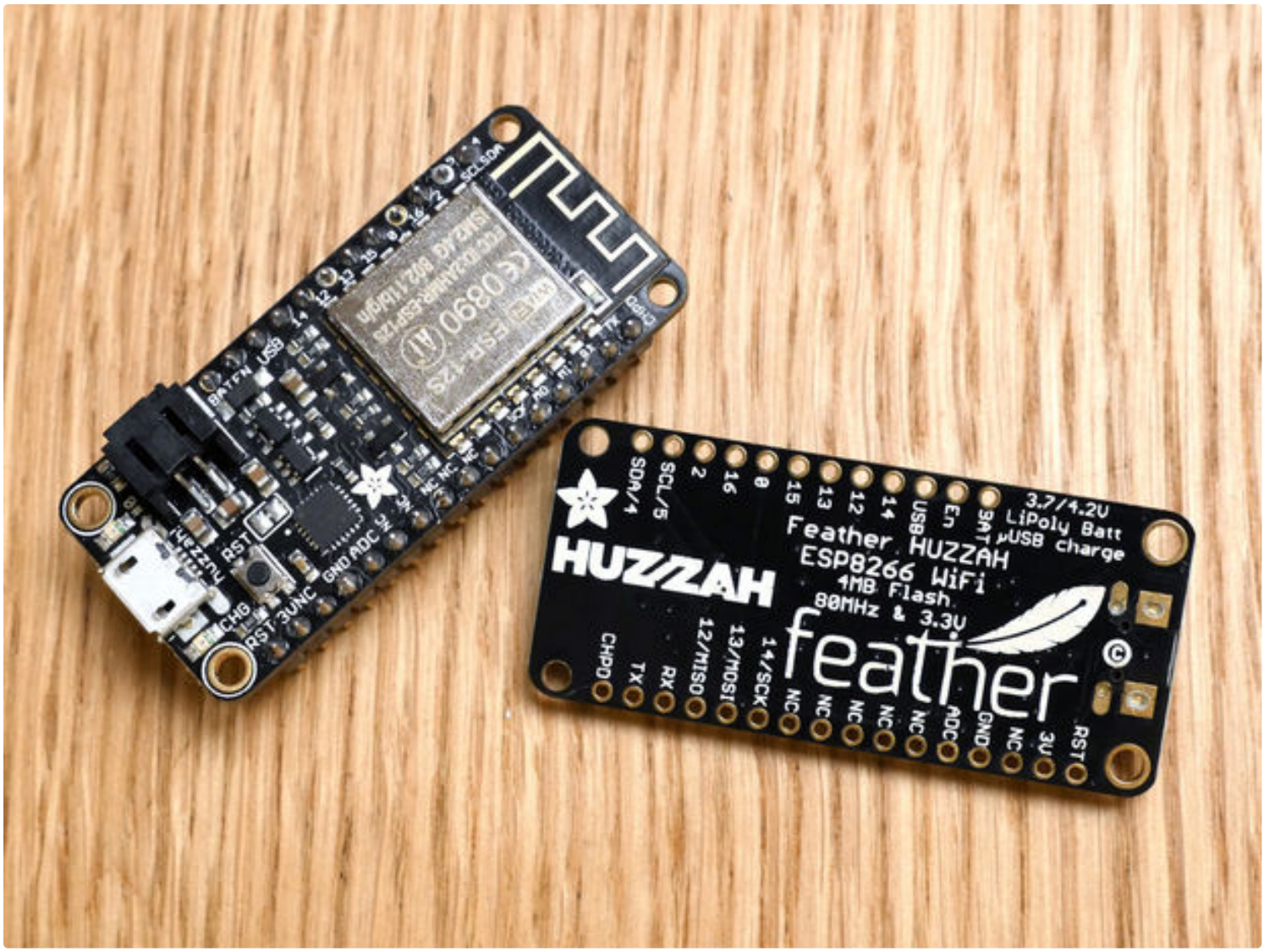
Software and Hardware Setup — There are a few things to download and configure to get your computer talking with the ESP8266 board, and a few tests to perform to be sure you've got a clear connection to the web. These necessary housekeeping lessons set you up for success and provide valuable reference/troubleshooting information.

Circuit Triggers Internet Action — Want to be notified when a door is opened, your water heater leaks, or there's movement at the bird feeder? This lesson shows you how to use a switch or sensor to trigger an online notification of your choice. Get acquainted with tools for building online projects like cloud data service Adafruit IO (https://io.adafruit.com/) and the API gateway site If This Then That (IFTTT) (https://ifttt.com/).

Circuit Displays Internet Data — Do you want to build a realtime weather monitor, YouTube subscriber counter, or other data-driven project? In this lesson, you'll build a circuit and program that listens to an online feed of data and then gives you some real world feedback by lighting up some LEDs accordingly.

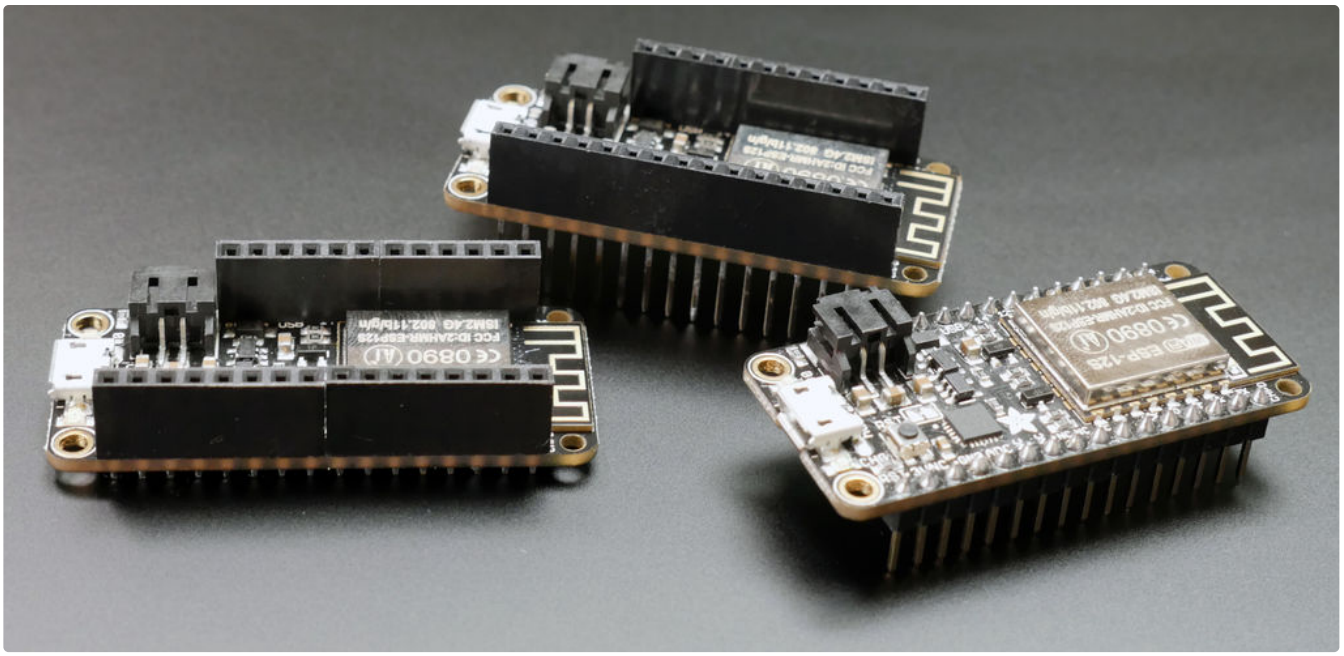
Combining Inputs and Outputs — Your final challenge will be to combine the coding skills you picked up from the previous two! By building an interactive device that both listens and speaks to the internet, you'll establish the foundation needed for more complex projects like telepresence valentines, automatic gardening systems, and much more.

Follow along to pick up the skills needed to create basic IoT devices on your home wireless network. At the bottom of each lesson there is a Q&A; section; I'll be checking in frequently to answer any questions you may have as you go. After you've built the basic examples in this class, you'll understand the terminology and design approach needed to research and incorporate new components into your projects. This class gives you the keys to unlock the web's vast resources for building internet-connected DIY electronics projects. The only limits to what you can create are your imagination and follow-through. Including Skynet (https://en.wikipedia.org/wiki/Skynet_(Terminator)).



Class Tools and Materials List

In this class, we'll be using an Arduino-compatible wifi board called the Feather Huzzah ESP8266, made by Adafruit. It's easy to program over USB without any additional hardware, you can get it pre-assembled so no soldering is required (at least not right away), and its compact size and onboard battery charger make it perfect for building the brains of any smart object. It also works with a wide variety of plug-and-play accessories, called Feather Wings, that make it easy to add on features with minimal circuit troubleshooting.

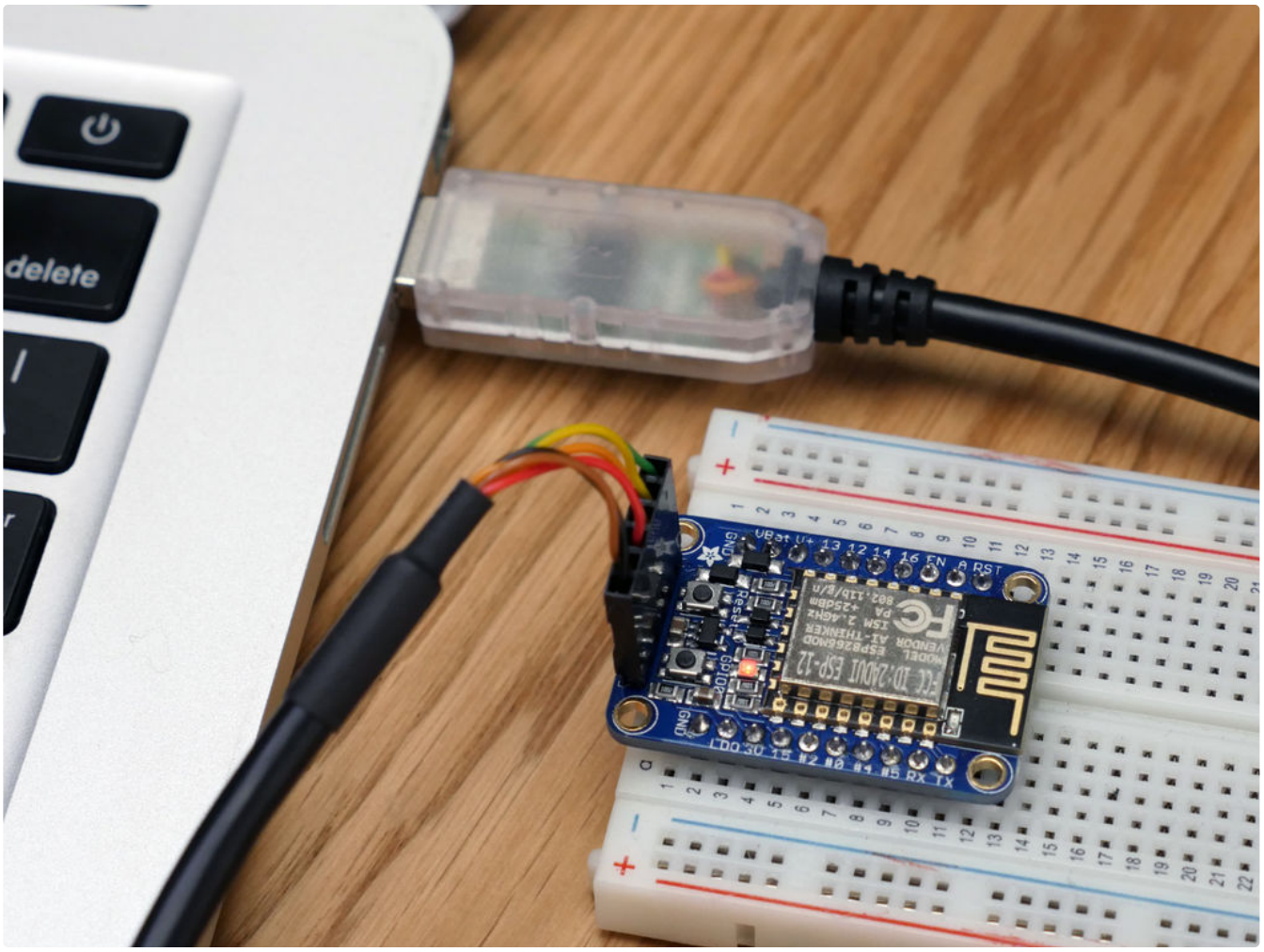


The lessons here will build on the programming and electronics prototyping skills first introduced in our beginner Arduino Class (https://www.instructables.com/class/Arduino-Class/), so please review the lessons there and pick up any skills that are unfamiliar, even if you've used Arduino before. Most of the supplies for this class overlap, so if you've already completed the Arduino Class, you only need to pick up the following additional parts:

- Adafruit Feather Huzzah ESP8266 (https://learn.adafruit.com/adafruit-feather-huzzah-esp8266) board in the configuration of your choice:
    - assembled with stacking headers (https://www.adafruit.com/products/3213) (for adding wings)
    - assembled with regular headers (https://www.adafruit.com/products/3046)
    - unassembled for solder-your-own headers (https://www.adafruit.com/products/2821)
- PIR motion sensor (https://www.adafruit.com/products/189)
- USB power supply (https://www.adafruit.com/products/1994) (optional)
- Lipoly battery (https://www.adafruit.com/product/1578) (optional)

from L to R: NodeMCU ESP8266-12E devkit board, Adafruit Feather Huzzah, generic ESP8266 module only, Adafruit Huzzah Breakout

Alternatively, you can substitute the ESP8266 module of your choice to follow along with this class. As my second choice, I recommend the NodeMCU devkit ESP8266-12E (https://www.amazon.com/dp/B010O1G1ES/?tag=instructabl09-20) (files on github (https://github.com/nodemcu/nodemcu-devkit-v1.0)), which is also programmable over USB.



The Sparkfun Thing (https://www.sparkfun.com/products/13231) and Adafruit Huzzah (https://www.adafruit.com/products/2471) (non-Feather variety) are also nice alternatives, though they both require an additional programming cable (https://www.adafruit.com/product/70). I do not recommend any of the tiny module-only boards (https://www.amazon.com/dp/B01EA3UJJ4/?tag=instructabl09-20) for beginners, as they are more difficult to set up (https://learn.adafruit.com/esp8266-temperature-slash-humidity-webserver/code?view=all#wiring) and use than the others (not breadboard friendly, require exactly 3.3v which makes programming tricky, even with a special cable, and many features are not easily accessible to the user). Pin configurations vary between boards, so if you're making a substitution, double check your product documentation when building your circuits with the diagrams in this class.

In addition to an ESP8266 board, here are all the components you'll need (so pick them up if you haven't already completed the Arduino class):

- USB cable (https://www.adafruit.com/products/592) (A to micro B, must be data+power, not power only)
- Solderless breadboard (https://www.adafruit.com/products/64)
- Breadboard prototyping wires (https://www.adafruit.com/products/153)
- Pushbutton (https://www.adafruit.com/products/3347)
- 10K ohm resistor (https://www.adafruit.com/products/2784)
- 2 LEDs with leads (https://www.adafruit.com/products/299) (your choice (https://www.adafruit.com/categories/90) of color and size)
- A handful of RGBW NeoPixels (aka RGBW WS2812b) LEDs, your choice:
    - Strips (https://www.adafruit.com/categories/183?q=rgbw%20led%20strip%20neopixel&): black or

white circuit board color, natural white LEDs, in densities 30/m, 60/m, and 144/m

- Sticks (https://www.adafruit.com/products/3039): natural white, cool white, warm white
- Rings (https://www.adafruit.com/products/3042): sizes 12, 16, 24, and 60 in natural white, cool white, and warm white
- Jewels (https://www.adafruit.com/products/3047): natural white, cool white, warm white

I've put together an Adafruit wishlist (http://www.adafruit.com/wishlists/431526) of all the components needed for this class— just remove anything you already have, then customize your header configuration and pixel type!

To solder your pixels, headers, and build more permanent circuits, review this intro lesson (https://www.instructables.com/lesson/Soldering-1/) and gather up the following tools:

- Soldering iron (https://www.adafruit.com/products/180) and solder (https://www.adafruit.com/product/145)
- Small needlenose pliers (https://www.adafruit.com/product/146)
- Flush diagonal cutters (https://www.adafruit.com/product/152)
- Wire strippers (https://www.adafruit.com/products/147)
- Multimeter (https://www.adafruit.com/products/2034) (optional but very handy)
- Tweezers (https://www.adafruit.com/products/421)
- Third hand tool (https://www.adafruit.com/products/291)
- Desoldering braid (https://www.adafruit.com/products/149) or solder sucker

(https://www.adafruit.com/products/148) (for mistakes)

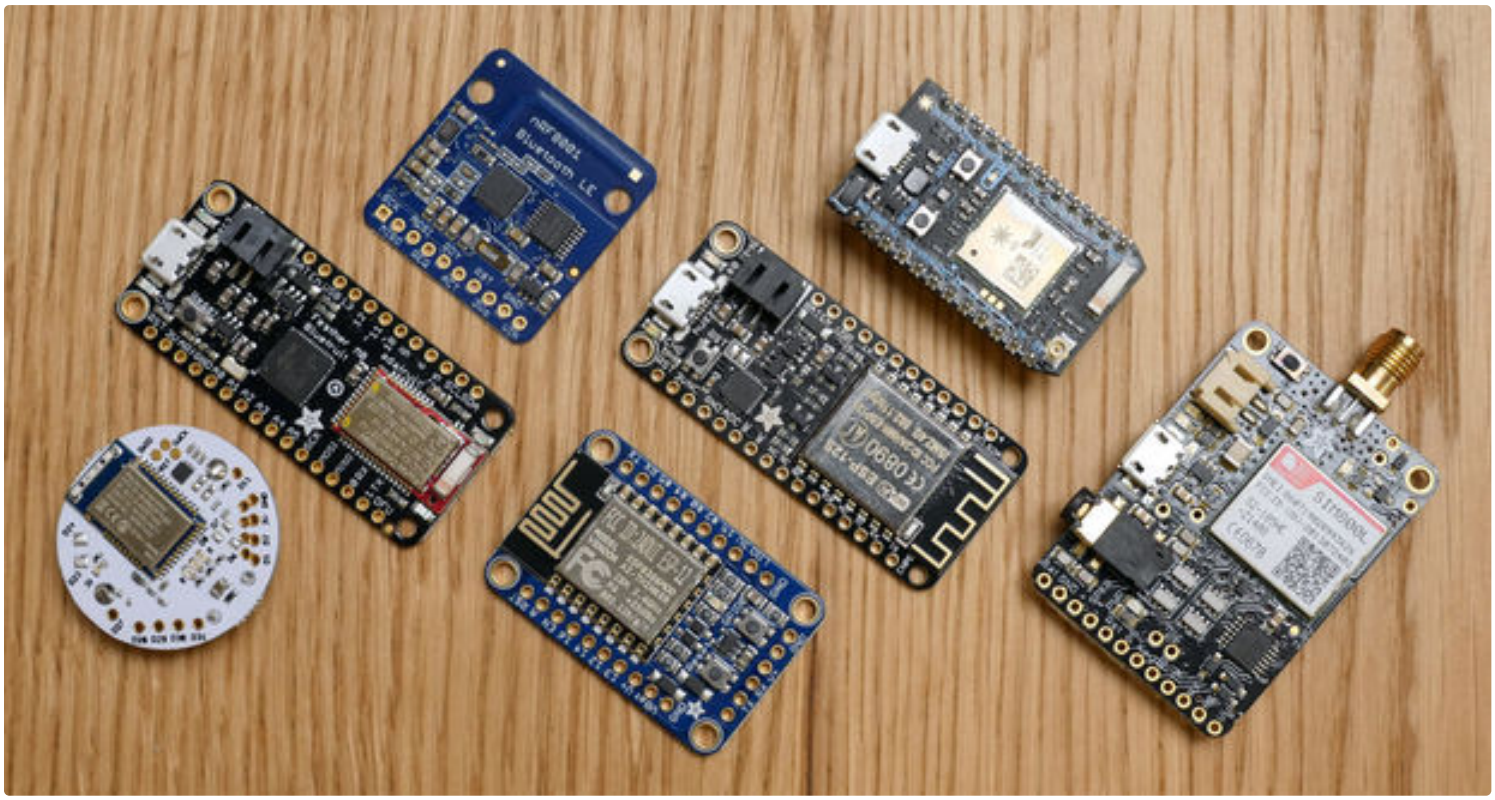
- Tape

To follow along with the lessons, you will need (free) accounts on the following sites:

- Adafruit IO (https://io.adafruit.com/) - cloud data service for storing important info
- IFTTT (https://ifttt.com) (If This Then That) - API gateway for linking in a multitude of online services

You'll also need a computer to run the Arduino software (https://www.arduino.cc/en/Main/Software) (Mac/Windows/Linux, unfortunately the web editor does not support the ESP8266 at this time), and access to an unrestricted wifi network, such as your own home network (with or without a password). School/work networks with web logins (captive portals) and/or firewalls can be problematic for getting your projects online. If this describes your situation, you may consider putting your phone into wifi tethering mode, effectively sharing the 4G data connection to devices connected to it as a wifi hotspot (students of mine have had success with this workaround in the past).

Keep your data safe! Always use strong, unique passwords. Enable two-factor authentication where offered and perform regular security checkups to update passwords and revoke app access to anything you don't use anymore.



About Other Platforms

There are many options available to you, the intrepid IoT DIYer. Let's round out a few of the other common tools and platforms you're likely to see around Ye Olde Internets. *(Clockwise around the Feather Huzzah starting at bottom left: Puck.js, Feather M0 Bluefruit, nRF8001, Particle Photon, Adafruit Fona, Huzzah ESP8266 basic breakout)*

Lua — Most ESP8266 boards you'll find, including the ones recommended in this class, come preprogrammed with NodeMCU's Lua interpreter. Lua (https://www.lua.org/) is a scripting language that is

quite popular with web developers entering the the IoT community because of its simple (and perhaps already-familiar) procedural syntax. While this class takes a different approach that builds on your prior Arduino experience, you may be interested to try Lua before beginning the lessons here. The Feather Huzzah product guide has a succinct tutorial about connecting to the Lua interpreter (https://learn.adafruit.com/adafruit-feather-huzzah-esp8266/using-nodemcu-lua?view=all#using-nodemcu-lua). I do not use Lua frequently, but you deserve to know about all the options available to you! It's never been a better time to learn DIY electronics, no matter what path your learning takes.

MicroPython — Similar to Lua, MicroPython (https://micropython.org/) is another way to write for and upload code to your ESP8266 board (and many other boards, too). Adafruit has some introductory tutorials and example projects (https://learn.adafruit.com/search?q=micropython%20esp8266&view=all) using MicroPython and their own derivative, CircuitPython (https://learn.adafruit.com/search?q=micropython%20esp8266&view=all).

Other popular Wifi platforms — Check out the Particle Photon (https://www.particle.io/products/hardware/photon-wifi-dev-kit), Arduino 101 (https://www.arduino.cc/en/Main/ArduinoBoard101), and SparkFun ESP32 Thing (https://www.sparkfun.com/products/13907)

Arduino-compatible Bluetooth boards —This class will not cover bluetooth applications, but much of what you learn in this class will directly apply to working with Arduino-compatible bluetooth boards like the NRF8001 (https://learn.adafruit.com/getting-started-with-the-nrf8001-bluefruit-le-breakout/) (great for adding onto the Arduino Uno you already have) or one of the bluetooth flavors of Adafruit Feather boards (https://www.adafruit.com/categories/835) (all-in-one boards). I recommend checking out the Adafruit Bluefruit LE Connect app (iOS (https://itunes.apple.com/us/app/adafruit-bluefruit-le-connect/id830125974?mt=8) and Android (https://play.google.com/store/apps/details?id=com.adafruit.bluefruit.le.connect&hl=en)) and associated tutorials (https://learn.adafruit.com/search?q=%22bluefruit%20le%20connect%22&) if you're looking for an Arduino-based bluetooth solution to control a robot or LED wearable project with a phone or tablet. You can also use a mobile device as a bridge between an Arduino-compatible bluetooth device and the outside internet through your device's data connection, whether cellular or wifi.

Raspberry Pi — Why doesn't this class use Raspberry Pi? I knew you were going to ask that! It's mostly because we have a dedicated class about Raspberry Pi (https://www.instructables.com/class/Raspberry-Pi-Class/) already, and it teaches you to create a Tumblr-posting photo booth, which seems pretty IoT to me. Go check it out and let me know how it goes.

The devices we'll build in this class require far less processing power than you'll find in even the simplest

Pi, and also require far fewer accessories to get up and running. I like to think of Raspberry Pi as capable of anything a cell phone can do: play video on a display, connect to the internet, etc. The ESP8266 is more like the circuit running your coffee maker than it is like a cell phone.

And many, many more — We've only just barely outlined a few corners of the vast sea of connected devices. We can learn and build with these platforms because they are open. But most of the embedded devices we encounter every day are closed, meaning you can't access the documentation about how they work. We still don't know too much about the potential flaws or backdoor access holes in many of the devices we're naively welcoming into our homes. There is a saying that goes "security through obscurity (https://en.wikipedia.org/wiki/Security_through_obscurity) is no security at all."

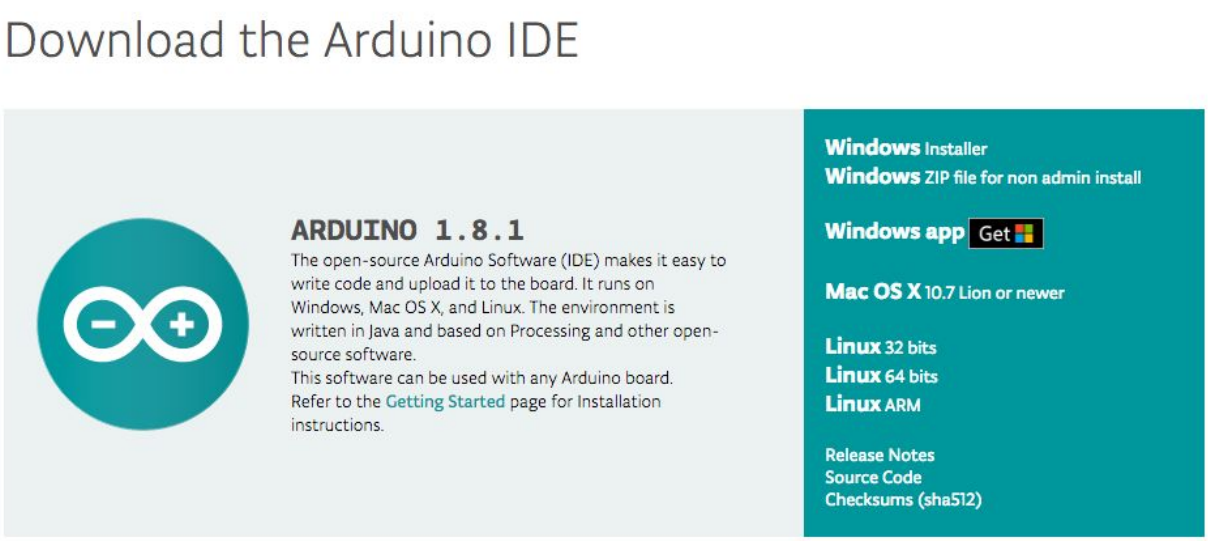
LESSON 2: SOFTWARE SETUP

Hang on to your keyboard and mouse, this part might get bumpy! Follow along closely as we configure and upload your first DIY wifi-connected software. In this lesson, we'll install the ESP8266 boards packages inside your Arduino software, install a device driver for the board's communications chip, and upload an Arduino sketch that connects to your home wireless network. Then you'll be all set to go tinker your heart out. Don't fret if these mandatory setup tasks get you down; the good news is that once you've completed the steps and tests in this lesson, you won't have to do them again (unless you're setting up a new computer).

To get started, you'll need:

- Computer running Arduino software (https://www.arduino.cc/en/Main/Software) (free, available for Mac/Windows/Linux, unfortunately the web editor does not support the ESP8266 at this time)
- ESP8266 board (we're using a Feather Huzzah (https://learn.adafruit.com/adafruit-feather-huzzah-esp8266))
- Programming cable (micro USB (https://www.adafruit.com/products/592) for Feather Huzzah, but some others require an FTDI programming cable (https://www.adafruit.com/product/70))
- Wireless internet connection (unrestricted by web login/captive portal or firewall) with network name and password (if applicable)

Adding ESP8266 Board Support to the Arduino Software



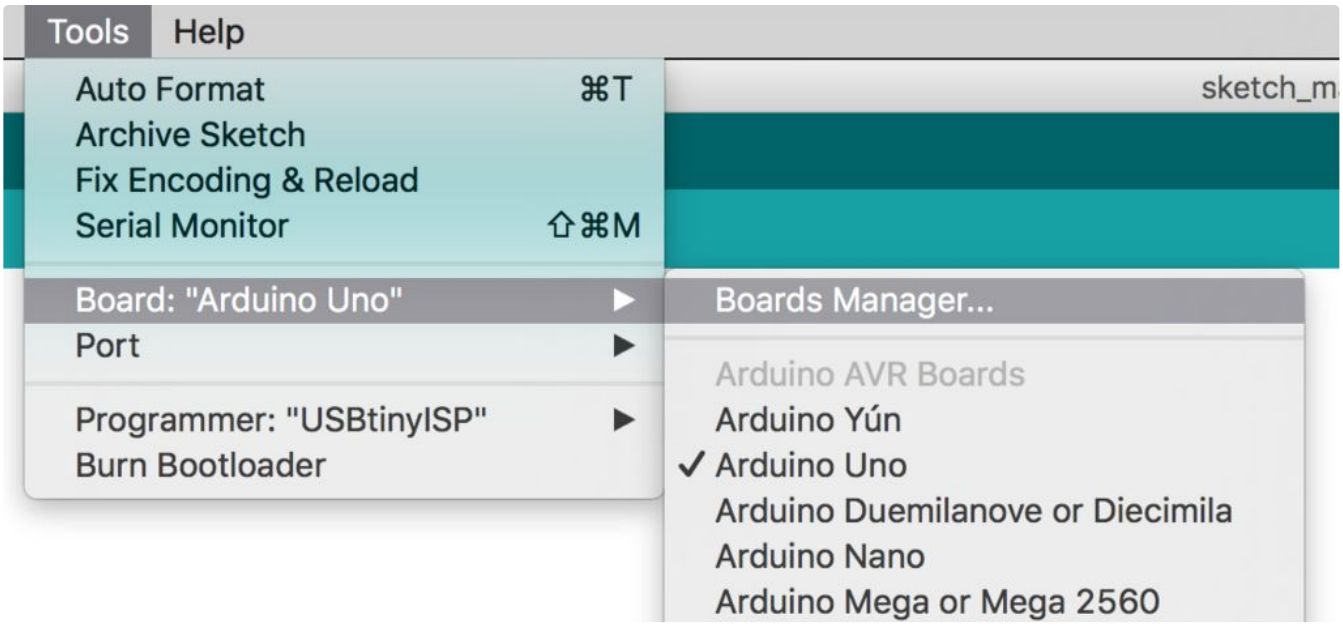
Download and install the latest version of the Arduino software (https://www.arduino.cc/en/Main/Software) on your computer, if you haven't already. Open up the Arduino application and navigate to the menu item Arduino->Preferences.

By default, the Arduino application supports chips used in official Arduino boards, but not the ESP8266. These boards can be programmed "out of the box" because the Arduino application already knows about each one and its properties. One cool thing about Arduino is that you can add support for other boards, and all you have to do is tell Arduino where to discover their properties. The first step of that process is providing a URL to the Additional Boards Manager. In the text box near the bottom of the Preferences window, paste in exactly this text:

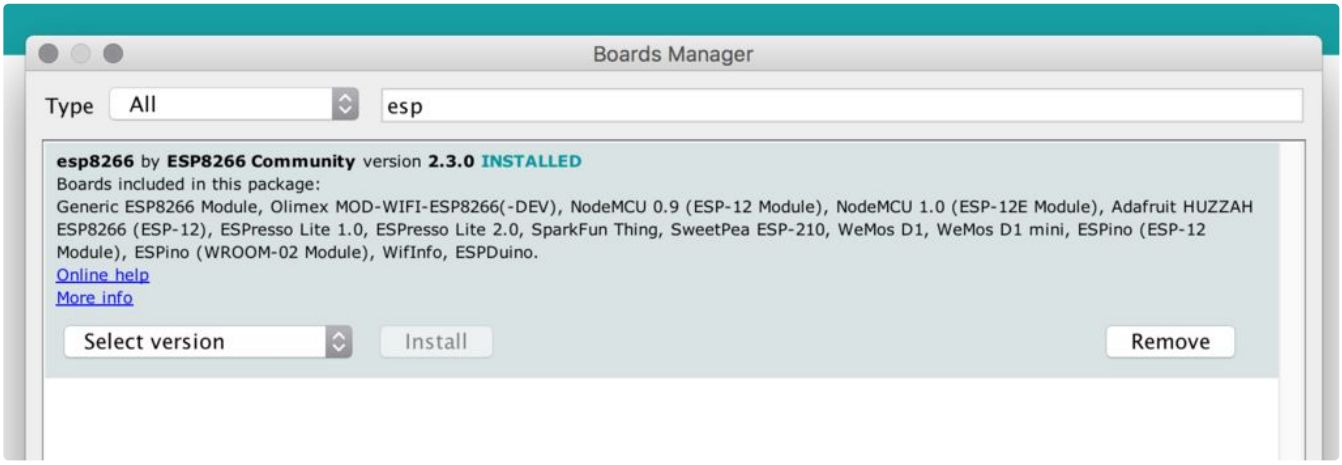
```
http://arduino.esp8266.com/stable/package_esp8266com_index.json
```

If the box was not blank when you opened the Preferences window, you may have some other boards already installed. If that's the case, append the text box's contents with the above URL, using a comma to separate multiple URLs.

Click OK to close the Preferences window. Now our Arduino application knows where to find info about ESP8266 boards in general.



To get specific, navigate to the menu item Tools-> Board:(board name)-> Boards Manager. Allow a moment for the boards manager to download its index, then start typing "ESP8266" into the search bar.



When you see "esp8266 by ESP8266 Community," you can stop typing in the search and click "Install" to get the latest boards package installed inside your Arduino application.

The Feather Huzzah has a handy USB communications chip on board, but it requires a free driver to be installed to function properly. Without it, your board would not show up in the list of available serial devices. Head to the SiLabs page (https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers) and download/install the driver that matches your operating system (Mac/Windows/Linux available).

Got all that? Great, let's test it out in the next step!

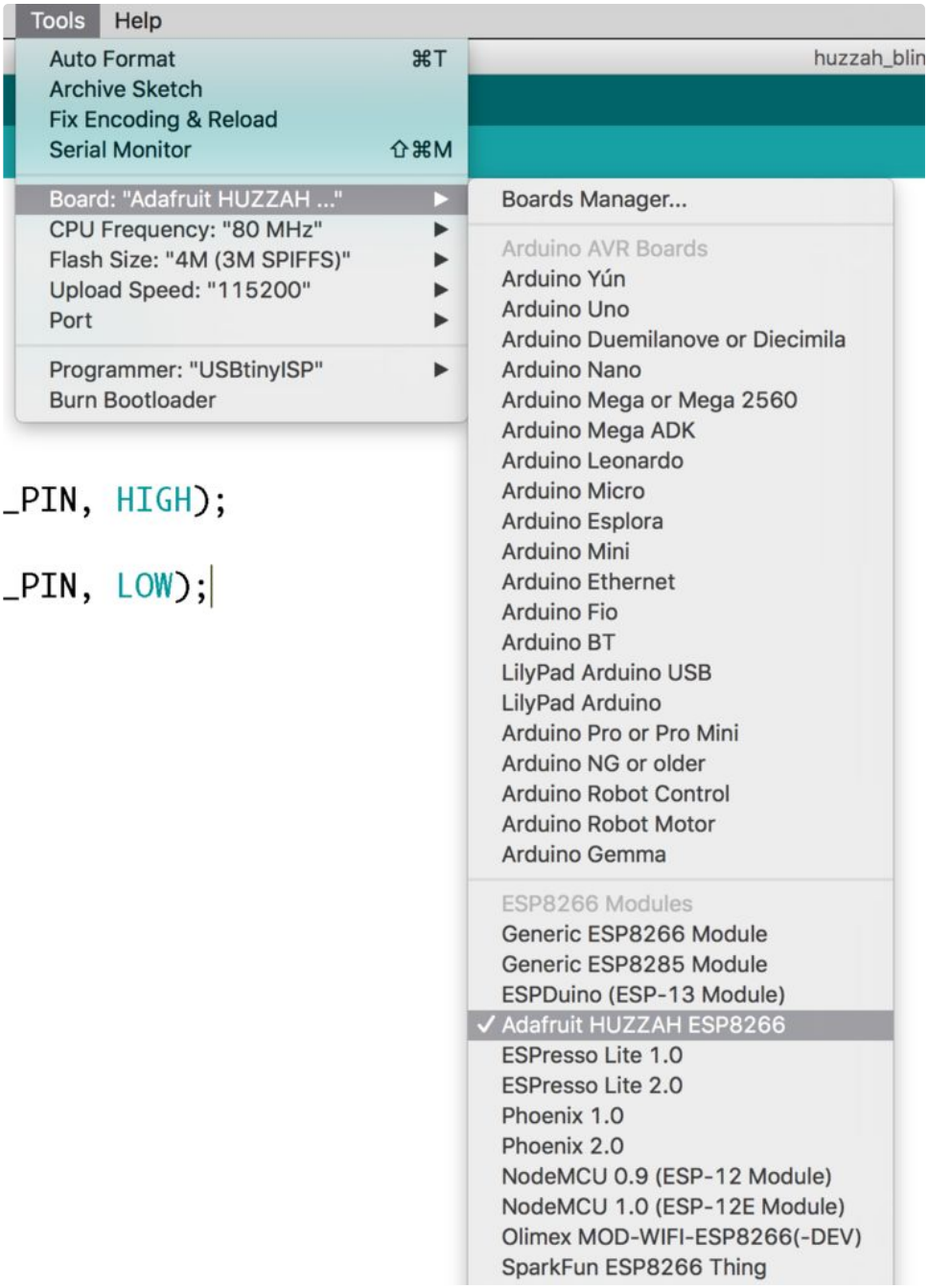
Connect to Wifi

It's time to put some code on your board! Open up a fresh Arduino sketch (File->New) and delete its default contents. Copy this block of code and paste it into the empty sketch:

```
#define LED_PIN 0

void setup() {
  pinMode(LED_PIN, OUTPUT);
}

void loop() {
  digitalWrite(LED_PIN, HIGH);
  delay(500);
  digitalWrite(LED_PIN, LOW);
  delay(500);
}
```

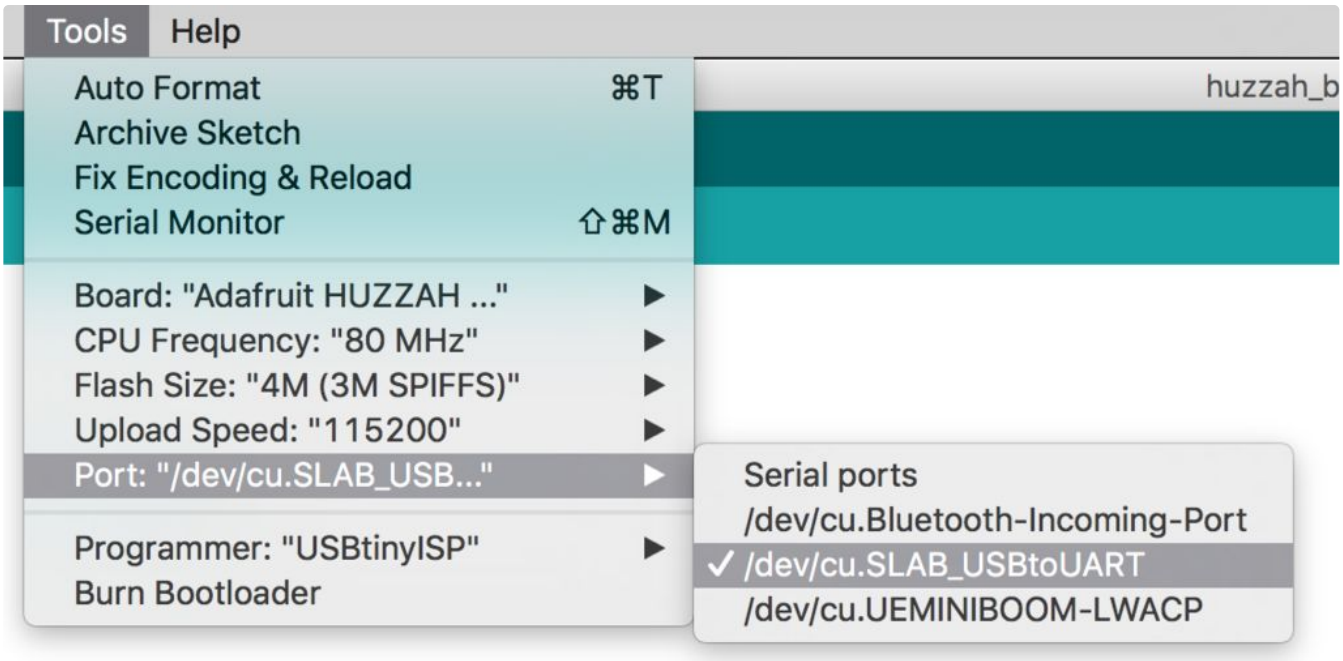
You may recognize this as the very first sketch you ever put on your first Arduino. Indeed we are saying "Hello, world!" once more, this time by blinking the Feather Huzzah's onboard LED (connected to pin 0).

This code sets up a variable for the pin connected to an LED, establishes that pin as an output, then loops an on-off pattern ad infinitum. If any of this code is confusing, review the Arduino lesson that introduces these coding concepts (https://www.instructables.com/lesson/Tools-and-Materials-for-Arduino/).

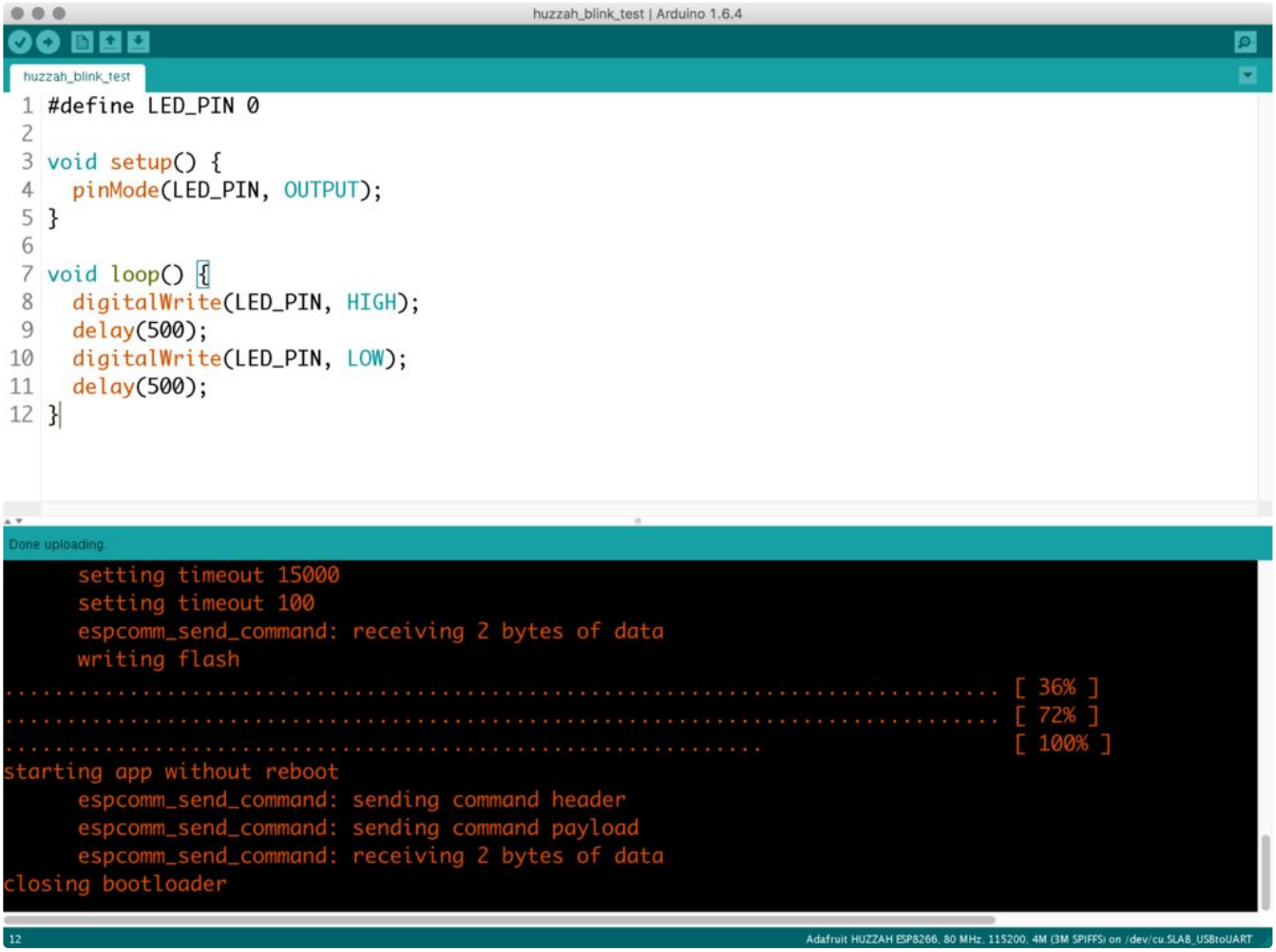


Plug your board into your computer with a USB cable, and check your settings in the Tools menu.

- Board: (Adafruit Huzzah ESP8266) (or name of your board, choose from list)
- Flash Size: "4M (3M SPIFFS)"
- CPU Frequency: "80 MHz"
- Upload Speed: "115200"
- Port should match your serial device (COMx on Windows, /dev/cu.SLAB_USBtoUART on Mac/Linux), which will only appear while plugged in
- Programmer: USBtinyISP



To program your board with the blink sketch, click the Upload button (round button in upper left with arrow icon). The button will turn yellow and the status bar text at the bottom of the window will update you on what's happening. Use the slider bar to adjust the size of the black debugging console below the status bar, and see updates listed as the application compiles your code and uploads it to your board.



This process will seem slower than you're used to when compared to other Arduino compatible boards. Watch the status dots and patiently confirm the text "Done Uploading" appears in the status bar. If get an error message instead, read it and try to figure out what the problem is (wrong port selected, board not plugged in, typo in code). If your board or port isn't appearing in the menus, try these troubleshooting tips and do not proceed until all the software setup steps are complete:

- Under the menu **Arduino -> Preferences**, are there any stray characters or errors in your external boards URL?
- Can you see the ESP8266 boards package in the boards manager?
- If you are using a Feather Huzzah, did you successfully finish installing the SiLabs driver?
- If you are using another board, say with an FTDI programming cable, did you look up any other required installation/setup steps? (Some boards require a combination of button presses to enter bootloader mode.)
- Is your USB cable data+power, or power only?

Try to assess and troubleshoot, but reach out for help (with screenshots or pasted error message text) in the Q&A; section below if you get stuck!

If all goes well, the red LED on your Feather Huzzah should be blinking. If you're using a different board, your LED may be connected to a different pin or not exist at all. The code would still upload successfully but may not cause the desired result (so look up your board or just continue for now). The next sketch you'll upload will get your board connected to the wifi network.

The next code example comes with the ESP8266 boards package and so is already available in your Arduino software. Access it by navigating to **File -> Examples -> ESP8266WiFi -> WifiClientBasic**. Alternatively, you can download the file (https://www.instructables.com/files/orig/FWB/EPUR/J16OV3EC/FWBEPURJ16OV3EC.ino) attached to the end of this step and open it with your Arduino software. However if you don't see the example sketches in your software's menu, you are probably also missing the necessary library files to compile it— go back to the software setup step and double check you've installed the necessary boards package.

```
// We start by connecting to a WiFi network
WiFiMulti.addAP("SSID", "passpasspass");
```

Edit the variables describing the wireless network name(s) and password(s) ("SSID" and "passpasspass"). If your network has no password, leave the password argument empty ("") but do not omit it. Save the sketch and upload it to your board.

Click the button in the upper right of your Arduino window to launch the Serial Monitor, and select 115200 as the baud rate. Press the reset button on your Feather Huzzah to start its program from the beginning, and watch the wireless connection info appear in the Serial Monitor.

You should see a successful connection message followed by the IP address your device has been assigned. If you don't, double check your wifi credentials for typos and try again. If connected, you'll then see a message: "Trying to connect to 192.168.1.1"

Since 192.168.1.1 is a local network location that may or may not exist, the connection most likely fails.

Update the host variable to any site you like, such as "google.com" just to test your connection. Upload the new sketch to your board and open the Serial Monitor to see the different output. Congratulations, your board just talked to the Internet! We'll dive deeper into the code examples in the sample project lessons.

Install Extra Code Libraries

Now that you've got the basic ESP8266 setup working, let's install some additional Arduino libraries that will be used for later lessons. In your Arduino software, navigate to **Sketch-> Include Library -> Manage Libraries...**, then search for and install the latest versions of the following libraries:

- ArduinoHttpClient (https://github.com/arduino-libraries/ArduinoHttpClient)
- Adafruit IO Arduino (https://github.com/adafruit/Adafruit_IO_Arduino)
- Adafruit MQTT (https://github.com/adafruit/Adafruit_MQTT_Library)

You can also install libraries manually by downloading them and putting them in your Arduino libraries folder. Learn more about Arduino libraries in the corresponding lesson in my introductory Arduino class (https://www.instructables.com/lesson/Skills-Infusion/#step3).

Sign Up for (Free) Cloud Services

To complete the lessons in this class, you will need to create free accounts on two cloud service websites: Adafruit IO and IFTTT (If This Then That). Adafruit IO is a **cloud data service** that allows you to set up data streams called **feeds** to collect information coming from your DIY electronics projects. You can visualize and act on these feeds within Adafruit IO, or expand its function by linking it with IFTTT. IFTTT is a site that aggregates and provides an interface for a multitude of applications, and as such is called an **API Gateway**.

Head to io.adafruit.com (https://io.adafruit.com/) and click "Sign In" then "Sign Up" to create an account.

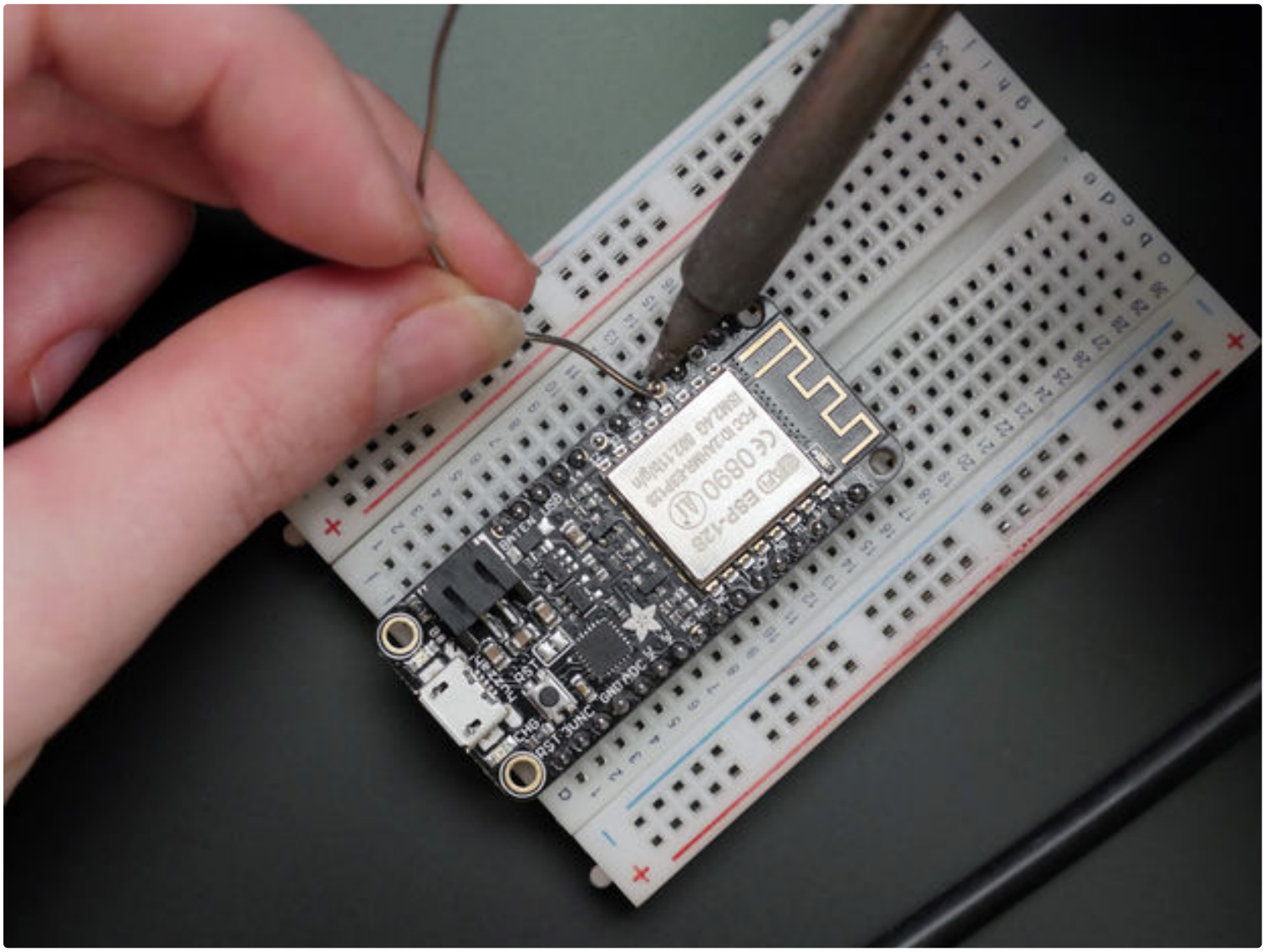
You should use strong, unique passwords. You may be asked to confirm your email address.

On IFTTT.com (https://ifttt.com/), click the "Sign up" button to create an account.

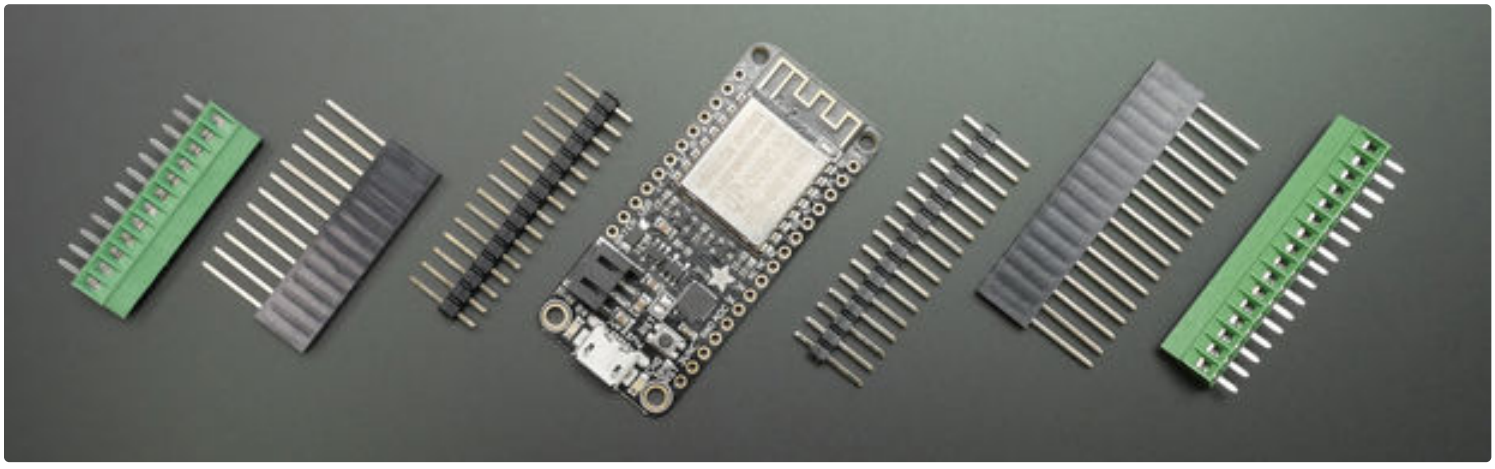
I highly recommend enabling two-factor authentication on your IFTTT account, as you will most likely want to link it to your other personal accounts such as Twitter, Instagram, Fitbit, etc. Keep your accounts safe from hackers and spambots! You will also need to link your Adafruit and IFTTT accounts, which you can do through either site. While you're at it, install the IFTTT app, if you've got an iOS (https://itunes.apple.com/us/app/ifttt/id660944635?mt=8) or Android (https://play.google.com/store/apps/details?id=com.ifttt.ifttt&hl=en) device.

Now that your software is configured, let's dig further into the hardware with the next lesson.

LESSON 3: HARDWARE SETUP

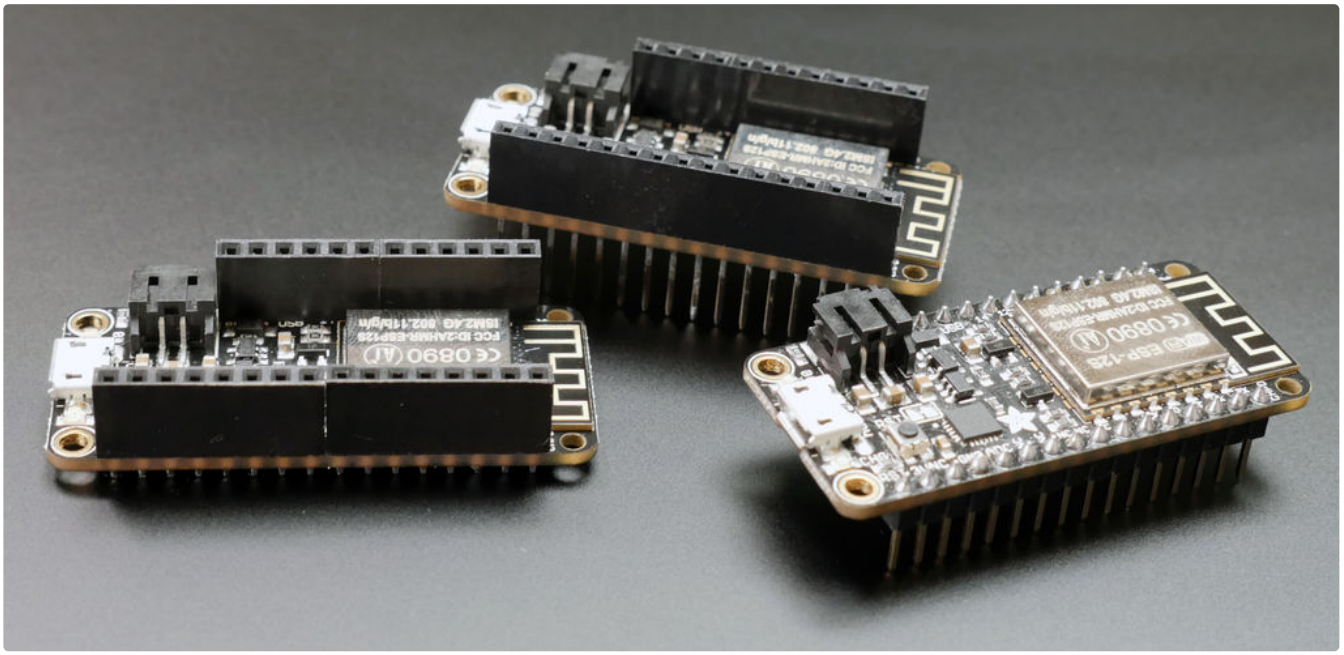


In this lesson, we'll take a closer look at the ESP8266 hardware and the techniques you'll need for creating your own circuits in the rest of the lessons. If you bought a pre-assembled board, you don't absolutely need to know how to solder to complete this class, but a lot opens up to you if you do! Check out the Soldering lesson (https://www.instructables.com/lesson/Soldering-1/) from Randy's Electronics Class to learn how or brush up on your skills.

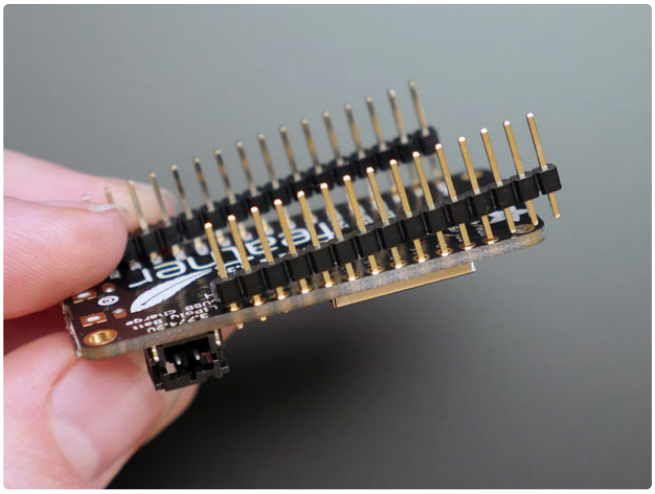


Solder Headers

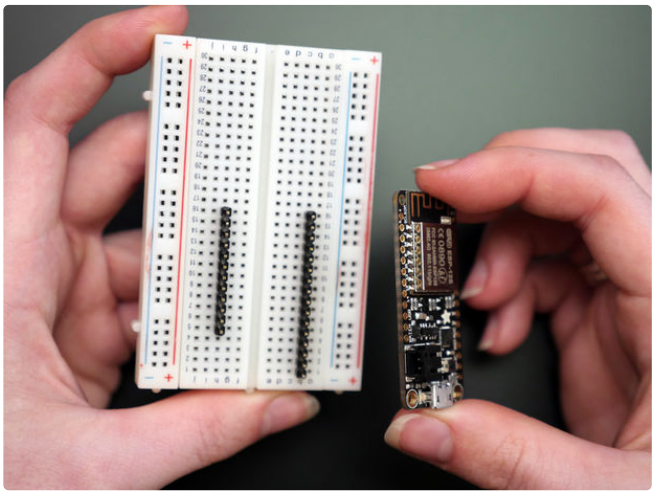
You have a few options when it comes to connecting to your Feather Huzzah board. The connectors we'll attach are called **header pins** (or just "headers" for short), and they are available in several flavors.

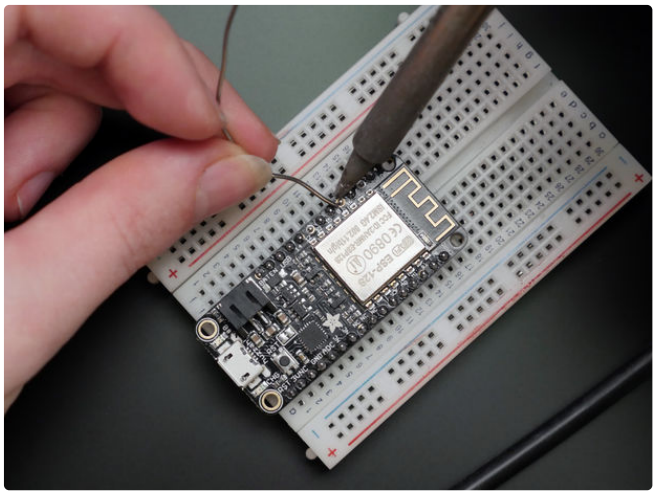


You can purchase the Feather Huzzah and other ESP8266 boards preassembled with male or stacking headers, but if you want terminal blocks or some other way to connect, you'll have to solder them up yourself.

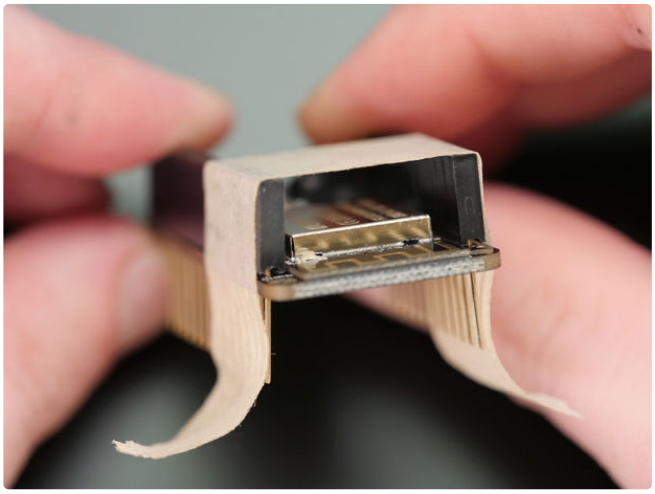


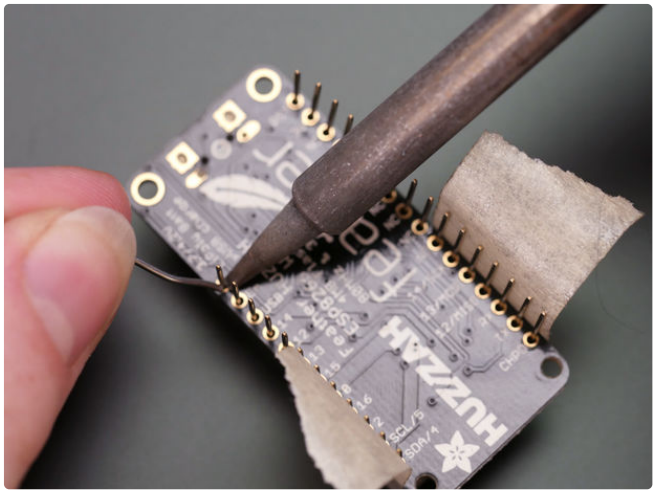
To solder on plain male header pins, first use the board to measure and cut the strips to length, if necessary.



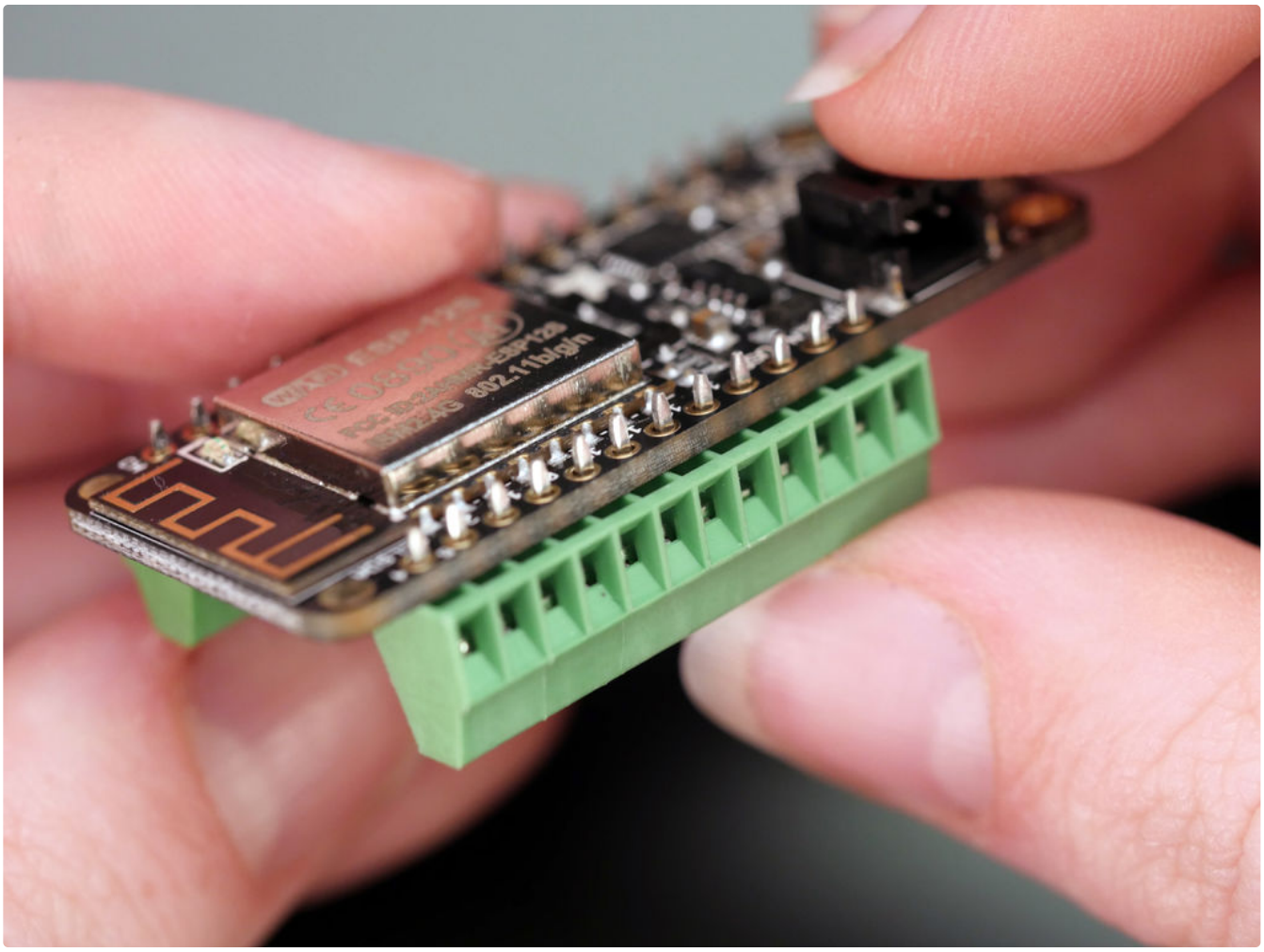


Insert the short end of the pins into the underside of the board and sandwich them into a solderless breadboard to align the pins and clamp them in place while soldering. Solder one pin on each strip to stabilize any wobble, then go back and solder the rest of the pins in place. This same header soldering procedure is handy for lots of other boards and accessories— add it to your toolbox!



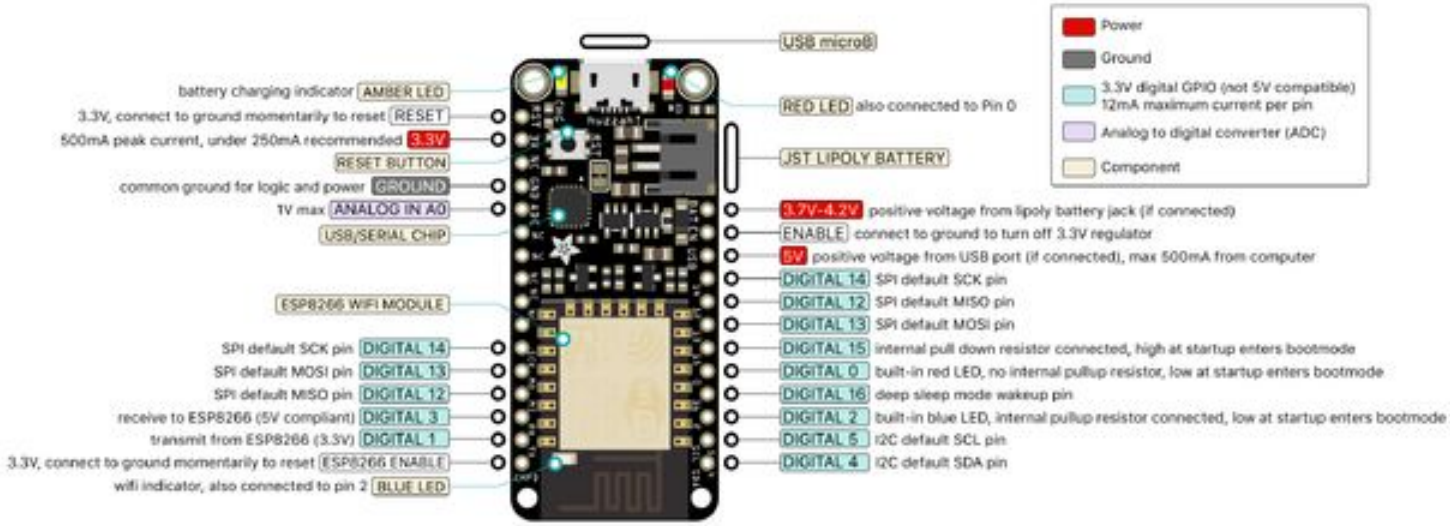


For stacking or female headers, use a piece of tape to secure the headers to the top side of the board. Tack solder one pin on each strip. If it's crooked, you can reheat this one pin while realigning the header strip before soldering the remaining pins.



For wires you need to route into an enclosure, or for components you want to disconnect frequently, terminal block headers are a useful choice. If you want your circuit to be super flat, you may opt to attach wires directly to the board and skip headers entirely.

I usually keep a Feather Huzzah board with stacking headers around, just for prototyping. When I want to build a finalized circuit, I might choose lower profile components, but the convenience of easily plugging and unplugging components works fabulously for the prototyping stage of any project.



Pinout Diagram

This chart shows the various functions of the Feather Huzzah ESP8266 board. **NC** stands for "not connected". This information is consolidated from the (much more detailed) product documentation for the Feather Huzzah ESP8266 (https://learn.adafruit.com/adafruit-feather-huzzah-esp8266/using-nodemcu-lua?view=all#pinouts). Here is a summary of the pins' functions and things to watch out for:

RST - 3.3V reset pin, connect to ground momentarily to reset ESP8266

3.3V pin (500mA peak current, under 250mA recommended to maintain wifi performance)

GND - common ground for logic and power

ADC - analog input (1V max, use a voltage divider if working with a higher voltage signal)

Pin 14/SCK - 3.3V digital GPIO, SPI default SCK pin

Pin 13/MO - 3.3V digital GPIO, SPI default MOSI pin

Pin 12/MI - 3.3V digital GPIO, SPI default MISO pin

(SPI is a protocol for controlling devices though bitbanging (https://en.wikipedia.org/wiki/Bit_banging)*, such as LED Backpacks* (https://learn.adafruit.com/adafruit-7-segment-led-featherwings/overview?view=all)*)*

RX - receive input to the ESP8266 module (5V compliant, has level shifter)

TX - transmit output from the ESP8266 module (3.3V), avoid using if serial communication is active

CHPD - 3.3V enable pin for ESP8266, connect to ground momentarily to reset ESP8266

BAT pin - positive voltage from lipoly battery jack (if connected)

EN pin - connect to ground to turn off 3.3V regulator (everything on board except lipoly charger)

USB pin - positive voltage from USB port (if connected)

Pin 14 - same as SCK above

Pin 12 - same as MI above

Pin 13 - same as MO above

Pin 15 - 3.3V general purpose input/output (digital GPIO), has pull down resistor built in, should not be high at startup to avoid bootmode

Pin 0 - 3.3V digital GPIO with built-in red LED (set LOW to turn on), no internal pullup resistor, should not be low at startup to avoid bootmode

Pin 16 - 3.3V digital GPIO, connect to RST to use this pin to wake from sleep mode

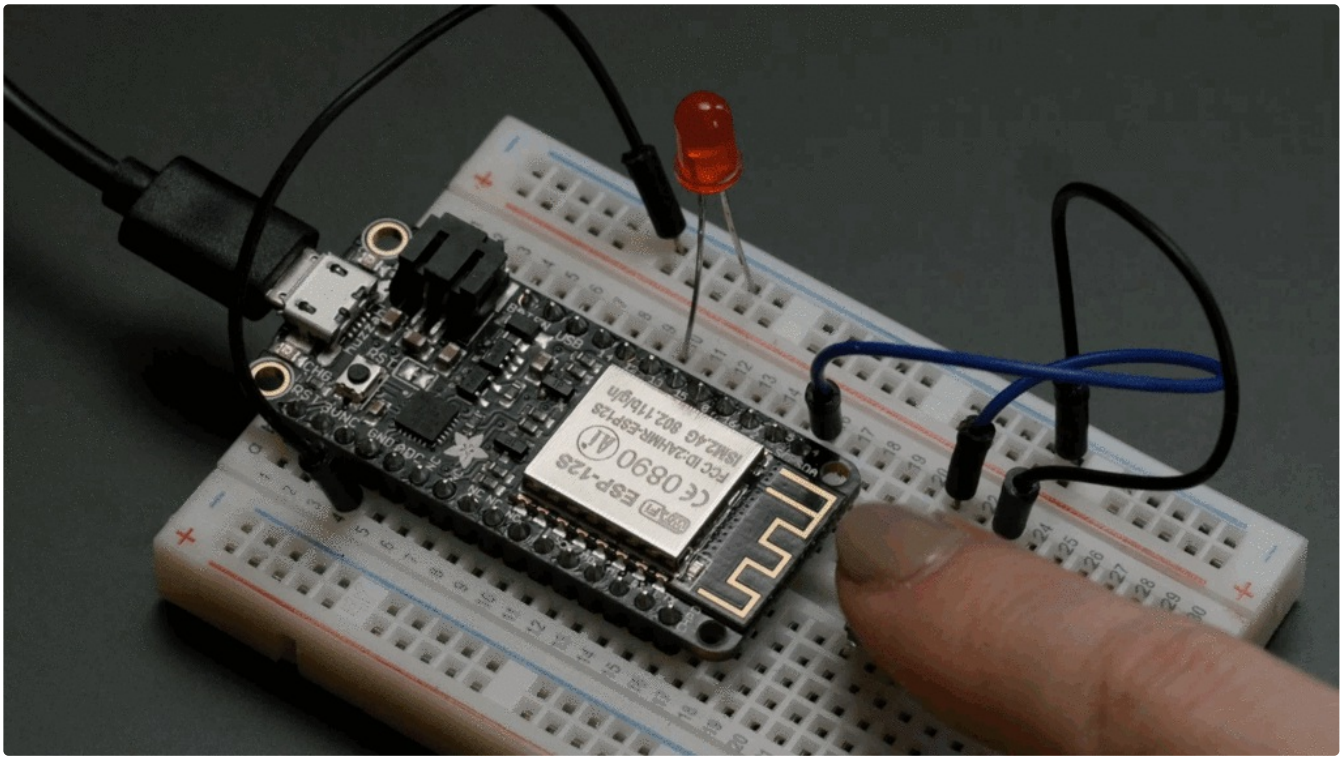
Pin 2 - 3.3V digital GPIO with built-in blue LED (which is also connected to the wifi antenna), internal pullup resistor connected, should not be low at startup to avoid bootmode

Pin 5/SCL - 3.3V digital GPIO, I2C default SCL pin

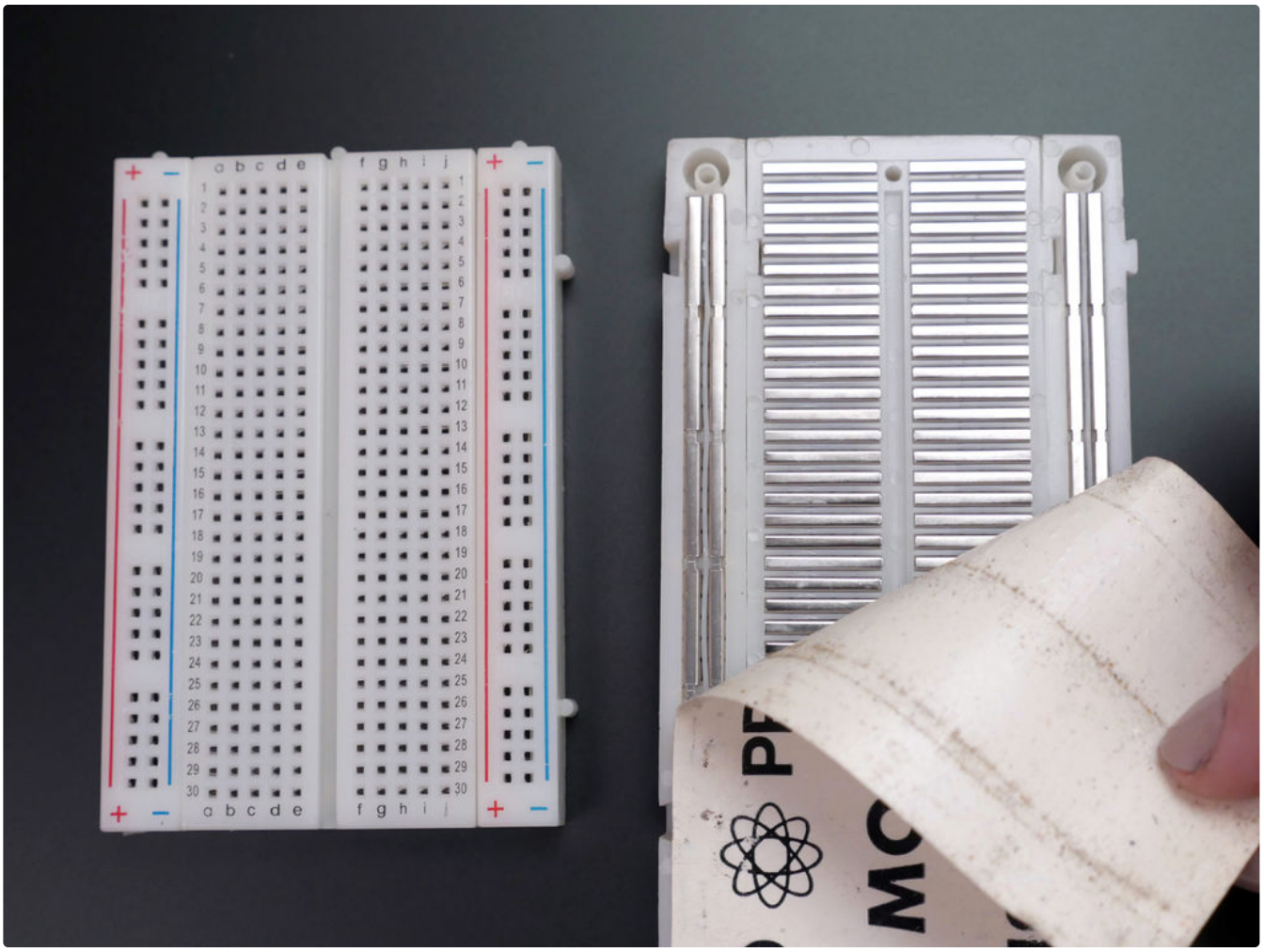
Pin 4/ SDA - 3.3V digital GPIO, I2C default SDA pin

(I2C or "eye squared see" or "eye two see" is a protocol for controlling devices though bitbanging (https://en.wikipedia.org/wiki/Bit_banging), such as LED Backpacks (https://learn.adafruit.com/adafruit-7-segment-led-featherwings/overview?view=all))

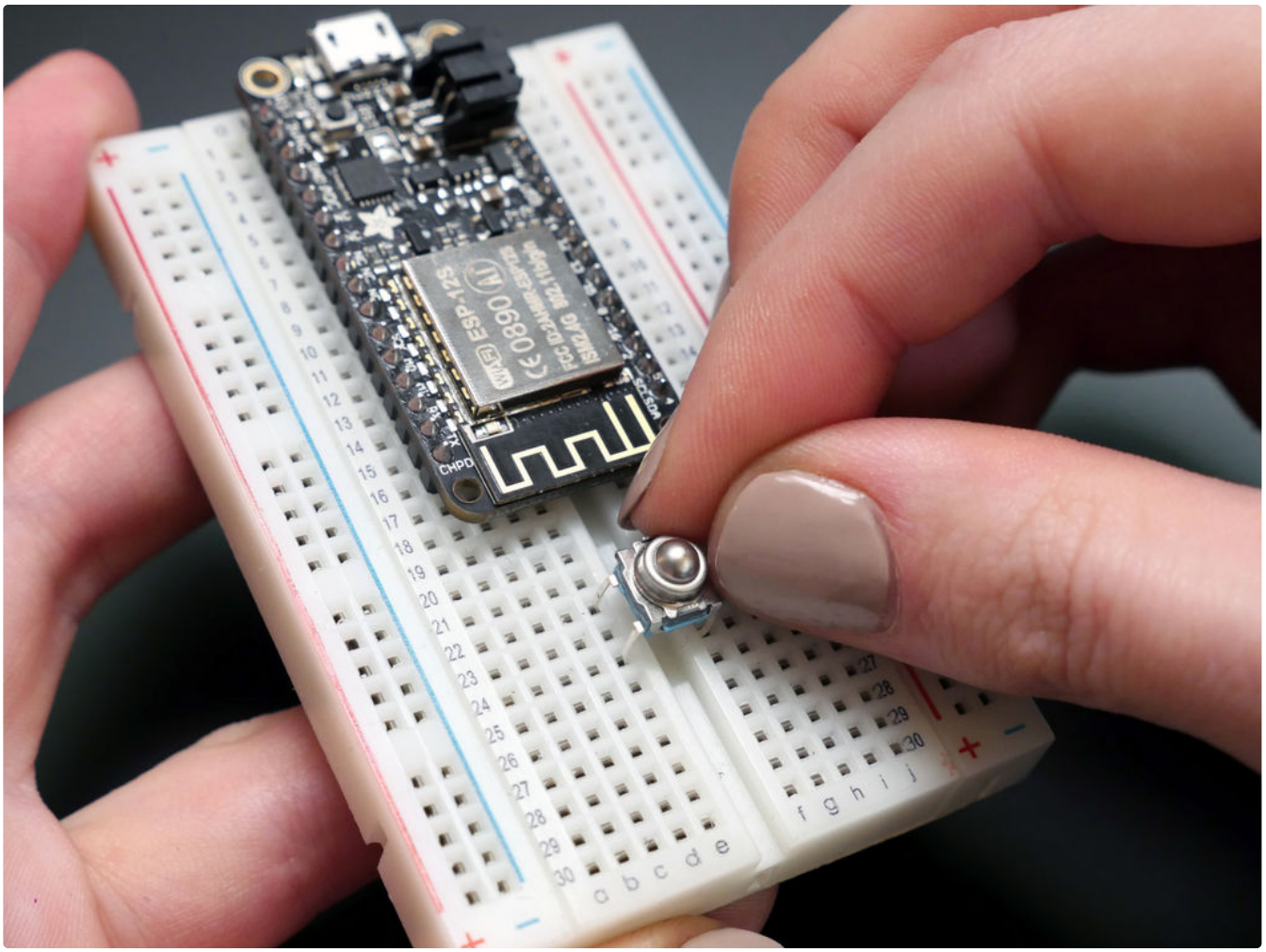
Building Circuits



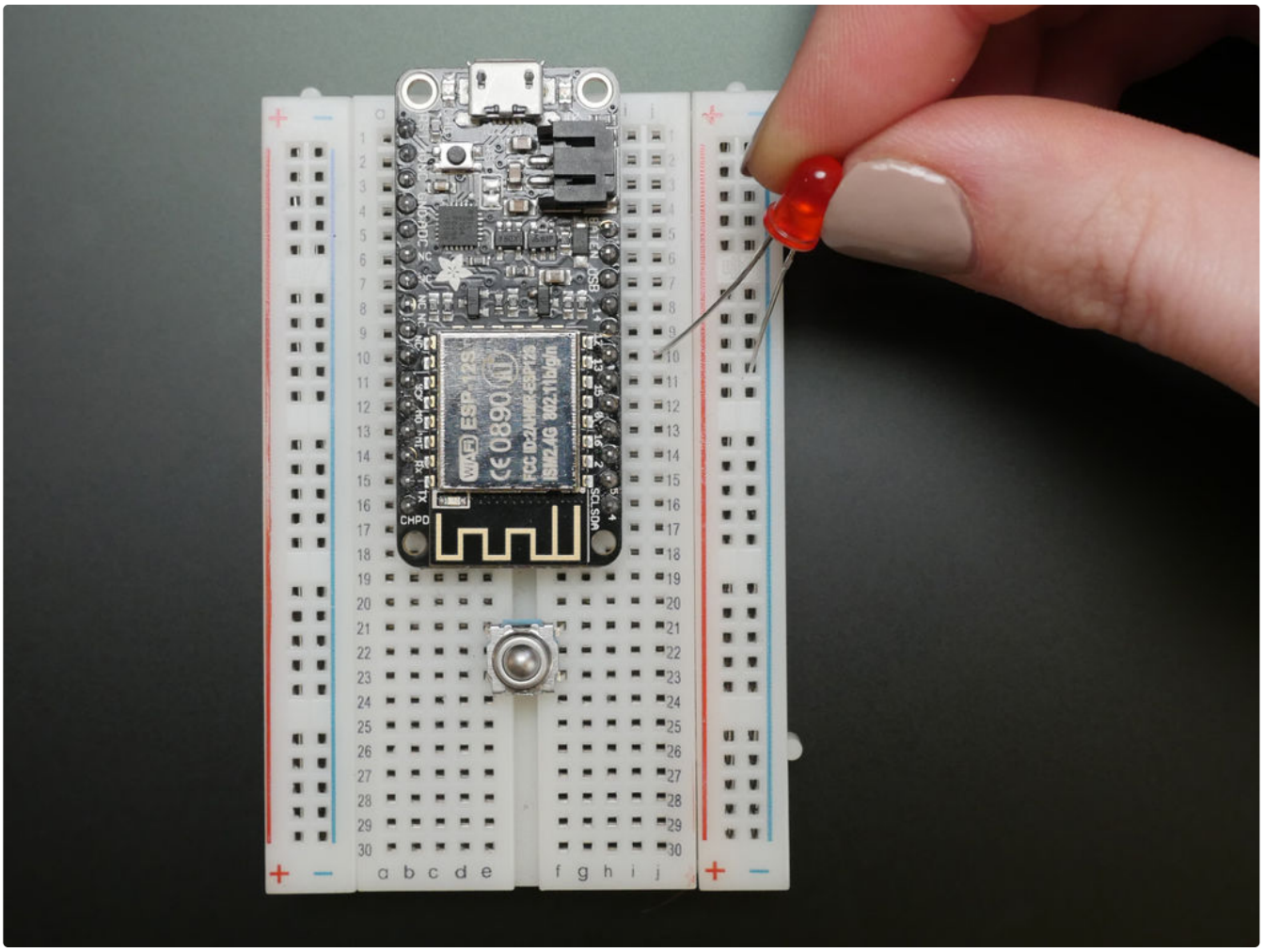
Let's put together a basic circuit that will be useful and relevant to the rest of the lessons. If the solderless breadboard is still a mystery to you, have no fear! First review the introductory lesson in the Arduino class (https://www.instructables.com/lesson/Your-First-Experiments/) to learn how breadboards work.



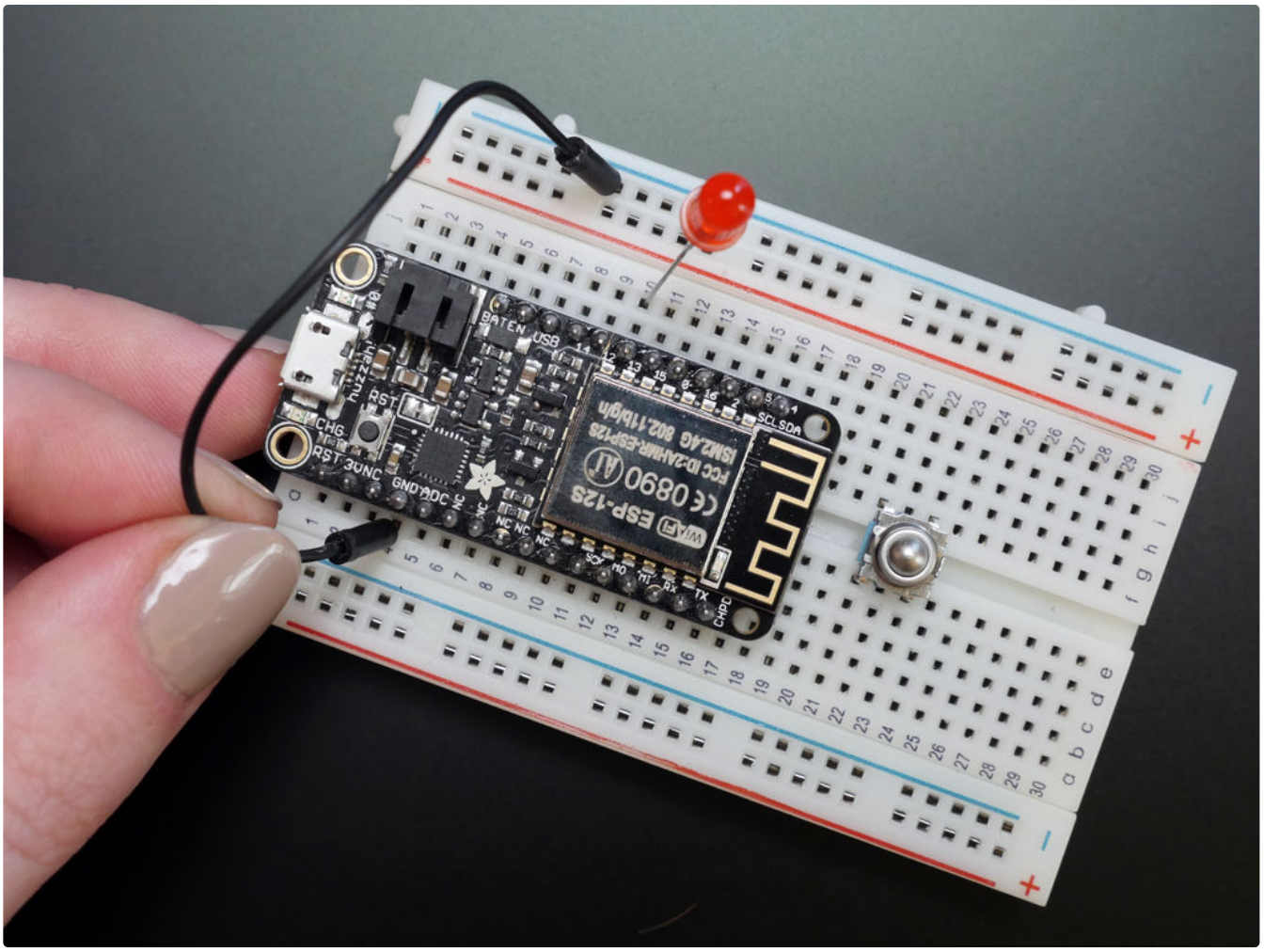
As a quick refresher, remember that the breadboard has long busses that run along each edge, and shorter connectors in the center, separated by a bar down the middle.



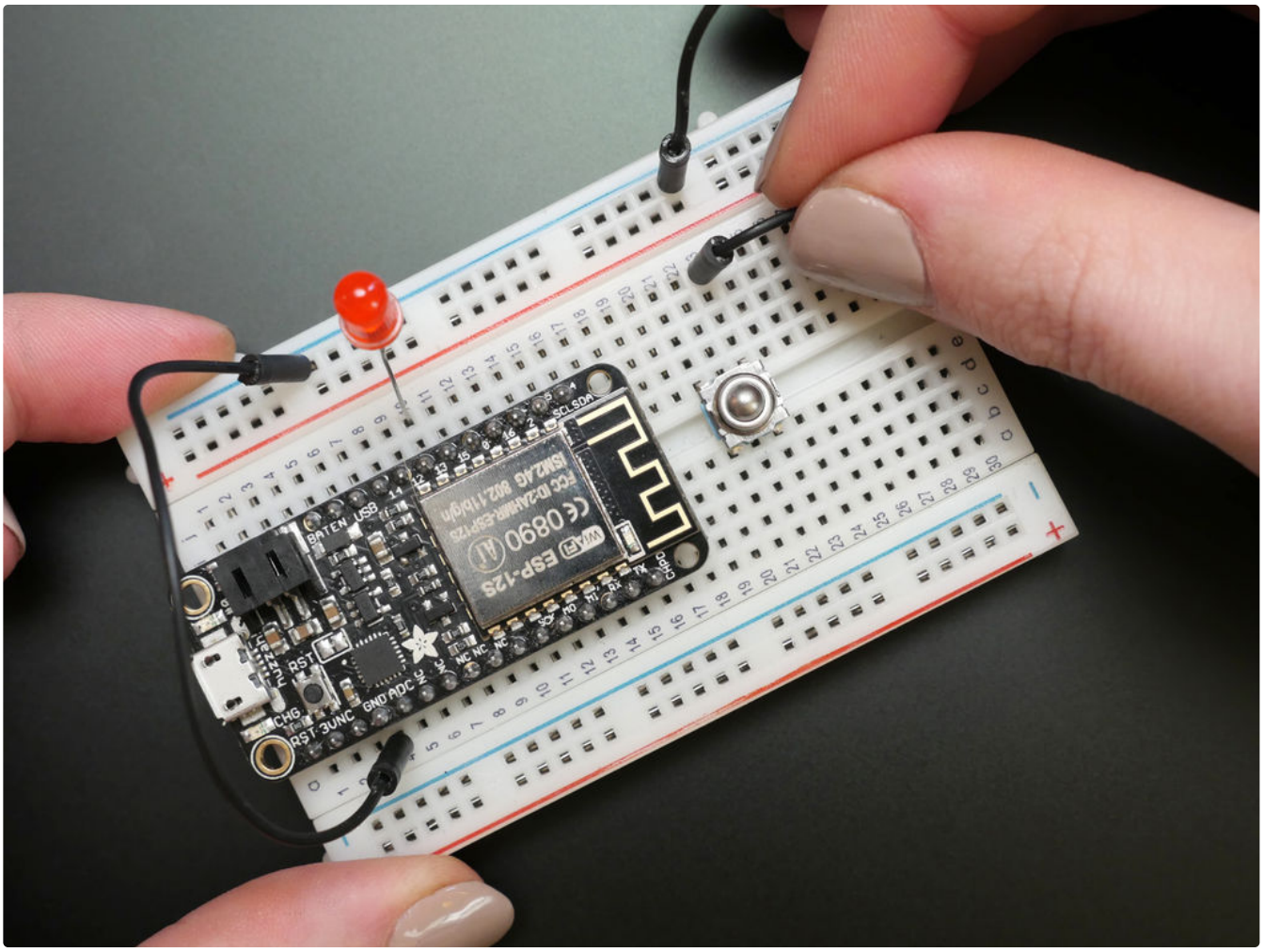
Insert your Feather Huzzah into your solderless breadboard, straddling the middle bar. Do the same with a small pushbutton. Remember to always disconnect power/USB when making wiring changes.



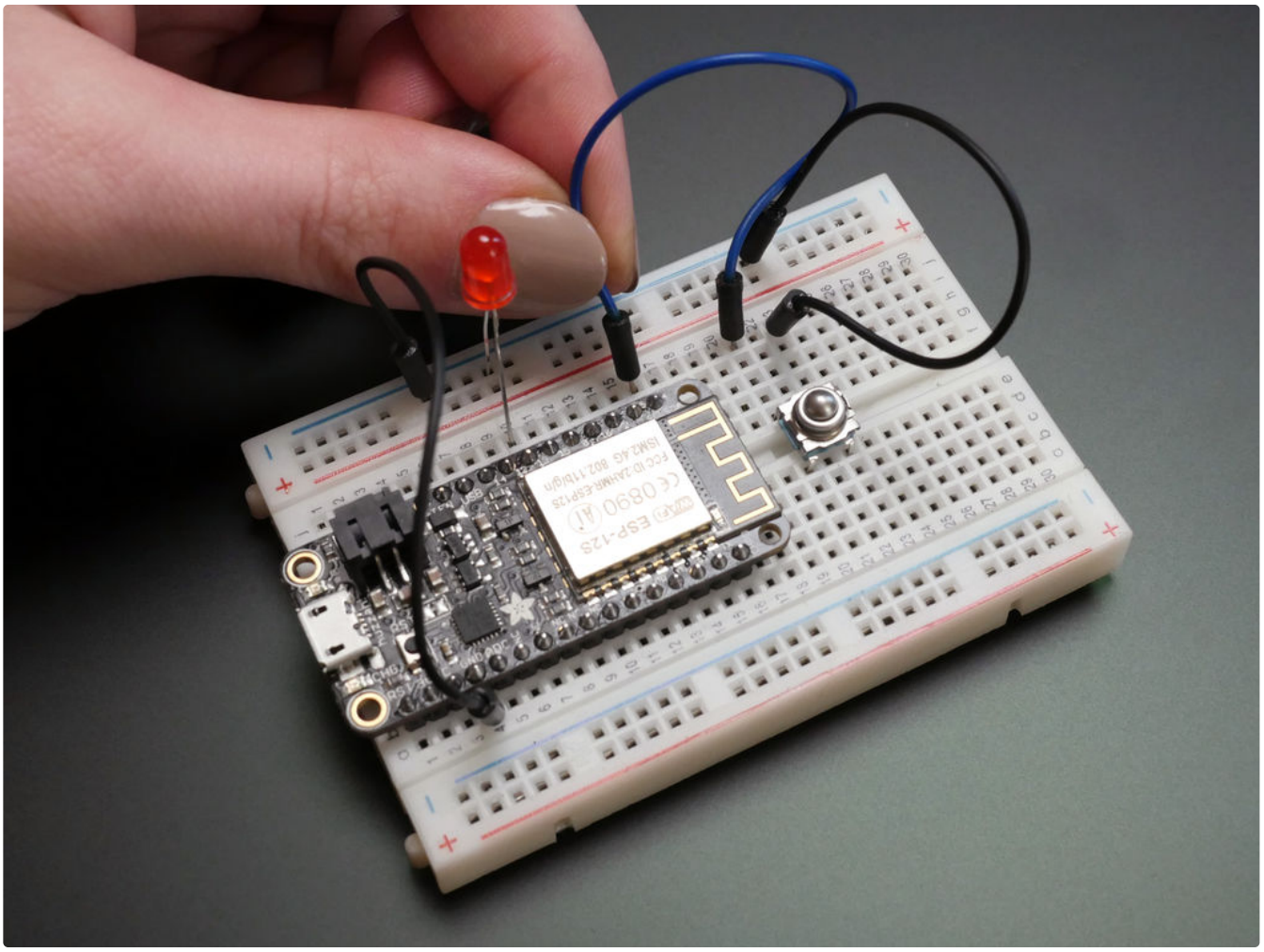
Next, grab an LED and connect its positive (longer) lead to a hole in the breadboard next to pin marked 13 on your board (D7 on NodeMCU). Plug the negative (shorter) lead of the LED into the breadboard's ground bus (any hole along the labeled blue line). The button and LED make up the most basic input and output for your first experiments. Since the Feather Huzzah is a 3V device (Arduino Uno is 5V), I chose not to use a resistor with my LED.



Let's add some wires to connect up our LED and pushbutton. First it's important to establish a common ground, so pick up a wire (any color will do but for convention I'm using black) and plug it into any hole along the ground bus and the hole next to the pin marked GND on your Feather Huzzah board.

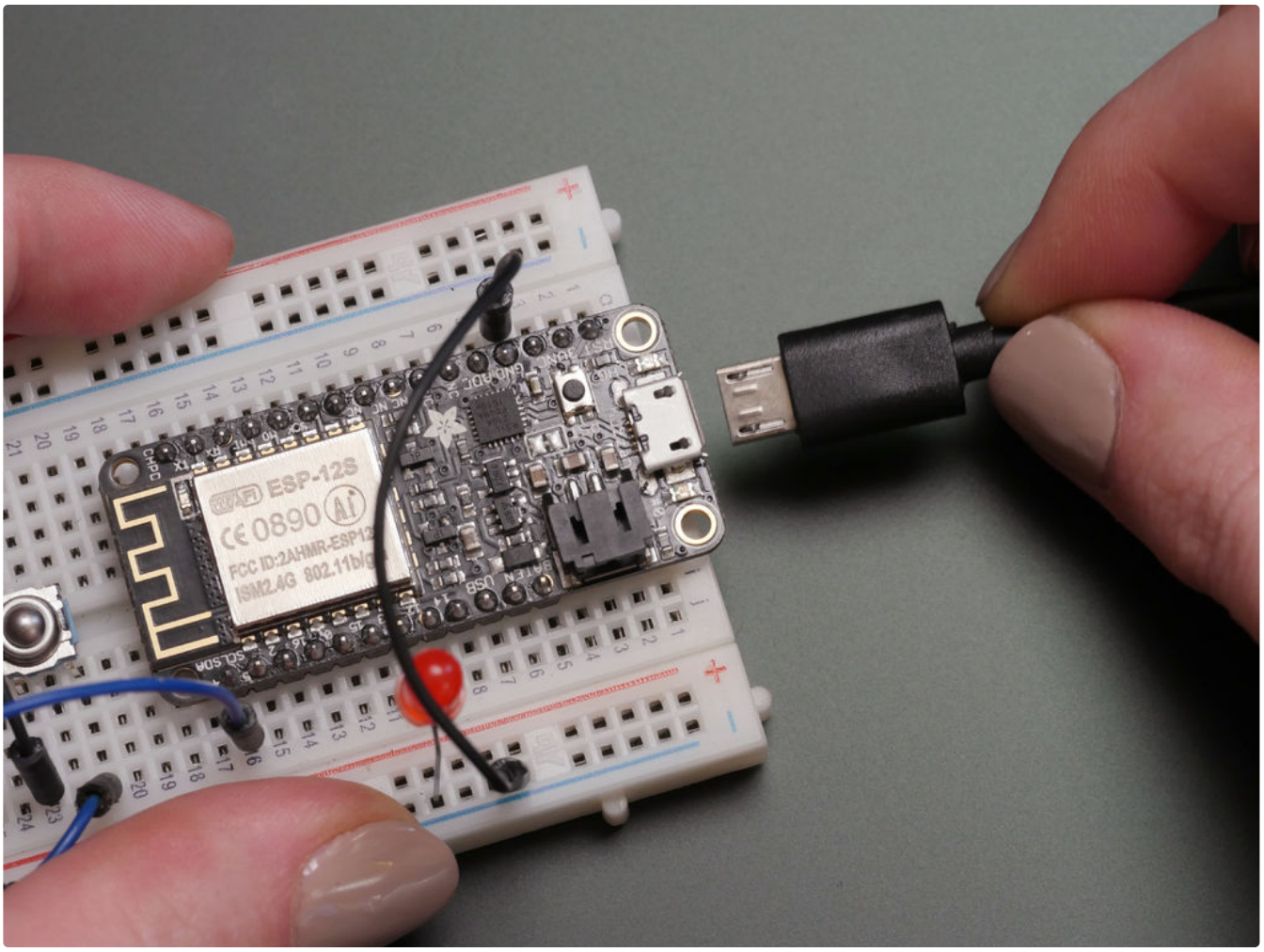


Connect another wire from the ground bus to one of the legs of the pushbutton.



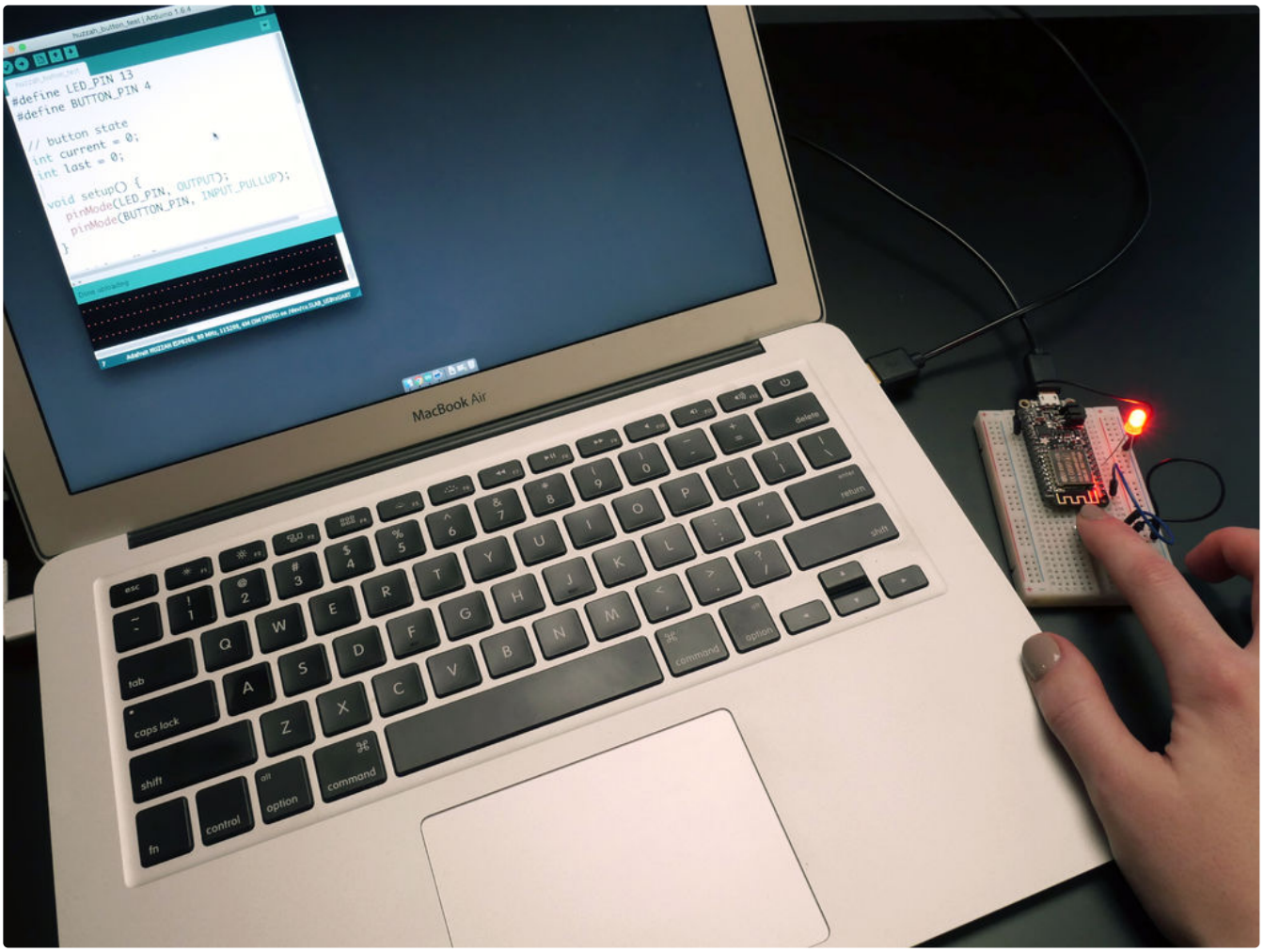
Lastly, connect a wire from another leg of the pushbutton to pin 4 on your board (D2 on NodeMCU). We'll be activating this pin's internal pull-up resistor (https://www.instructables.com/lesson/InputOutput/) in the code in order to read the button.

Double check your circuit against the circuit diagram, then plug in the USB cable to your computer.



Load up this sample code (copy from below or download this step's attachment (https://www.instructables.com/files/orig/FBV/CU0W/J1CETA97/FBVCU0WJ1CETA97.ino) and open in Arduino); it's just a basic button-illuminates-LED circuit that will test your wiring and further cement your understanding of the software/hardware workflow.

```
#define LED_PIN 13
#define BUTTON_PIN 4

// button state
int current = 0;
int last = 0;

void setup() {
  pinMode(LED_PIN, OUTPUT);
  pinMode(BUTTON_PIN, INPUT_PULLUP);
}

void loop() {
  // grab the current state of the button.
  // we have to flip the logic because we are
  // using INPUT_PULLUP.
  if(digitalRead(BUTTON_PIN) == LOW)
    current = 1;
  else
    current = 0;

  // return if the value hasn't changed
  if(current == last)
    return;

  digitalWrite(LED_PIN, current);

  last = current;
}
```



Test that once the code is uploaded to your board, pressing the pushbutton lights up the LED. If it doesn't, check this list of common errors:

- Wire connection missing/incorrect (unplug before making changes)
- LED connected backwards
- Code not uploaded successfully
    - Board type incorrect
    - Port not selected/incorrect
    - Board must be in boot mode (automatic on Feather Huzzah and Node MCU, often achieved with a combination of button presses on other boards)

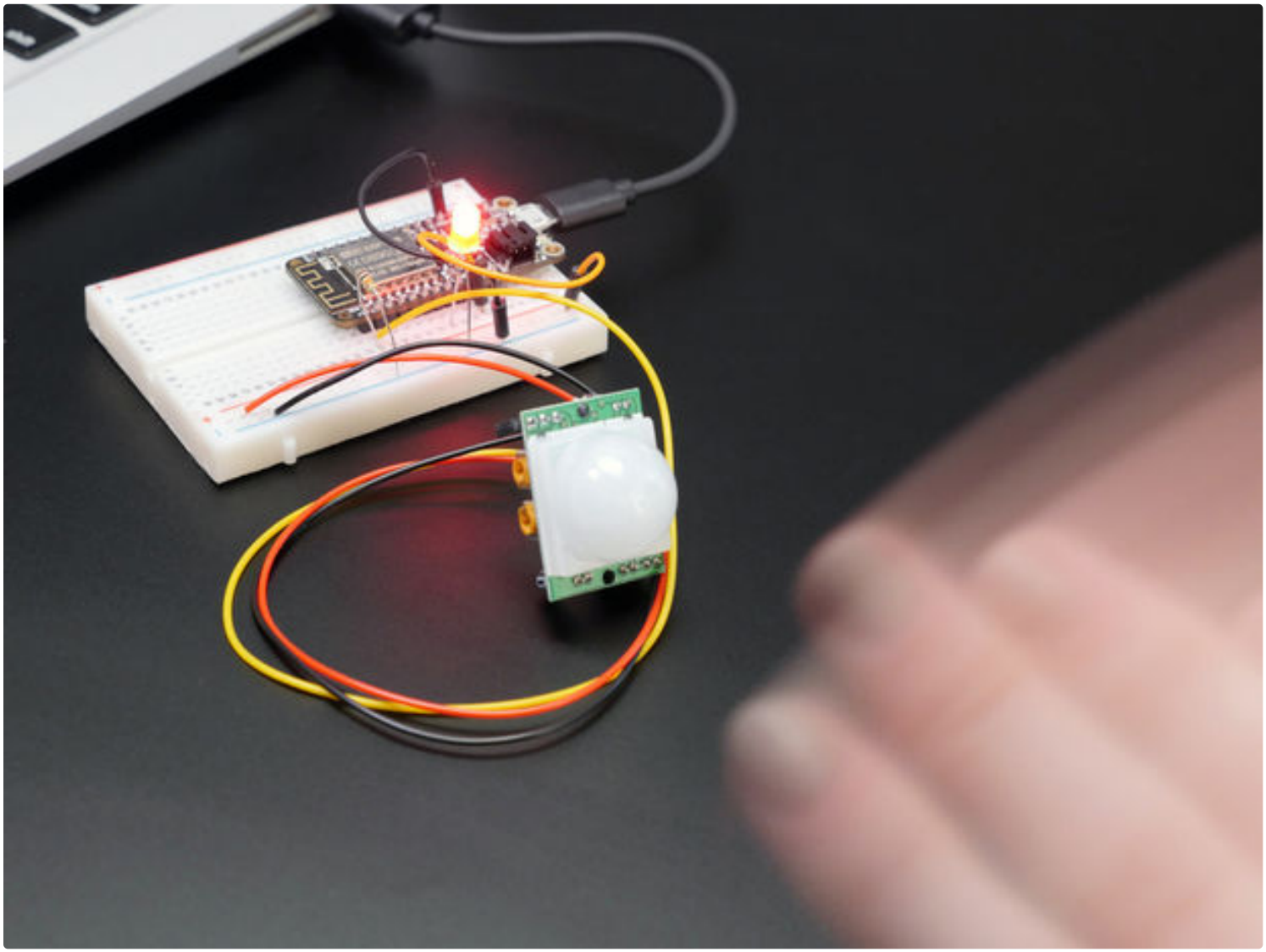
Now you're all set to tackle your first project prototype! Post a celebratory photo of your illuminated circuit in the Class Project module below. Do not proceed until you've got a working pushbutton circuit, as it is the basis for adding internet-connectivity and more complex features later in the class. If you're stuck, look down below for the Ask a Question section!

Download (https://cdn.instructables.com/ORIG/FBV/CU0W/J1CETA97/FBVCU0WJ1CETA97.ino)

http://www.instructable…

(https://cdn.instructables.com/ORIG/FBV/CU0W/J1CETA97/FBVCU0WJ1CETA97.ino)

LESSON 4: CIRCUIT TRIGGERS INTERNET ACTION



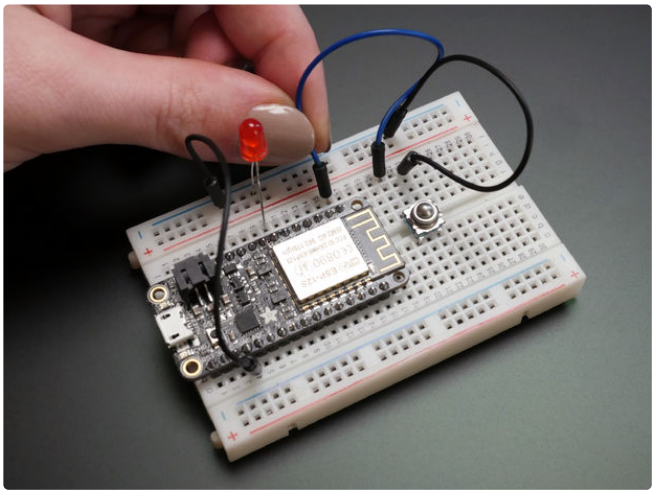
Now is a good time to brainstorm a list of IoT projects you want to make. I have a hunch that many of them would involve a physical circuit sending a simple message. You might want an alert when the water heater's leaking, or when a particular door is opened. In this lesson, we'll build a basic project that triggers an internet action when a physical switch is activated.

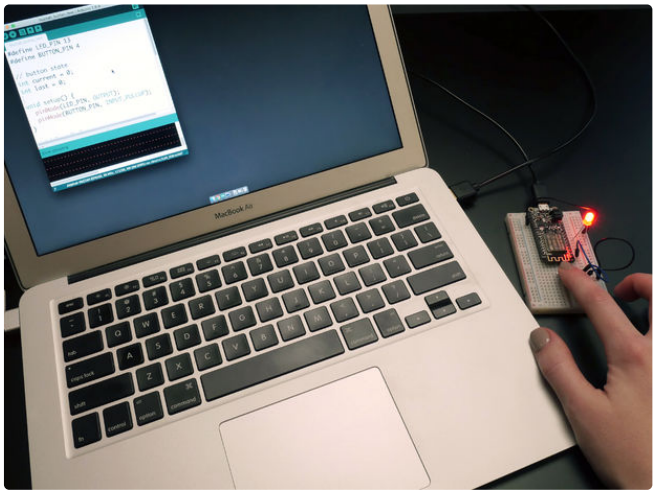
The code will detect the switch and send a message to a feed on the cloud data site Adafruit IO. Another cloud services site, IFTTT, will monitor this feed and send an email when activity is detected (account setup is covered in the Software Setup lesson (https://www.instructables.com/lesson/E6VZE9NIY4QPSLF/#step4)).

This basic circuit and code example makes a great starting point for projects that need to detect and report about a physical event/environment. In later lessons, we'll go over the reverse (displaying information from the internet with a circuit) and combining the two.

Wire Up the Circuit

Wire up and test the pushbutton circuit from the Hardware Setup lesson (https://www.instructables.com/lesson/EXA92B0IY4QPSYL/#step3).





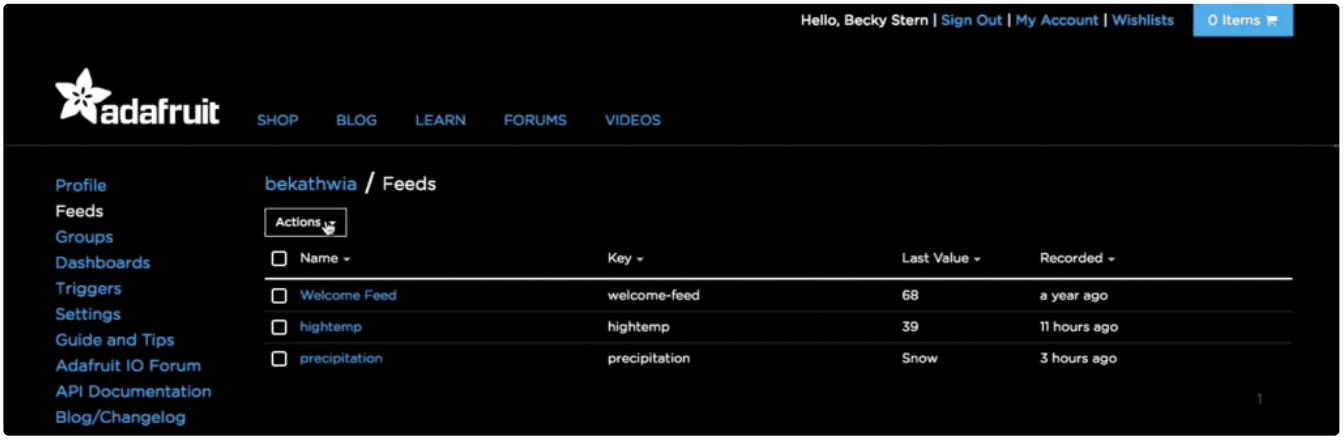
After uploading the pushbutton code (https://cdn.instructables.com/ORIG/FBV/CU0W/J1CETA97/FBVCU0WJ1CETA97.ino) to your board, you should have a circuit that lights up an LED when the button is pressed. Next we'll add in some code to send a message to Adafruit IO each time the button is pressed, and create a corresponding feed to catch that incoming data.

Download (https://cdn.instructables.com/ORIG/FBV/CU0W/J1CETA97/FBVCU0WJ1CETA97.ino)

http://www.instructable...

(https://cdn.instructables.com/ORIG/FBV/CU0W/J1CETA97/FBVCU0WJ1CETA97.ino)

Create Adafruit IO Web Feed



Create a new feed in your Adafruit IO account, and name it "command."

Find your AIO key on the Settings page.

Upload Code to Connect to Feed

Download the project code (https://cdn.instructables.com/ORIG/FNN/461M/J1GOVOOQ/FNN461MJ1GOVOOQ.ino) attached to this step and open it with the Arduino IDE, then update the settings to match your Adafruit IO username and key as well as your wifi network name and password. Be sure you have the Arduino libraries "ArduinoHttpClient", "Adafruit IO Arduino", and "Adafruit MQTT" installed. Upload the code to your board.

Download (https://cdn.instructables.com/ORIG/FNN/461M/J1GOVOOQ/FNN461MJ1GOVOOQ.ino)

http://www.instructable...

(https://cdn.instructables.com/ORIG/FNN/461M/J1GOVOOQ/FNN461MJ1GOVOOQ.ino)

Troubleshooting With the Serial Monitor

The serial monitor (https://www.instructables.com/lesson/InputOutput/) is an indispensable tool in your Arduino toolbox. The sample sketch is composed with helpful status updates that print to the serial monitor while the board is connected to the computer. In the Arduino software, you can open it up by clicking the magnifying glass button in the upper right of your sketch window, and set the baud rate to 115200 (matches the serial port created in the code's setup function). Press the reset button on your board to start its sketch from the beginning, and see what info pops up in the monitor.

Does everything appear to be in order with no errors in the Arduino software? Cool, that means you've probably got data in Adafruit IO to check out! Navigate to your "command" feed in the browser and check to see what your data input looks like so far. (Once logged in to io.adafruit.com, click on "Feeds" on the left side of the page, then click on the name of your feed, "command".)

Once again, if you get an error, try to read it and troubleshoot the problem with a few things you know to look out for:

- Board connected with data-capable cable
- Correct board and port selected in tools menu
- AIO and wifi credentials typed/pasted correctly
- Internet connection is active

And a few that may be new:

- Check to see that Adafruit IO is up with no issues (check blog/forum linked on left side of page at io.adafruit.com)
- Incorrect baud rate selected in Serial Monitor
- Feed name in Arduino sketch does not match AIO feed name

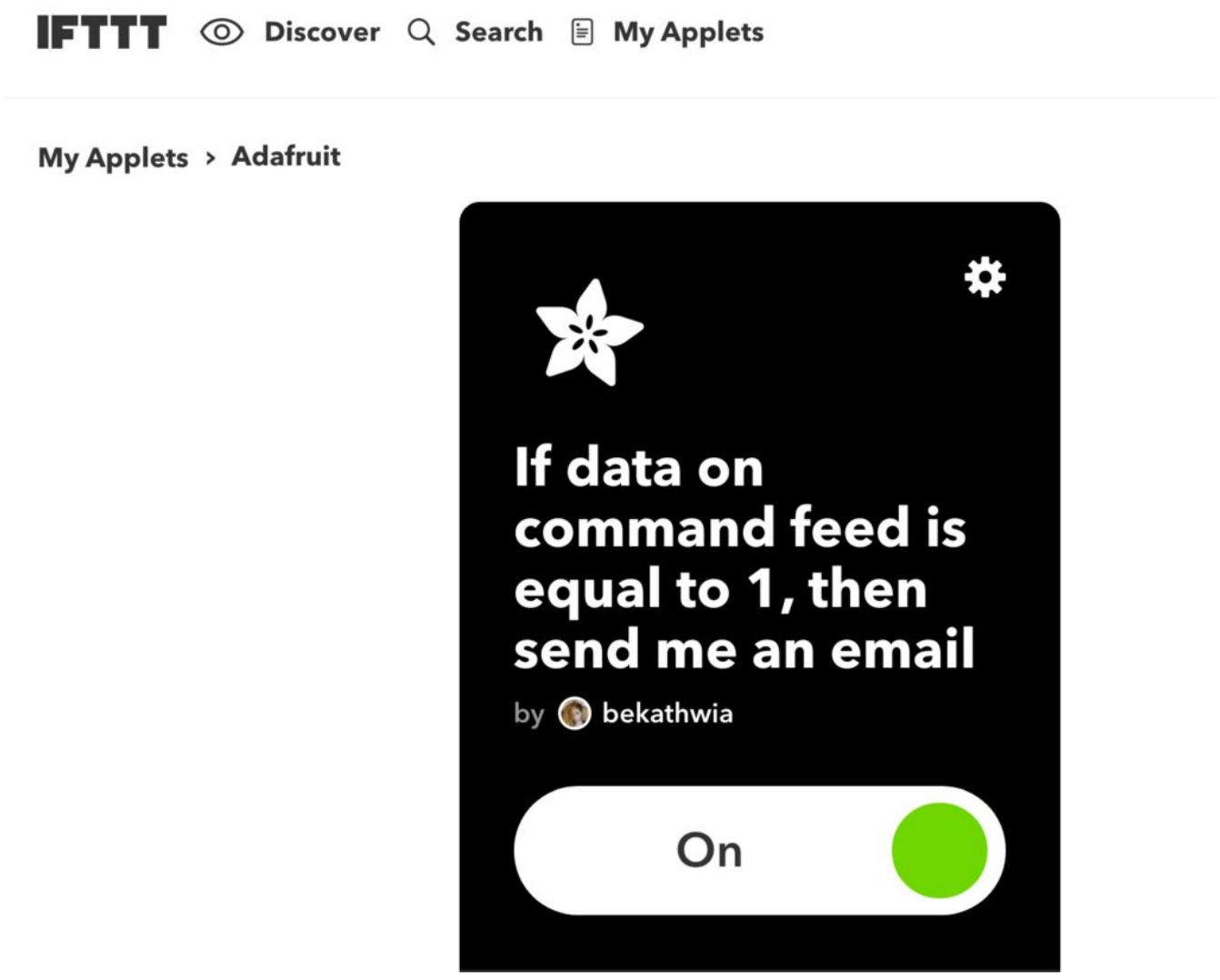
The resulting graph shows each time you pressed the button, along an axis of time. Now that we've got a working prototype that sends data to Adafruit IO, let's add an online service to watch the feed and take actions based on it.

Connect IO Feed to IFTTT

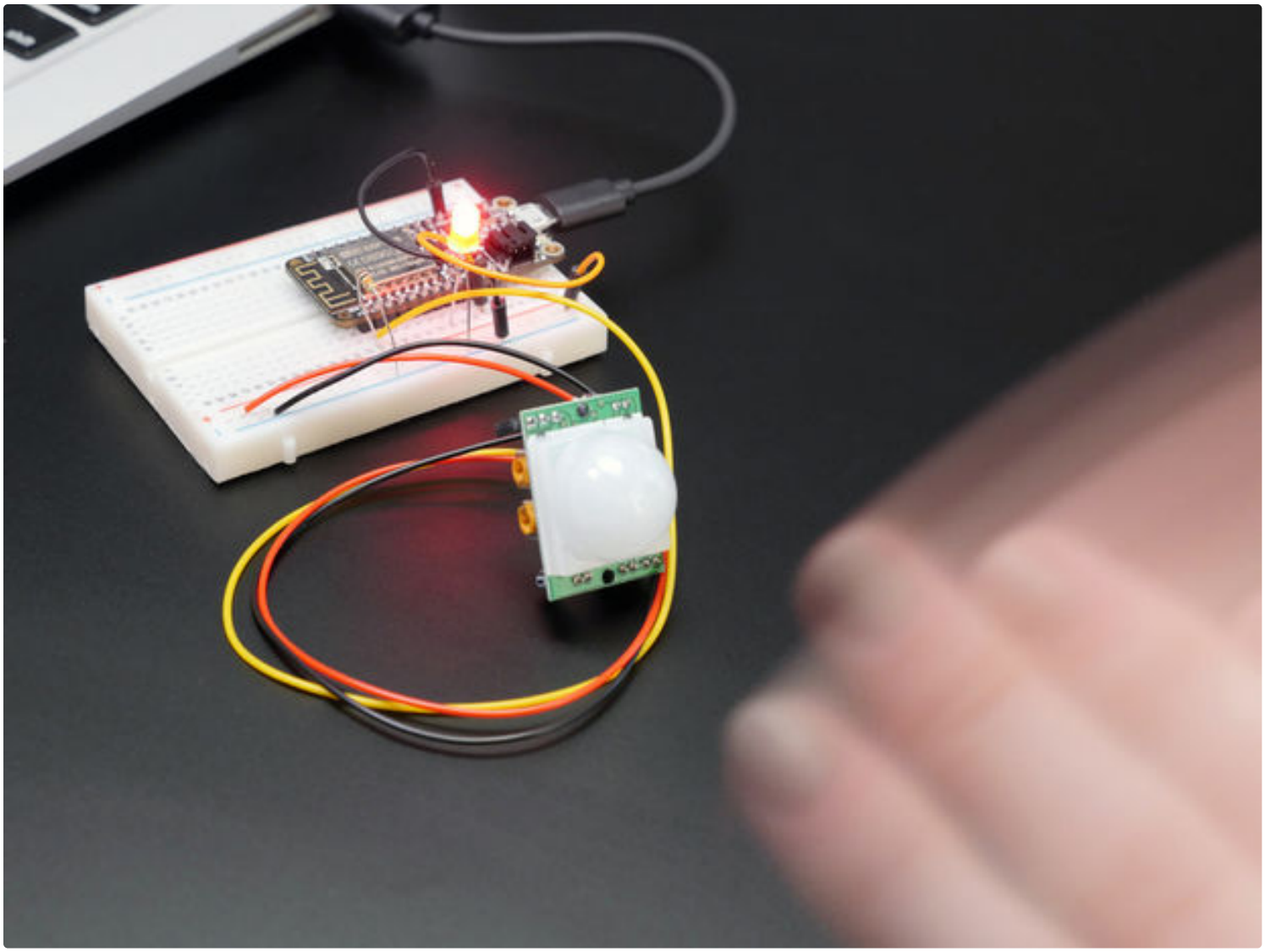
Sign in to your IFTTT account and click My Applets at the top of the page, then the button marked "New Applet" on the right of page. Click "+this" to choose an input service, and start typing "Adafruit" into the search bar until it appears, then click the button marked Adafruit to see the available triggers. For this example, either trigger will work, but for the sake of showing more settings, choose "Monitor a feed on Adafruit IO."

From the dropdown menus, select the feed name "command" and the relationship "equal to". Type the number 1 into the Value field, then click the button marked "Create trigger".

Next, click "+that" to choose an output service, and then click "email" (or choose a different output service like Twitter, SMS, etc.). Click through to customize the email message, then click the button marked "Create action". Lastly, customize the name of the applet (if you wish) and click the button marked "Finish".



Now when you press the button, you should see an email in your inbox (up to a few minutes later).



Switch Up Your Input

Buttons are all well and good, but they are just the tip of the input iceberg. Now that you've got a basic switch circuit working, you can swap the button out for any normally open switch without changing the code (or at most, changing line 17 to evaluate for `HIGH` instead of `LOW`). Here are some ideas:

- Magnetic reed switch (https://www.adafruit.com/product/375)
- Vibration switch (https://www.adafruit.com/?q=vibration%20switch&)
- Tilt switch (https://www.adafruit.com/product/173)

Learn more in the Switches lesson (https://www.instructables.com/lesson/Switches/) of Randy's Electronics Class!

One particularly fun upgrade to this circuit is to add a PIR motion sensor (https://www.adafruit.com/product/189). PIR stands for Passive Infrared, which describes how the sensor works— by detecting changes in the overall amount of IR light it can "see" with its collector dome. This sensor is designed to output a simple HIGH/LOW signal when its (adjustable) threshold is exceeded, so

we can easily use it in place of a pushbutton using the same circuit we've been using all along. In addition to the PIR motion sensor, grab a one more wire (for powering the sensor), a 10K ohm resistor (brown, black, orange, gold) and wire them up according to the following circuit diagram:

The PIR motion sensor comes with a wired three-pin connector that plugs into three header pins on the sensor board— the black wire should line up with the pin marked GND.

The code to test the motion sensor is very similar to the button test code, with some minor (but significant) changes. Since the motion sensor sends a `HIGH` signal when it detects movement, we'll need to remove the internal pull-up resistor from the `pinMode` declaration in the code's setup, and change the main `if` statement to check for a `HIGH` signal instead of a `LOW`.

```
#define LED_PIN 13
#define MOTION_PIN 4

// button state
int current = 0;
int last = 0;

void setup() {
  pinMode(LED_PIN, OUTPUT);
  pinMode(MOTION_PIN, INPUT);
}

void loop() {
  // grab the current state of the button.
  if(digitalRead(MOTION_PIN) == HIGH)
    current = 1;
  else
    current = 0;

  // return if the value hasn't changed
  if(current == last)
    return;

  digitalWrite(LED_PIN, current);

  last = current;
}
```

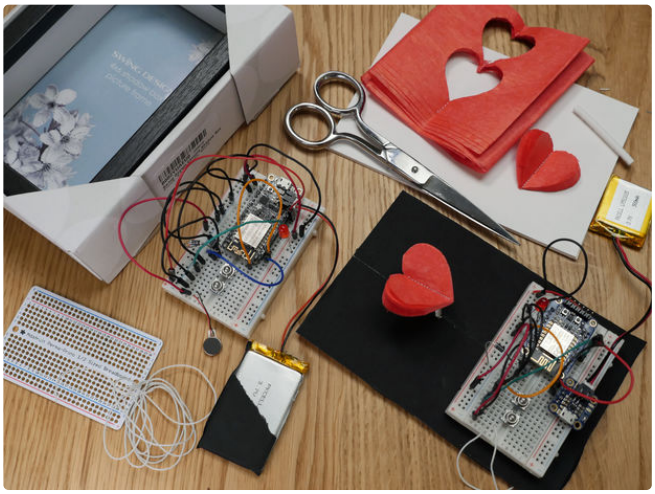
Copy or download this code (https://cdn.instructables.com/ORIG/FB9/YM4G/J1CETARL/FB9YM4GJ1CETARL.ino) into your Arduino software (a new blank sketch, please), or if you're feeling adventurous, make the two small changes we just mentioned to your button test code by hand and save the file with a new name, such as **huzzah_PIR_test.ino**.

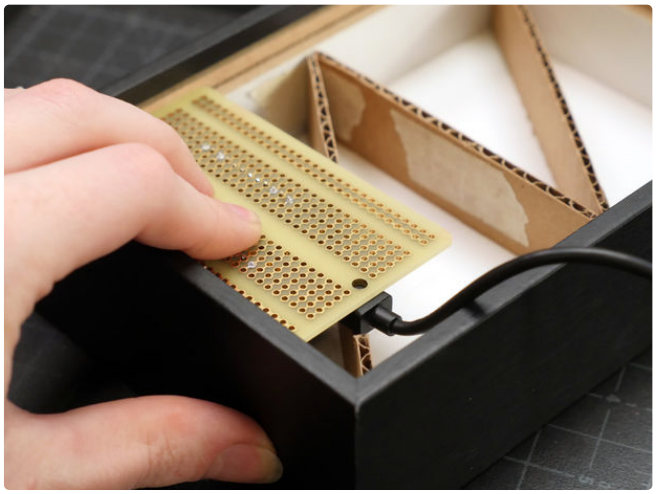
Upload the PIR test code to your Feather Huzzah and test that waving your hand in front of the sensor causes the LED to light up for a few seconds, then turn off again. You can adjust the sensitivity of the trigger and the duration of the HIGH signal by turning the sensor board's tiny potentiometers with a small screwdriver.

Taking It Further

The Feather Huzzah has several pins capable of reading a digital input. Avoiding the special pins that cause the board to enter bootmode, the logical choices for adding more switches are pins 5, 16, 13, 12, and 14.

Using an analog sensor like a photocell or potentiometer (https://www.instructables.com/lesson/InputOutput/#step4) with the ESP8266 is a bit trickier than just plugging it in and calling `analogRead();` because the analog to digital converter (ADC) has a 1V maximum. To scale down a 3 or 5 volt sensor signal, you must build a voltage divider (https://www.instructables.com/lesson/Resistors/#step7), which is composed of a few carefully selected resistors. There is only one ADC on the ESP8266, and it's in code it's called **A0**. For example, you might divide the 3V resisted by a light-dependent resistor with a 10K ohm resistor (https://learn.adafruit.com/adafruit-io-basics-analog-input/wiring) to scale its output to within the 0-1V range.



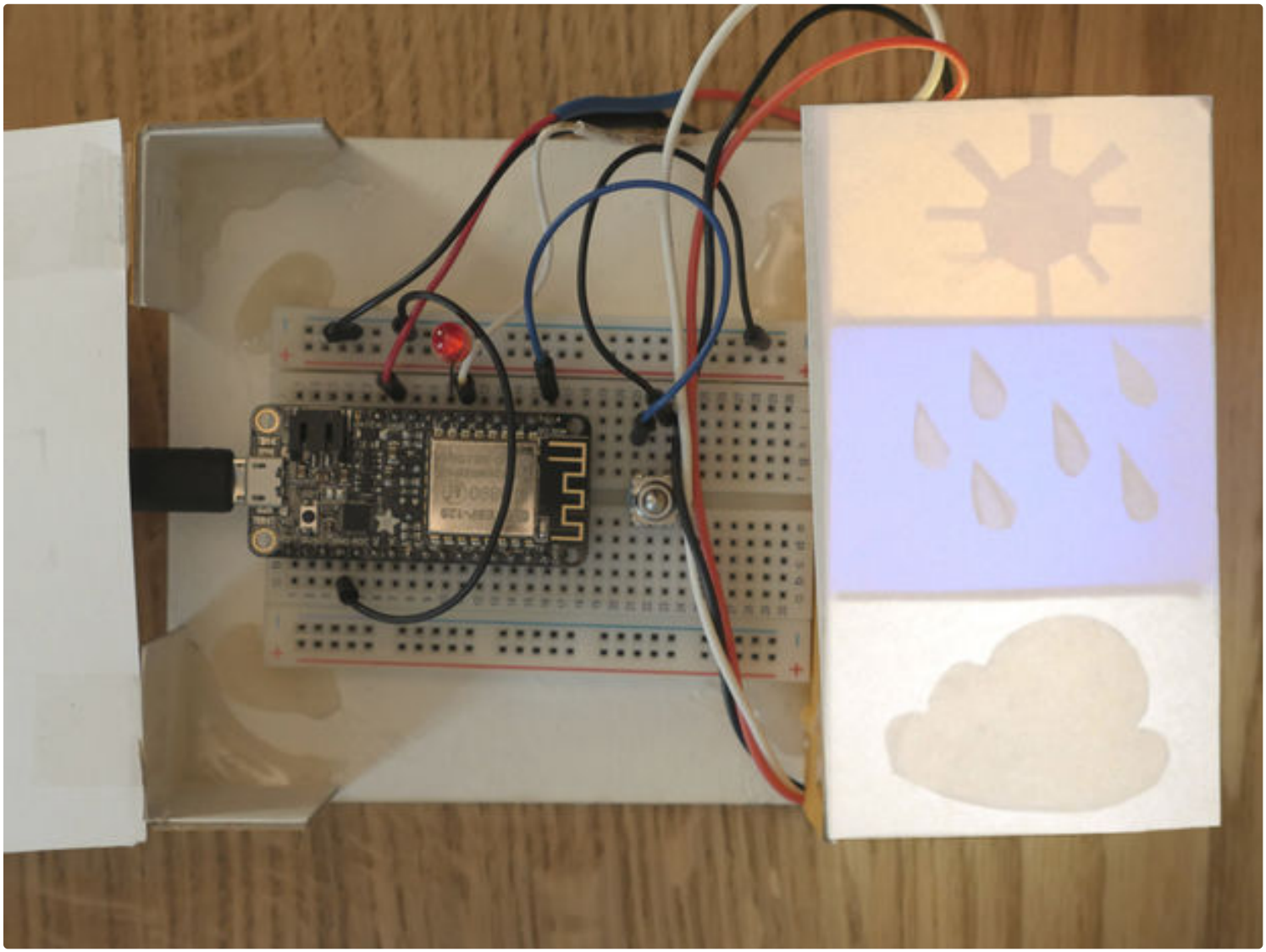


After creating a breadboard prototype, the next logical project step is to make a more permanent, soldered version of the circuit. Consider some of the following factors as you design:

- How will it be powered— USB direct/battery bank, Lipoly battery, other? Know your power requirements (https://www.instructables.com/lesson/Skills-Infusion/#step5).
- Where will it be used— will it be exposed to friction, weather, humidity? Protect your circuit, and use an enclosure or mounting technique that is suited to the function.
- Think safety— don't leave DIY electronics projects alone with your pets or small children.

What would you make with this circuit, code, and web configuration? To complete this lesson, post up an idea or sketch for a project you'd like to make, or a photo of your trigger circuit, in the Class Project module below. I'm looking forward to checking them out and also answering your questions in the Q&A; section at the bottom of the page.

LESSON 5: CIRCUIT DISPLAYS INTERNET DATA

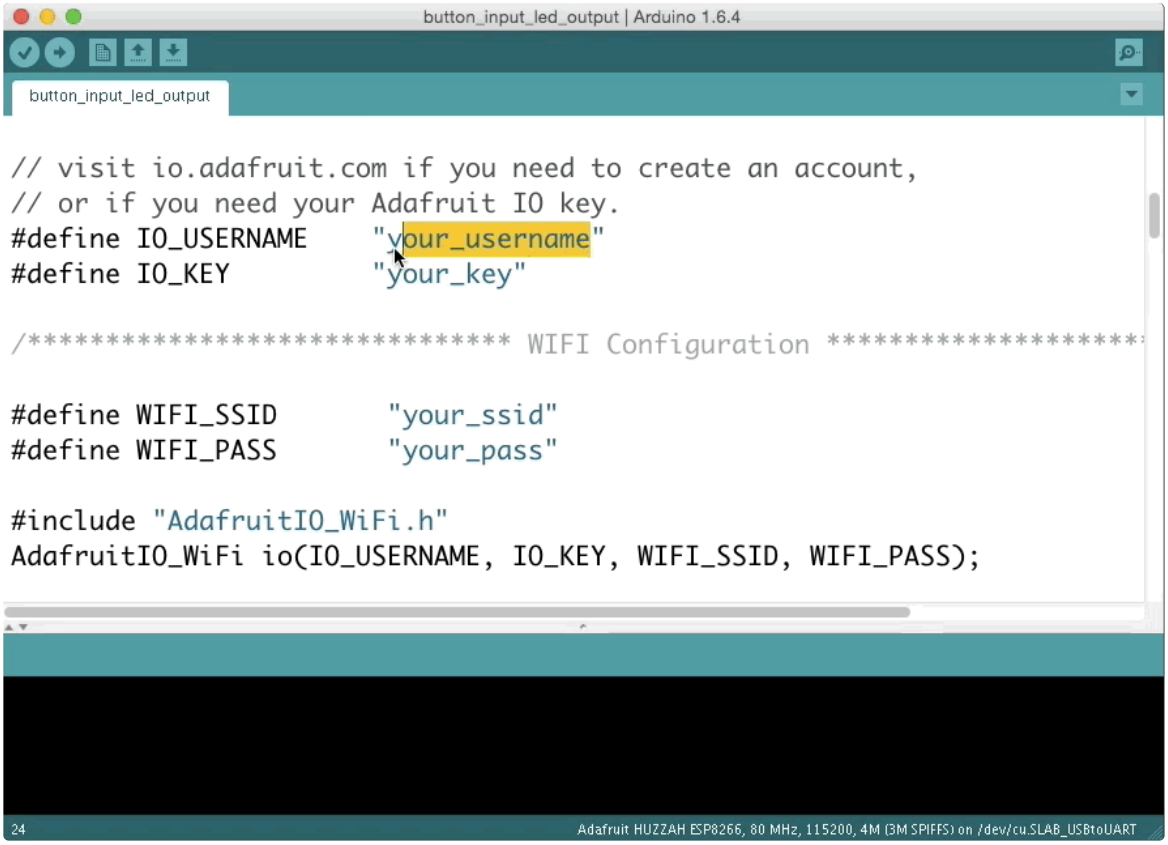


Congratulations on making it this far! You're through the hardest part of getting up to speed with all of the tools, and now it's time for more fun playtime with data display. In this lesson, we'll build a prototype for a real-time LED weather display. You'll learn to code NeoPixels to light up different colors according to the condition updated from IFTTT by way of Adafruit IO.

Simple LED Circuit

Adafruit IO username and key (https://www.instructables.com/lesson/E4H53DMIY4QPXDQ/#step2)

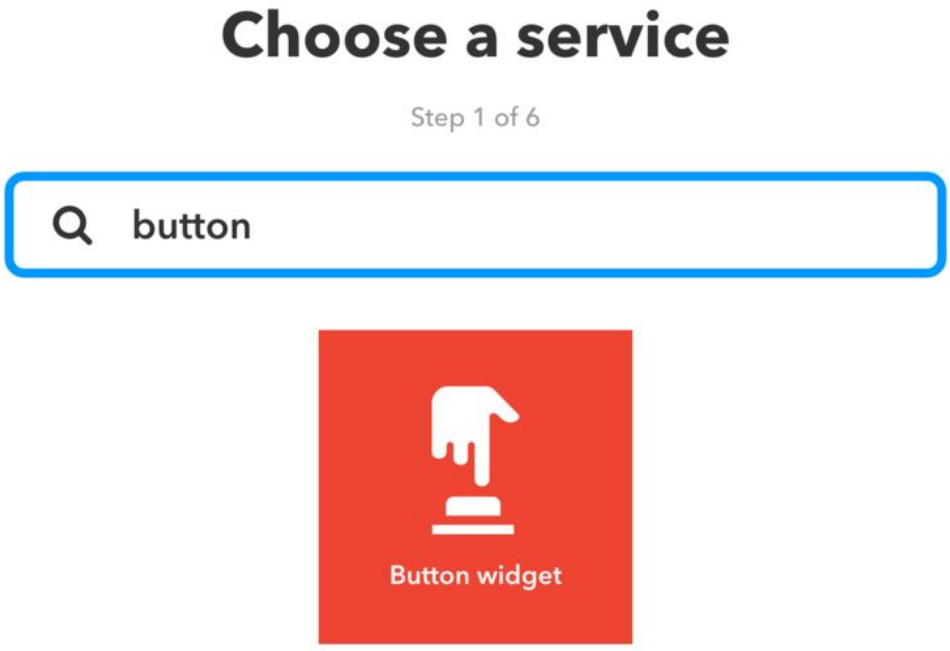
Does this circuit look familiar? It should by now! It's the same one we used at the beginning of the previous lesson— a simple pushbutton and LED connected up to the Feather Huzzah ESP8266. Rebuild the circuit according to the diagram and load up the button-to-Adafruit IO test code once more by navigating to **File -> Sketchbook -> button_input_led_output**. If you can't find it, no worries, just download it (https://www.instructables.com/files/orig/FT1/GG48/J1WW417F/FT1GG48J1WW417F.ino) from this step (and take better care of this copy, will ya?).



Before uploading the code to your board, fill in your Adafruit IO username and key (https://www.instructables.com/lesson/E4H53DMIY4QPXDQ/#step2) as well as your wireless network name and password (if you haven't already). Upload the code and test that the button works to illuminate the LED. Use the serial monitor to check if there are any problems (incorrect wifi details would prevent connection, etc.).

```
int command = data->toInt();

  if (command == 1){ //light up the LED
    Serial.print("received <- ");
    Serial.println(command);
     digitalWrite(LED_PIN, HIGH);
     delay(500);
     digitalWrite(LED_PIN, LOW);
  } else {
    Serial.print("received <- ");
    Serial.println(command);
  }
```

In this code, the LED does not light up in direct response to the button press, but rather in response to a message being received by the Adafruit IO feed "command". Pressing buttons is only one way to get data into a feed— let's use the IFTTT mobile app to light up the LED instead. If you haven't already, you may wish to switch your previous applet off (the one that sends you email) in your IFTTT settings.



After installing the IFTTT app for Android or iOS, you can complete the applet creation process in your computer's web browser or on your mobile device (the process may have some differences in appearance between platforms but is largely the same).

Click to create a new applet, and pick "Button widget" then "Button press" as the trigger ("+this") and "Adafruit" -> "Send data to Adafruit IO" as the output ("+that").

Select your feed "command" from the dropdown menu, and enter the number 1 in the Data field. Click through to finish creating the action.

Applets created with the button widget are special— they put a widget on your mobile device that triggers the action with one click of the icon on your home screen. Add your new widget to your home screen using IFTTT's excellent multi-platform guide (https://ifttt.com/help/applets-with-widgets), and tap away!

What do you notice about the behavior of the circuit? How long does it take for the LED to light up after you activate the widget? When I'm prototyping IoT devices, I will often create a button widget to trigger my circuit manually when it would only otherwise be triggered under certain conditions that complicate prototyping (activities that occur only once per day, only when the weather changes, etc.). It can also be a useful tool in demonstrating a proof of concept prototype where the proposed data source is difficult to simulate live (simulating arriving at a GPS location while inside a presentation hall with wifi but no GPS signal, for example). It's also a great troubleshooting tool.

Add NeoPixels

Let's upgrade this circuit's output with some color changing addressable LEDs, aka NeoPixels. You just need one at the minimum, and the sample code later in this lesson uses six. I don't recommend using more than ten pixels for this exercise. If you haven't already, you can learn to solder wires onto your NeoPixels and install the Arduino code library by reading the Skills Infusion lesson in my free Arduino Class (https://www.instructables.com/lesson/Skills-Infusion/). You might find it helpful to resize your browser window so that this lesson is visible along side your Arduino software window.

Wire up your NeoPixel(s) to your existing circuit according to the following diagram:

We will add to the button sketch from the last step to make the NeoPixel light up whenever the LED does. Although you can download the example sketch (https://cdn.instructables.com/ORIG/FSJ/4Y9Z/J1GP5IFN/FSJ4Y9ZJ1GP5IFN.ino) for this step and open it with your Arduino software, I encourage you to try to make the three small code changes by hand, and use the sample file as a fall back and troubleshooting comparison tool. This will flex your coding muscles and hone your eye for details, both of which are essential skills for developing your own projects down the line. I'll walk you through the code changes one at a time, let's go!

Go to **File-> Save as...** and pick a new name for this modified version of the file. Near the top of the sketch, after the wifi configuration, add in the following NeoPixel setup code, changing the NUM_LEDS variable to match the number of pixels you've got hooked up.

```
#include <Adafruit_NeoPixel.h>

#define PIXELS_PIN 15
#define NUM_LEDS 6
#define BRIGHTNESS 50

Adafruit_NeoPixel strip = Adafruit_NeoPixel(NUM_LEDS, PIXELS_PIN, NEO_GRBW + NEO_KHZ800);
```

The first line includes the NeoPixel library, so your program can access its features, the three variable declaration hold the info about where your pixels are plugged in, how many there are, and how bright they should be, and the last light sets up a new NeoPixel object called strip with the features you just described.

Next, inside the sketch's setup function, add in some lines to activate the NeoPixels:

```
strip.setBrightness(BRIGHTNESS);
strip.begin();
strip.show(); // Initialize all pixels to 'off'
```

While setting the global brightness of your NeoPixels isn't mandatory, it can be helpful for prototyping because it conserves a lot of power. A computer's USB port can only deliver about 500mA, so if I'm experimenting with different code and I've got NeoPixels connected, dialing down the brightness makes the circuit easier to power while changing the code frequently. Oh, and it keeps your project from blinding you while you're working on it, too! `strip.begin();` and `strip.show();` are required to activate the pixel object.

Just below the line of code that illuminates the LED, add two lines that set and show the pixel color change. Customize the color to suit your taste by swapping out the R, G, B, and W values in `strip.color();`

```
//change NeoPixel color here using format strip.Color(R,G,B,W)
strip.setPixelColor(0, strip.Color(0,0,0,100));
strip.show(); //always remember to call strip.show() to display changes
```

Don't forget to add another `strip.setPixelColor();` and `strip.show();` to turn the pixel off when the LED goes LOW (lines 144 and 145 in the screenshot above).

```
strip.setPixelColor(0, strip.Color(0,0,0,0)); //turn off NeoPixel
strip.show(); //always remember to call strip.show() to display changes
```

After uploading the updated code, your first pixel should flash along with the LED you connected earlier. At this point you may decide to remove the LED entirely from your breadboard and code, though I like to leave it in place while prototyping/troubleshooting. If you try to compile your code but get errors, comb over your code looking for typos (extra commas, missing semicolons, etc.), and compare to the downloadable sample. If your code uploads but does not light up, check your wifi and AIO details for errors and use the serial monitor to check if your circuit is connecting and receiving data.

To control more pixels, just set their colors with strip.setPixelColor(), each on its own line (and don't forget to double check that the NUM_LEDS variable at the top of the sketch matches your pixel count). Remember to call strip.show(); to display your changes, and turn them all off too.

```
strip.setPixelColor(0, strip.Color(0,0,0,0)); //turn off NeoPixel
strip.setPixelColor(1, strip.Color(0,0,0,0)); //turn off NeoPixel
strip.setPixelColor(2, strip.Color(0,0,0,0)); //turn off NeoPixel
strip.setPixelColor(3, strip.Color(0,0,0,0)); //turn off NeoPixel
strip.setPixelColor(4, strip.Color(0,0,0,0)); //turn off NeoPixel
strip.setPixelColor(5, strip.Color(0,0,0,0)); //turn off NeoPixel
strip.show(); //always remember to call strip.show() to display changes
```

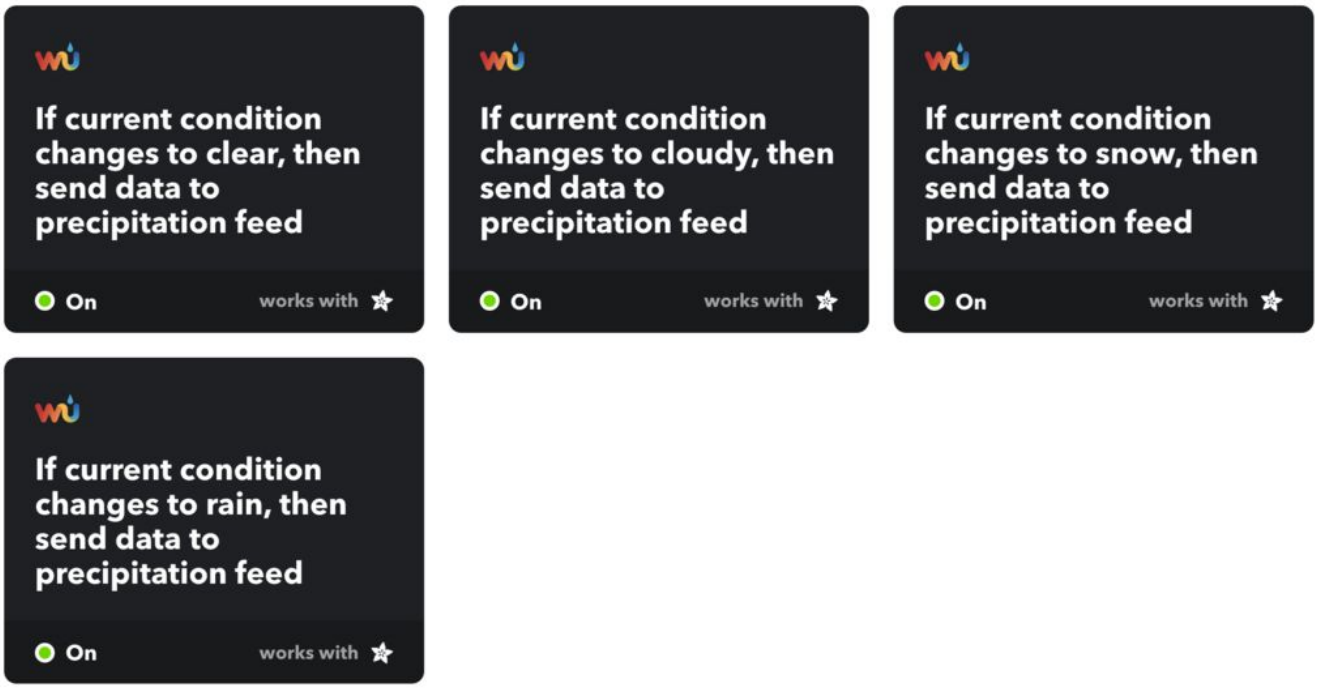
Any time your code has similar repeating element, it's a sign that you can probably make that code more concise. For example, to turn off all the pixels, you could "zero out" the color of each one, on it's own line of code (six pixels shown above), or replace that list with a small for loop:

```
for(int i=0; i<strip.numPixels(); i++) { //turn off all NeoPixels
  strip.setPixelColor(i, strip.Color(0,0,0,0));
}
strip.show(); //always remember to call strip.show() to display changes
```

This kind of small code improvement, or **optimization**, can make your code more flexible and easy to read, and even make the resulting file size smaller. While you may not feel confident enough in your coding skills yet to know when to make these changes, it's important to realize that there are many ways to write code that performs the same actions. In this case either way is fine, but there is one key difference/tradeoff to note: the for loop iterates to `strip.numPixels();`, which is a built-in function that will return with the number of pixels established at the start. So if you do change the number of pixels later, you'd have less work to do to update the whole sketch to suit.

Next, let's explore another type of data to represent with our pixels.

Collect Weather Conditions in Feed



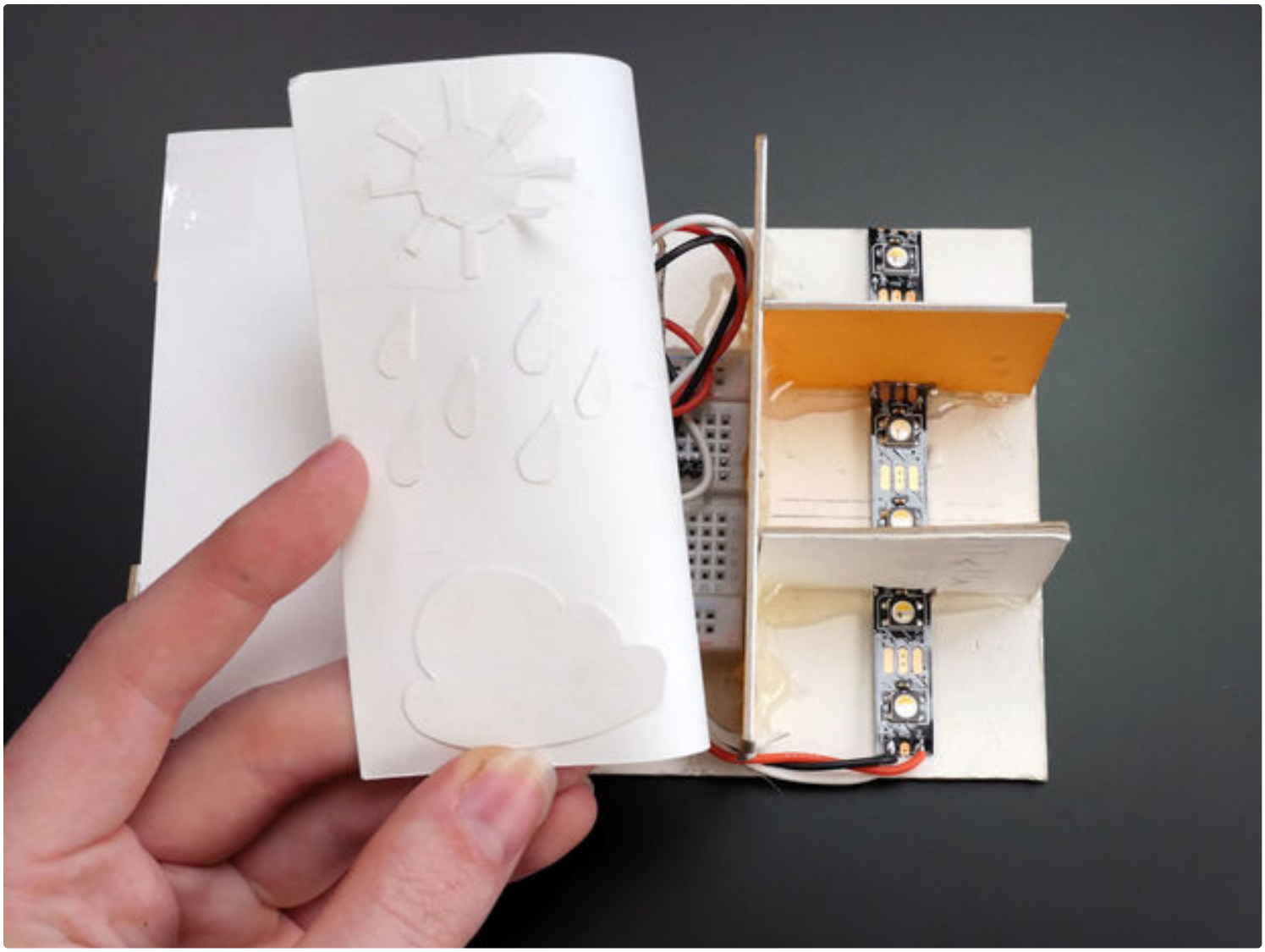
Up until now, our feed's data has consisted only of 0s and 1s. Let's see what happens when we collect some weather data into a new feed called "precipitation". To get an update each time the current weather condition changes, create four IFTTT applets that send data to the same precipitation feed using the Weather Underground trigger, each with a different one of the four condition options. Since this feed is only updated when the weather changes, it can take a few days to see the variation in the data.

Instead of numbers, the feed is filled with words representing a variety of weather conditions. In Arduino, this feed data type is called a `String`, which is an object that can store multi-character text. You can learn more about this variable type on the official Arduino site (https://www.arduino.cc/en/Reference/StringObject), and explore several String examples in the Arduino software. Since we are looking to evaluate the strings in the feed, the StringComparisonOperators example (https://www.arduino.cc/en/Tutorial/StringComparisonOperators) is particularly relevant. In the sample

sketch (also accessible in your Arduino software by navigating to **File-> Examples->08.Strings->StringComparisonOperators**), you can find several if statements with descriptive comments that explain and show how to compare two strings in several different ways.

```
// two strings not equal (case sensitivity matters):
  stringOne = "This";
  stringTwo = "this";
  if (stringOne != stringTwo) {
    Serial.println(stringOne + " =! " + stringTwo);
  }
  // you can also use equals() to see if two strings are the same:
  if (stringOne.equals(stringTwo)) {
    Serial.println(stringOne + " equals " + stringTwo);
  } else {
    Serial.println(stringOne + " does not equal " + stringTwo);
  }

  // or perhaps you want to ignore case:
  if (stringOne.equalsIgnoreCase(stringTwo)) {
    Serial.println(stringOne + " equals (ignoring case) " + stringTwo);
  } else {
    Serial.println(stringOne + " does not equal (ignoring case) " + stringTwo);
  }
```

From this section of the example, you can see that comparing strings can be just like comparing numbers by using an if statement and comparison operators == (is equal to) and != (is not equal to), as well as two built-in functions `.equals` and `.equalsIgnoreCase`. All three function very similarly, but we'll use `.equalsIgnoreCase` to provide the maximum input flexibility.



Display Weather Data With NeoPixels

Let's start with a new downloadable sample sketch (https://www.instructables.com/files/orig/FZE/83II/J1MEVOME/FZE83IIJ1MEVOME.ino), which does not contain button input, and has been loaded up with the necessary weather handling bits. Open it up in Arduino, personalize it to reflect your wifi and AIO info, and upload the code to your Feather Huzzah NeoPixel circuit. You may leave the pushbutton and LED connected, although they won't be used again until the next lesson.

To mock up a weather display, let's make the pixels illuminate some symbols. One of the simplest and best ways to diffuse LEDs is to give them space, so build a small cardboard enclosure that has some depth to it. Add more cardboard walls to divide the pixels into chambers (I'm using three chambers of two pixels each). A piece of plain white printer paper taped to the openings of the chambers provides the "screen," and glueing on another layer of cut paper creates shadowy outlines of whatever shapes you choose.

To test this code, you can manually add data to your feed by clicking on Actions -> Add Data. Type in a weather condition (stick to the ones described in the Arduino sketch) and click Create to add it to the feed. If all's well, your pixels should change whenever the feed updates.

Let's take a look at how the code works so you can customize it for your own projects.

```
AdafruitIO_Feed *precipitation = io.feed("precipitation"); // set up the 'precipitation' feed
```

Before the startup function, you'll need to set aside some memory for the feed you'll be using. Though it doesn't have to have the same local name as you call it on Adafruit IO, it sure is helpful in keeping track of everything!

Inside the setup:

```
// set up a message handler for the 'precipitation' feed.
// the handleMessage function (defined below)
// will be called whenever a message is
// received from adafruit io.
precipitation->onMessage(handleMessage);
```

Pretty much what the comment says— this line calls the function handleMessage whenever the precipitation feed is updated. Scroll down to below the main loop to the handleMessage function, line 101:

```
String forecast = data->toString(); // store the incoming weather data in a string
```

This line puts the feed data into a string called "forecast".

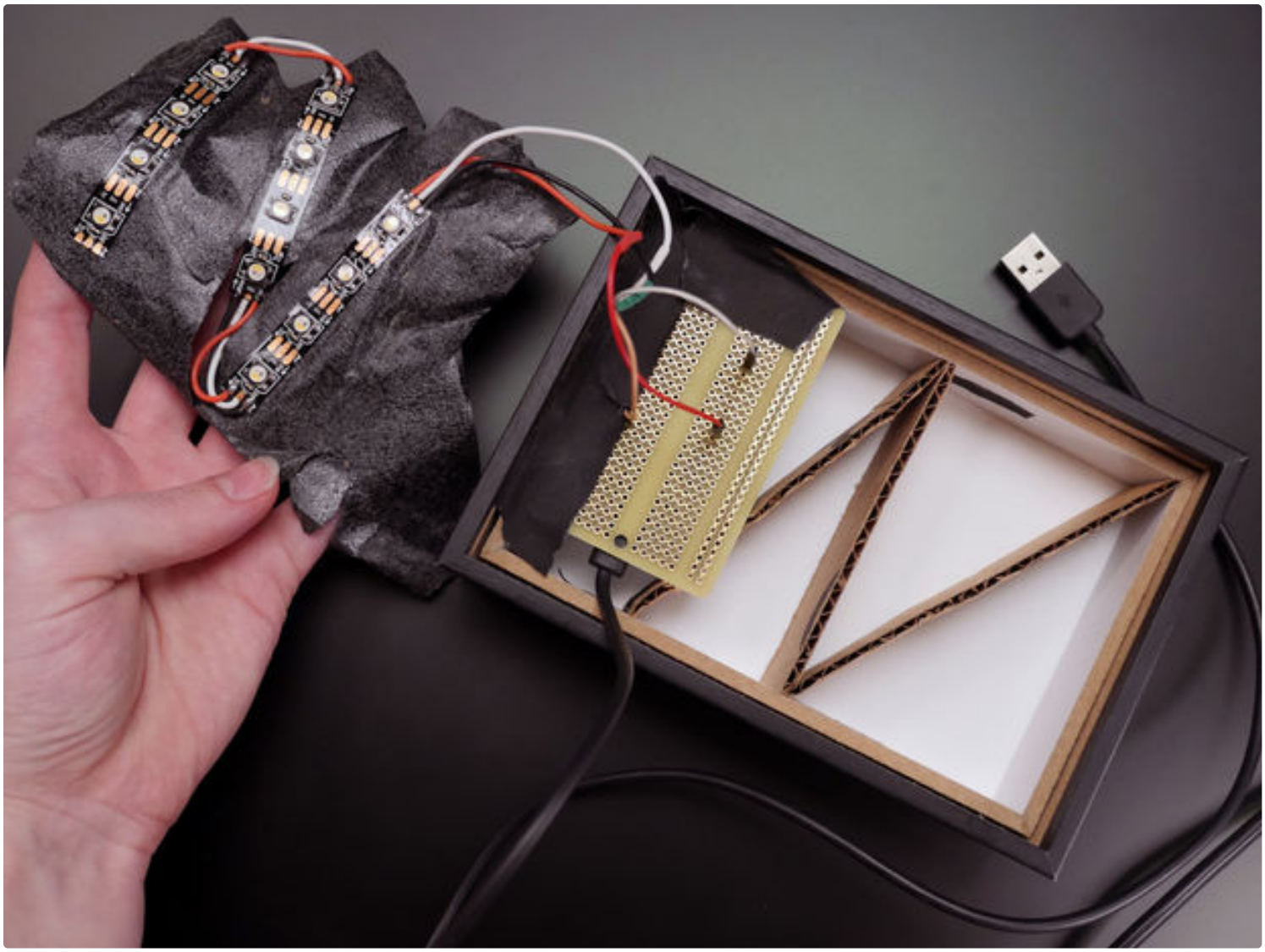
```
//the following strings store the varous IFTTT weather report words I've discovered so far
String rain = String("Rain");
String lightrain = String("Light Rain");
String rainshower = String ("Rain Shower");
String AMshowers = String ("AM Showers");

String rainandsnow = String("Rain and Snow");
String snow = String("Snow");
String snowshower = String("Snow Shower");

String cloudy = String("Cloudy");
String mostlycloudy = String("Mostly Cloudy");
String partlycloudy = String("Partly Cloudy");

String clearsky = String("Clear");
String fair = String("Fair");
String sunny = String("Sunny");
```

These variable declarations describe all the weather conditions I've observed in the feed, grouped loosely by condition. These declarations could be moved to the start of the program, but keeping them here makes it easier to understand the next part with all the if statements:

```
// if there's rain in the forecast
  if (forecast.equalsIgnoreCase(rain) || forecast.equalsIgnoreCase(lightrain) || forecast.equalsIgnoreCase(rainshower) || forecast.equalsIgnoreCase(AMshowers)){
    Serial.println("precipitation in the forecast today");
    strip.setPixelColor(0, strip.Color(0, 0, 0, 255));
    strip.setPixelColor(1, strip.Color(0, 0, 0, 255));
    strip.setPixelColor(2, strip.Color(0, 30, 150, 150));
    strip.setPixelColor(3, strip.Color(0, 30, 150, 150));
  }
```

This statement compares the feed data stored in the forecast string against the rainy terms we defined moments ago. Using the **or** `||` comparison operator lets us add multiple conditions to the same if statement, grouping several rainy conditions into the same blue and white NeoPixel output.

Three more similar if statements evaluate for the three other groups: snow, cloudy, and clear sky. Don't forget to call `strip.show();` to display the color changes!



Taking It Further

If you find the weather display ideas intriguing and/or useful, you may wish to build a more finalized version with a soldered circuit inside an enclosure of some kind.

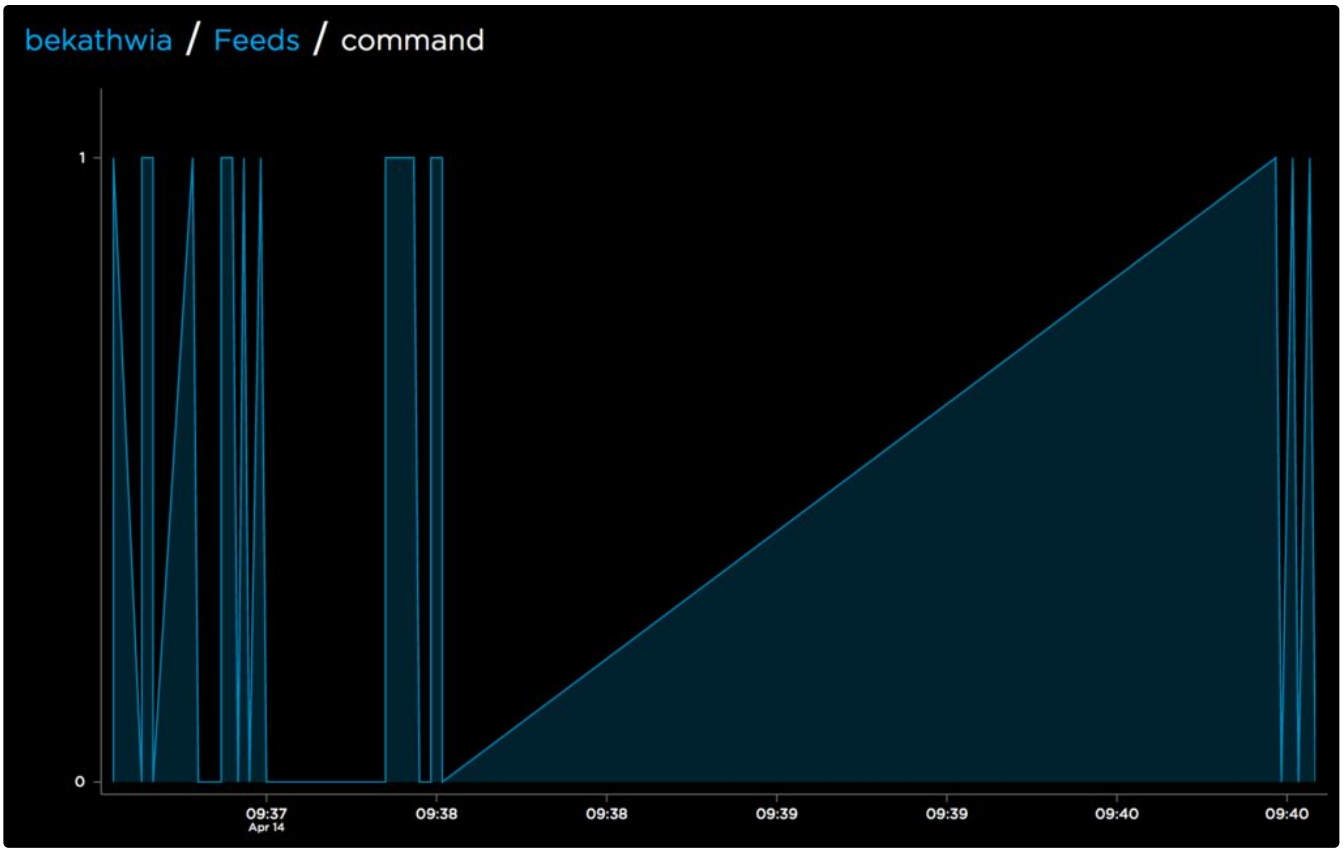
You may wish to add a numeric LED display to show the temperature too— find techniques for both in my WiFi Weather Display Instructable (https://www.instructables.com/id/WiFi-Weather-Display-With-ESP8266/)!

Perhaps you're interested in visualizing some numerical data like number of YouTube subscribers— I've got you covered there too (https://www.instructables.com/id/YouTube-Subscriber-Counter-With-ESP8266/). These are just two examples of projects that display internet data, check out more project ideas in the collection below! To complete this lesson, post up a picture of your circuit from this lesson, or a sketch and description of a project you'd build with the skills you learned here. Next up we'll combine inputs and outputs to create two devices that can talk to one another from anywhere there's a wifi connection.

LESSON 6: COMBINING INPUTS AND OUTPUTS

Let's go through one more example to cement your new knowledge. In some cases, you may find your project needs to both trigger internet action and displays internet data. This lesson combines techniques we've learned in earlier lessons to make a pair of devices that communicate with one another. This is also a great example to base your project on if you want to trigger an internet action and display some feedback about that action, whether on the same device or another. We'll use the Internet Valentine (https://www.instructables.com/id/Internet-Valentine/) project as a case study throughout the lesson, which flashes the LED and buzzes the pager motor on both devices when buttons on either one are pressed.

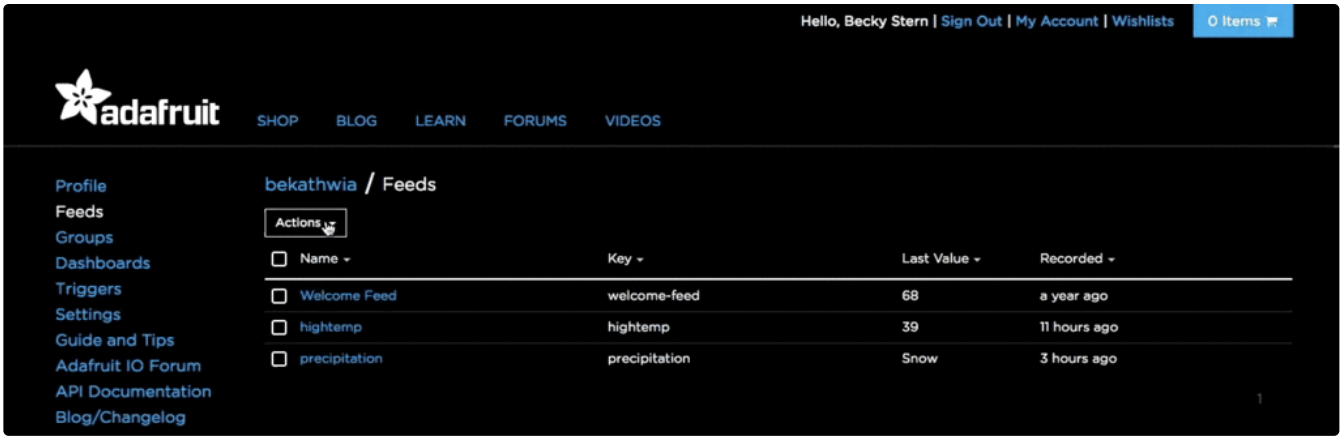
Establish Feed(s) First

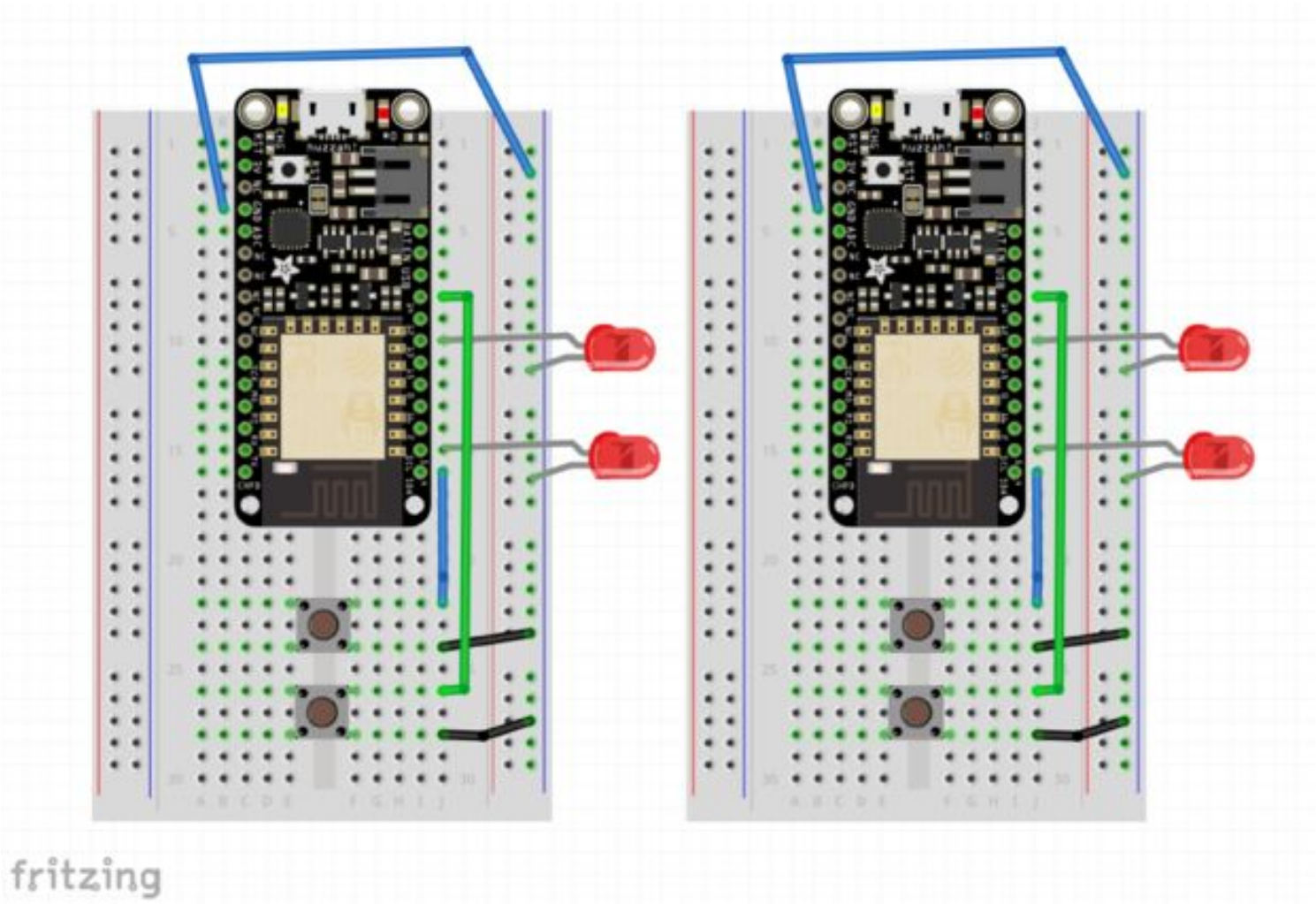


The project design workflow for any IoT project starts with the data. Both the Arduino sketch and IFTTT applets look to the feed on Adafruit IO for interaction, so let's define that first. We can break it down as follows:

- three input states: off, button 1 pressed, and button 2 pressed
- correspond to three output states: off, LED flashes, and motor wiggles

Let's reuse the "command" feed from the Circuit Triggers Internet Action lesson, or create a new feed with a name describing what your data represents.

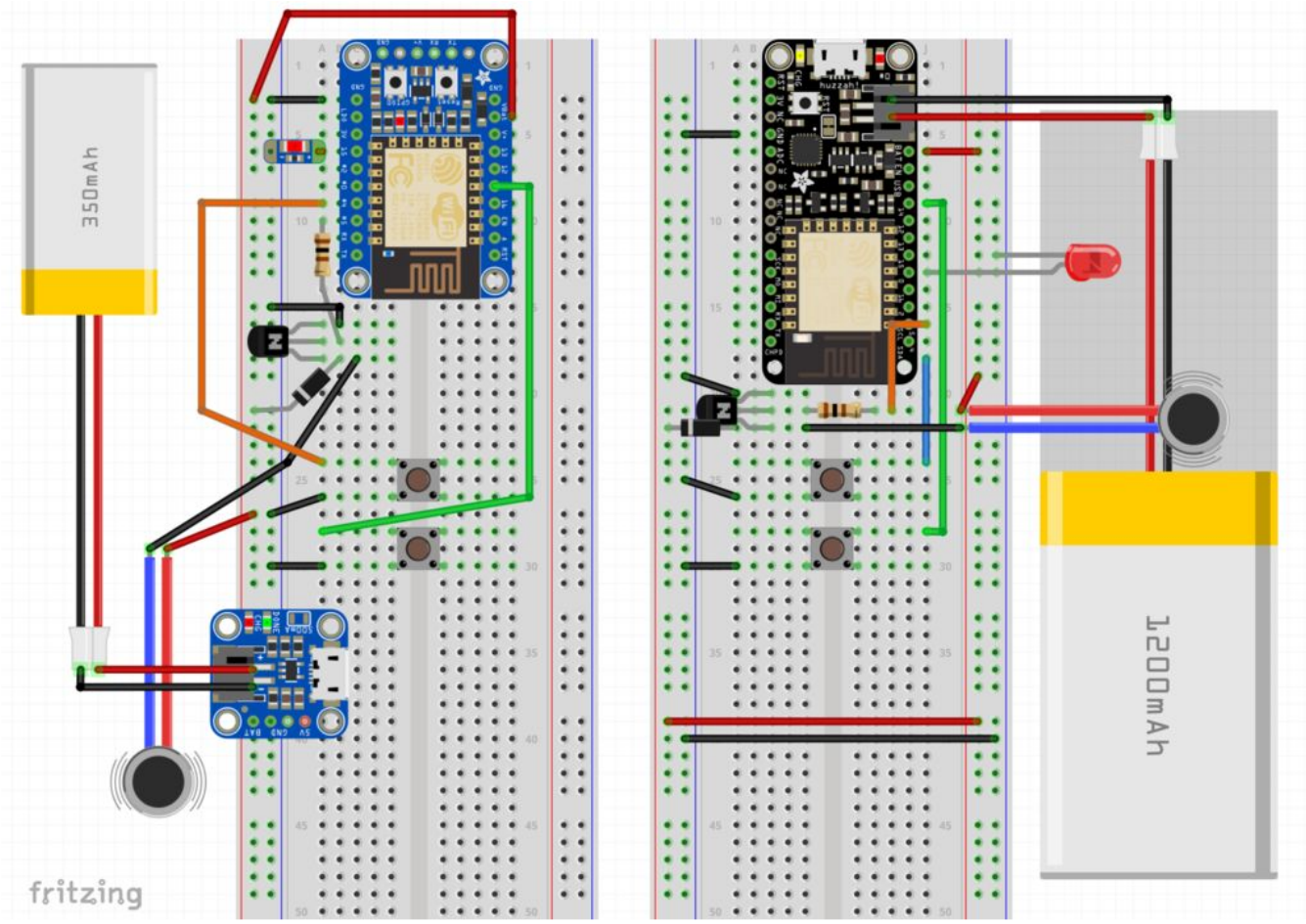




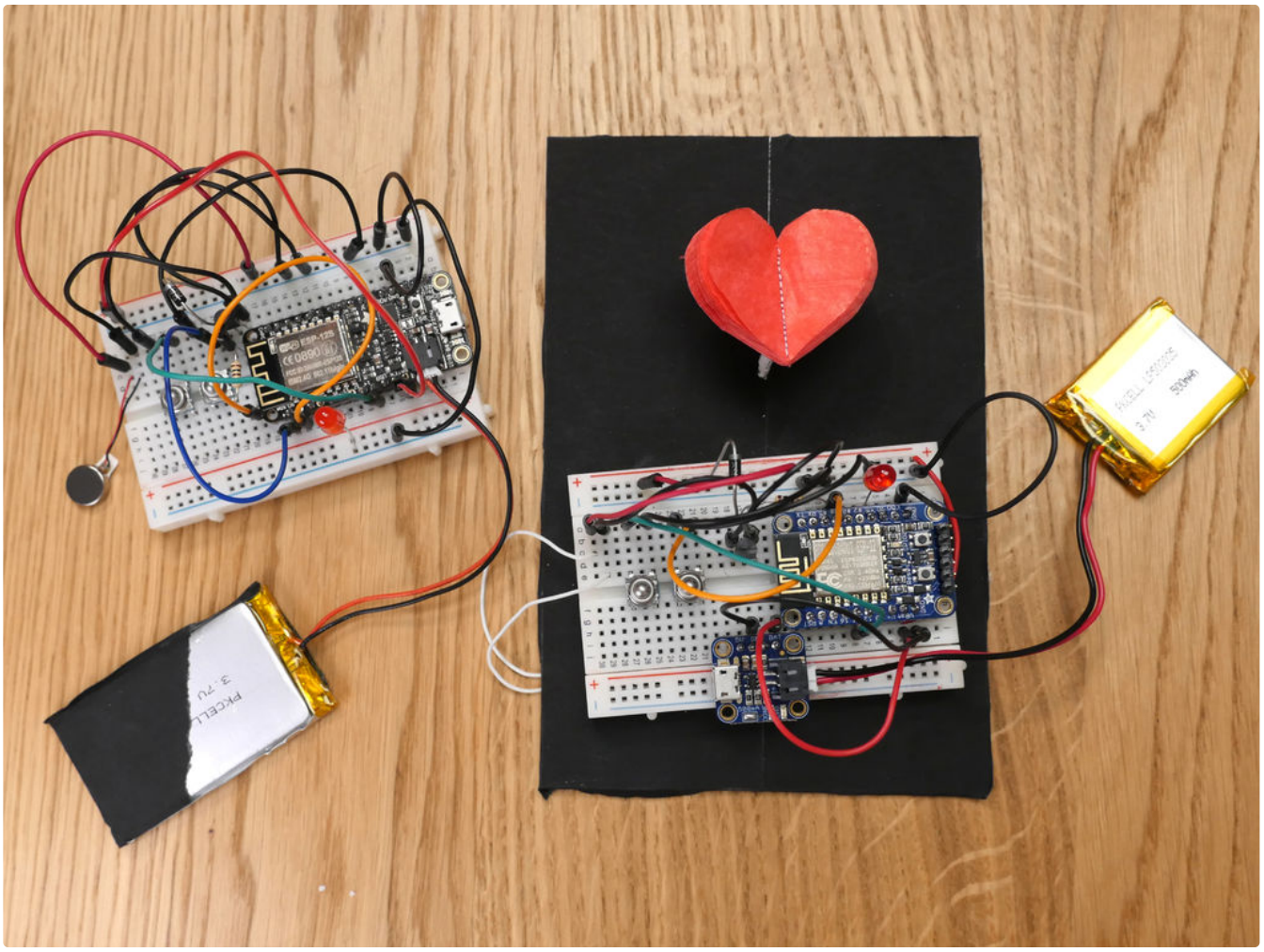
Create Two Identical Circuits

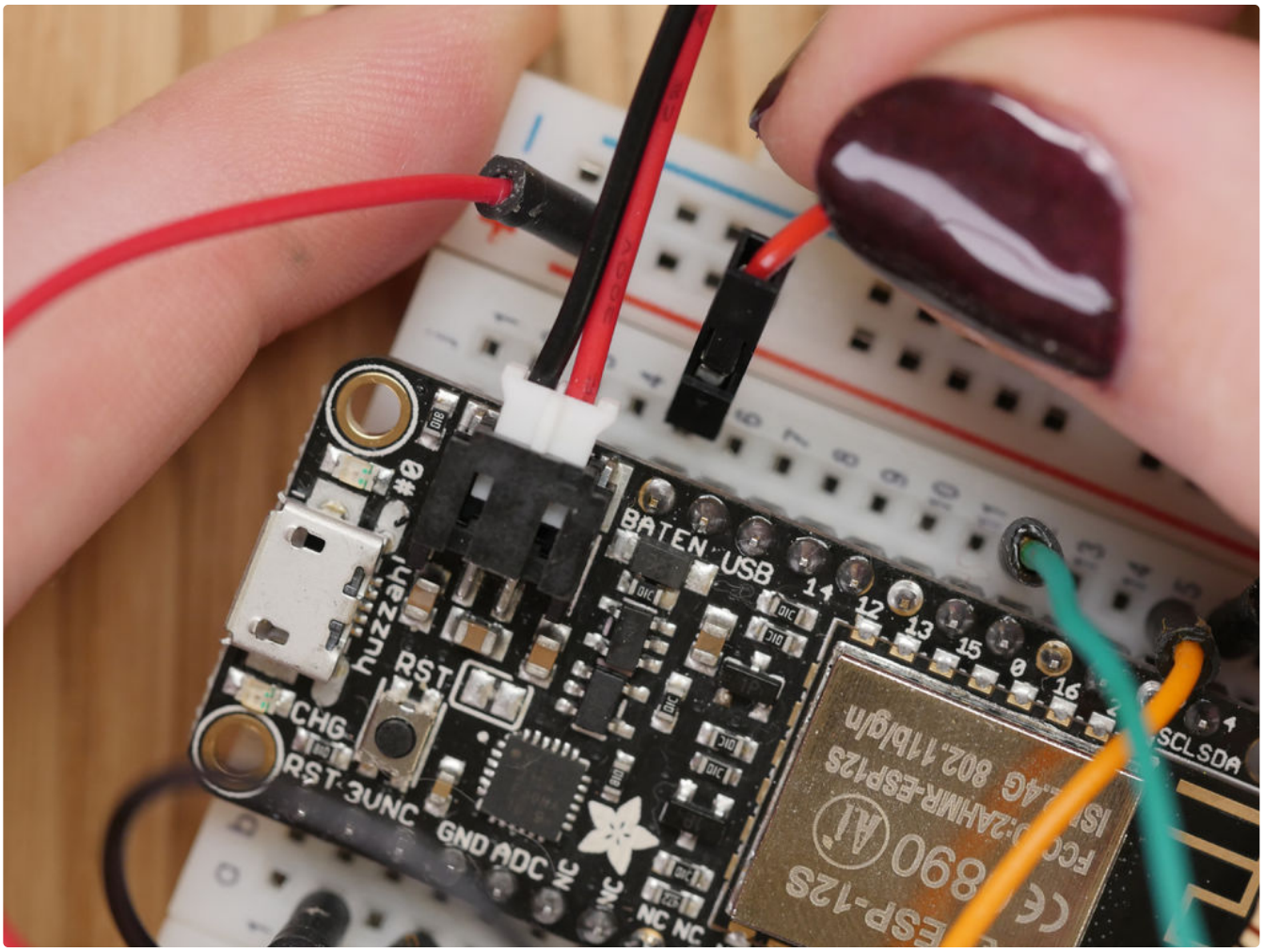
Build the same starter circuit as we've been using all along, but this time add a second pushbutton and LED.

If you've got an extra set of parts, build a second identical circuit. Or shake things up with the inputs and outputs of your choice— shown here is just an example of two digital inputs controlling two outputs.



If you're feeling fancy, add a transistor/vibrating motor instead of a second LED.





Power the boards via USB or with LiPoly batteries (adjust position of power wire between USB and BAT accordingly, if you are using it for something like NeoPixels, a motor, etc.).

Input/Output Code

Download the example sketch (https://cdn.instructables.com/ORIG/F16/MLGF/J1MEW0DU/F16MLGFJ1MEW0DU.ino) and open it with the Arduino IDE. Customize the Adafruit IO username and password as well as your wifi network name (SSID) and password. Be sure you have the "ArduinoHttpClient", "Adafruit IO Arduino", and "Adafruit MQTT" libraries installed, and upload the code to your board.

This section looks very similar to the first button code you used with Adafruit IO, but with a second button and if statement. Now the feed data will contain three different values depending on the state of the buttons: 0, 1, and 2.

Later on, the handleMessage function checks for each of these three states and either buzzes the motor with analogWrite();, flashes the LED, or does nothing but print to the Serial Monitor.

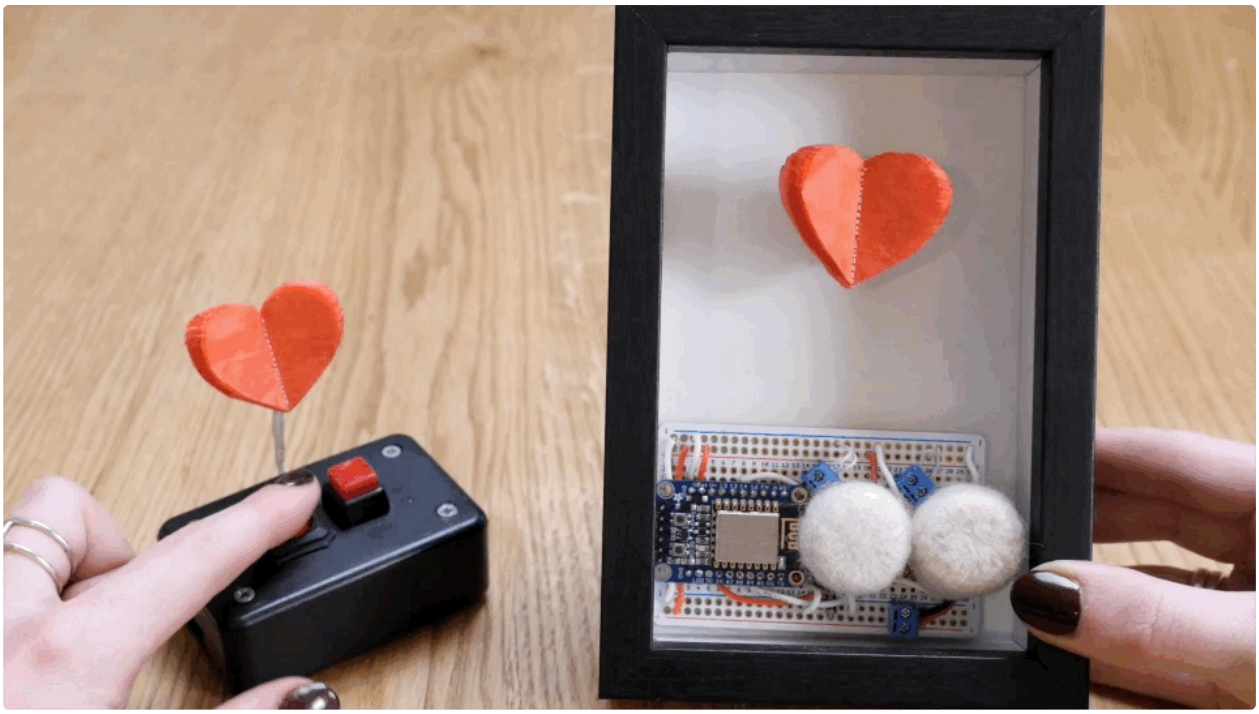
Put the same code on both ESP8266 circuits and observe the way they behave. For example, how long do the messages take to echo back from Adafruit IO, and how much does that time vary? You can move one device to a friend's house (don't forget to update the wifi network details in the Arduino code) and experiment some more.

Download (https://cdn.instructables.com/ORIG/F16/MLGF/J1MEW0DU/F16MLGFJ1MEW0DU.ino)

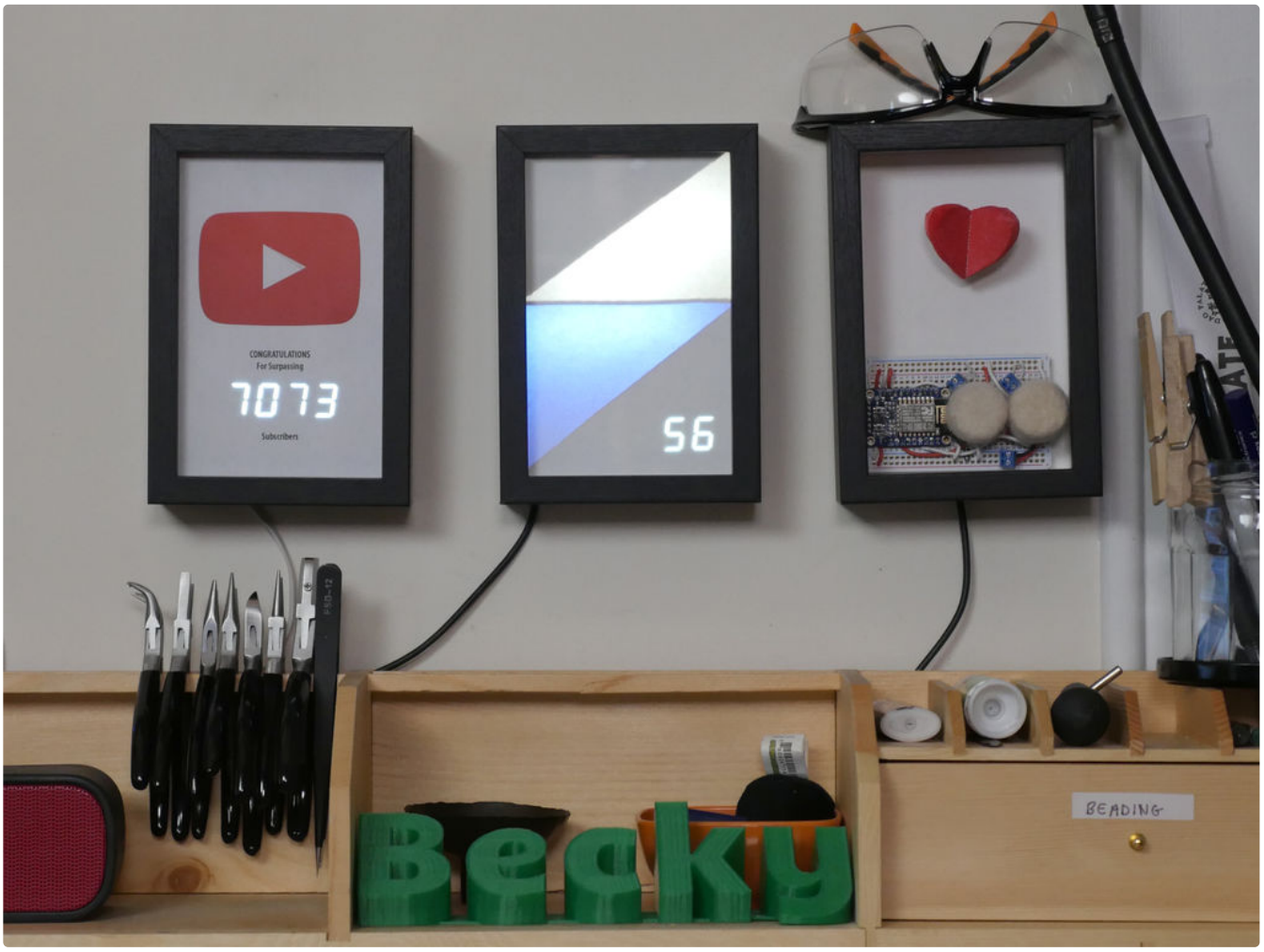
http://www.instructable...

(https://cdn.instructables.com/ORIG/F16/MLGF/J1MEW0DU/F16MLGFJ1MEW0DU.ino)

Taking It Further



Now that you've come this far, you should feel confident in adapting the examples in this class to your own inputs and outputs. You could add another feed and modify each device's code to only control one another (no local feedback), or hook up an IFTTT applet that sends different messages based on which device sent the trigger.



You may find that one ESP8266 project is not enough to satisfy your new curiosity. Check out the projects in the collection below for more inspiration, and don't forget to explore more examples in the Arduino software under **File -> Examples -> ESP8266WiFi** and **File -> Examples -> Adafruit IO Arduino**. Additionally, check out the following sites for new DIY IoT projects posted frequently:

- Instructables with ESP8266 (https://www.instructables.com/howto/esp8266/)
- ESP8266 on Twitter (https://twitter.com/esp8266)
- (https://twitter.com/esp8266)Hackaday blog (http://hackaday.com/blog/)
- Adafruit blog (https://blog.adafruit.com/)

Thank you for coming on this class journey with me! I welcome your feedback and look forward to seeing your own projects. Please post up Instructables documenting your designs (and enter them in our frequent contests). If you liked this class, here are some more you may enjoy!

- Raspberry Pi (https://www.instructables.com/class/Raspberry-Pi-Class/)
- LEDs & Lighting (https://www.instructables.com/class/LEDs-and-Lighting-Class/)
- Electronics (https://www.instructables.com/class/Electronics-Class/)

- Robots (https://cdn.instructables.com/F25/DJR0/IO0G9BRH/F25DJR0IO0G9BRH.RECT600.jpg)
- Wearable Electronics (https://www.instructables.com/class/Wearable-Electronics-Class/)