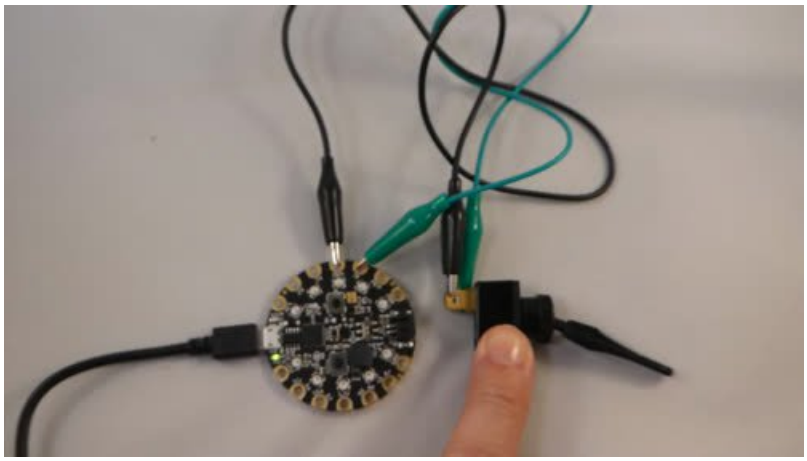
adafruit learning system

Make It Switch

Created by Mike Barela



Last updated on 2018-08-22 04:09:44 PM UTC

Guide Contents

Overview

(Youtube Video: Star Trek Type 4B Computer Replica (https://adafru.it/C85), by Jeff Kaler, displaying various switches)

For a century and a half, it is the mechanical switch that has allowed interaction between us humans and the machines we build. On/off, the switch is ultimately what we modelled to build computers which contain millions of tiny silicon switches to calculate in binary.

The old-fashioned switches are still very important today in nearly any project.

You'd think the use of an external switch for triggering actions with your project is straightforward enough. You connect a push button or toggle switch to a microcontroller and . . . and perhaps nothing.

Using switches is great - we encourage you to do so! Here are some tips for getting your project working effectively so you can work on other things besides a cranky switch.

Types of Switches

There are many types of switches. Here are the basics and how they boil down to the simple switch in most ways:

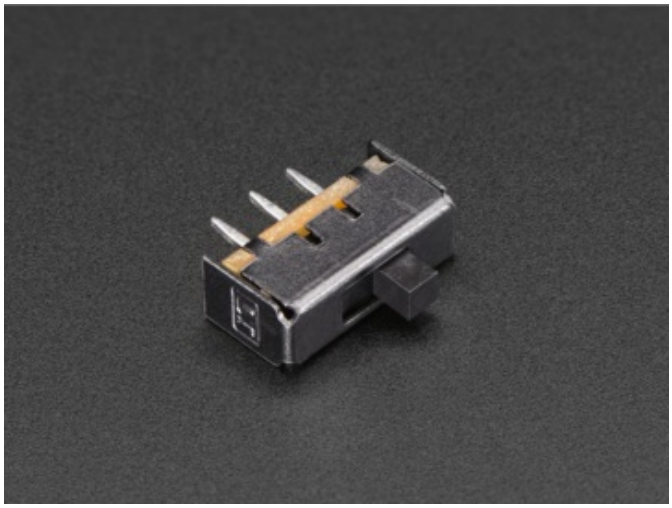
Switch Packaging

Below are some switch packaging that may not be familiar at first, but electrically they are the same as the simple switches discussed above.

Arcade Pushbutton Switches

Adafruit carries a wide selection (https://adafru.it/C8a) of this type of pushbutton switch. They come in various sizes and colors.

There are also arcade switches with an LED inside to indicate switch status - this is usually a second set of terminal tabs on the bottom of the switch and you have to hook up the appropriate leads to power the LED when you want it to glow.



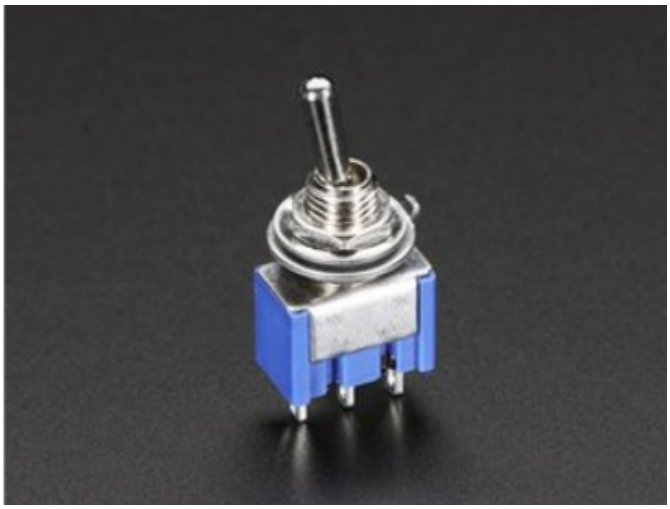
Slide Switch

Slide switches (https://adafru.it/drN) are activated by moving the button to the left or right. The one to the left is a single pole, double throw and is made to fit the pin spacing in breadboards.

Other switches on the market, especially older ones may have multiple positions of multiple poles. If unsure, use a multimeter to get a feel for which contacts connect when in a certain position.

Tactile Switches

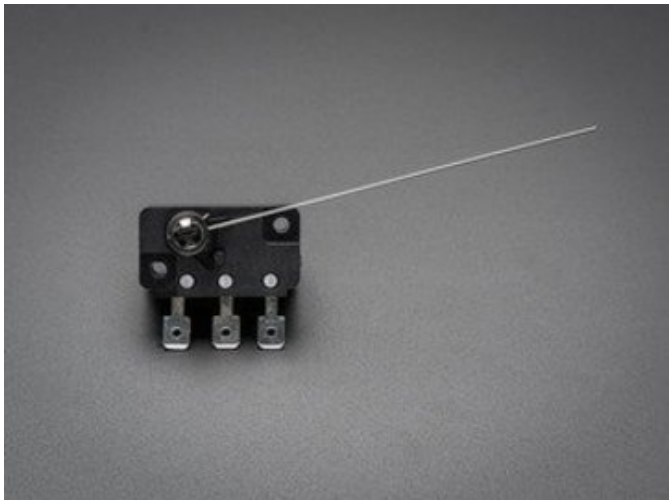
Tactile switches (https://adafru.it/C8b) are most often used on circuit boards and breadboards. They are inexpensive and easy to use (as long as the contacts are connected correctly, see the Hook Up page for help on that).



Toggle Switches

Toggle switches (https://adafru.it/C8c) are good for a nice, tactile feel and a visual if the switch is in one position or another. They can come in Single Pole Single Throw, Single Pole Double Throw, and Double Pole Double Throw configurations.

Some toggle switches have a guard on top to prevent them from accidentally being activated by a bump of a panel. Others may have extra leads for a lighted connection.

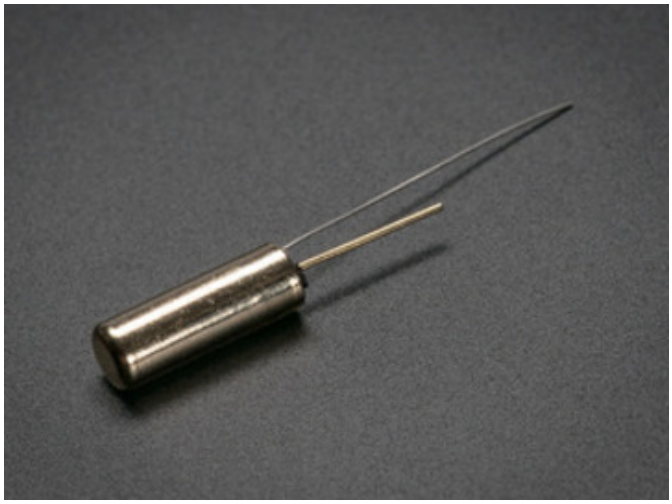


Microswitch

A micro switch (https://adafru.it/t8E) is another name for a small switch that is usually triggered by some movement onto it's switching mechanism. The switch may have a lever to help it trigger.

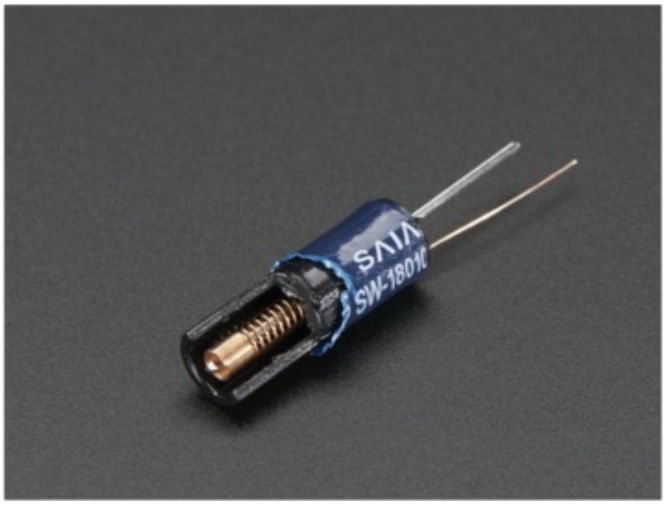
Foot Switch

A foot switch (https://adafru.it/emk) is just another switch, but it is triggered by one's foot. Often single pole single throw or single pole double throw.



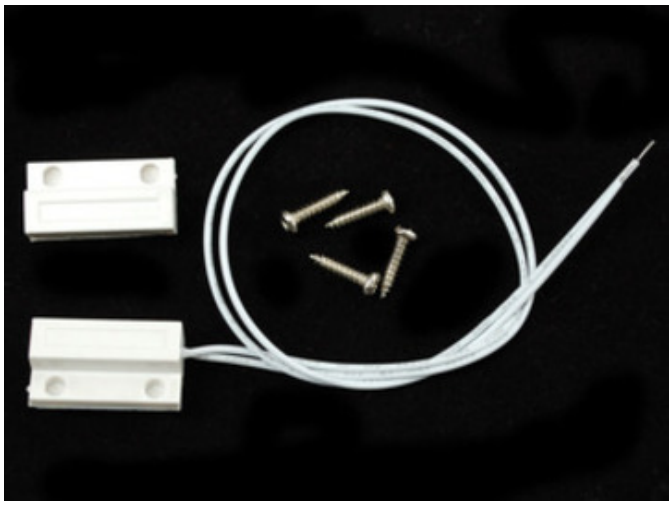
Tilt Sensors

Adafruit sells several tilt sensors (https://adafru.it/C8d). They are just a switch that activates not by push but by tilt. You can read them the same way as the switches discussed earlier.



Vibration Switches

Vibration Switches (https://adafru.it/uF4) look similar to tilt sensors. The one to the left has been opened to reveal the switch will activate when bounced around, not tipped, due to the spring.



Magnetic Contact Switch

At left is a single pole single throw simple magnetic contact switch (https://adafru.it/uF3) that activates when a magnet gets near it.

Mag switches can come in other configurations such as single pole, double throw, and double pole double throw from other manufacturers.

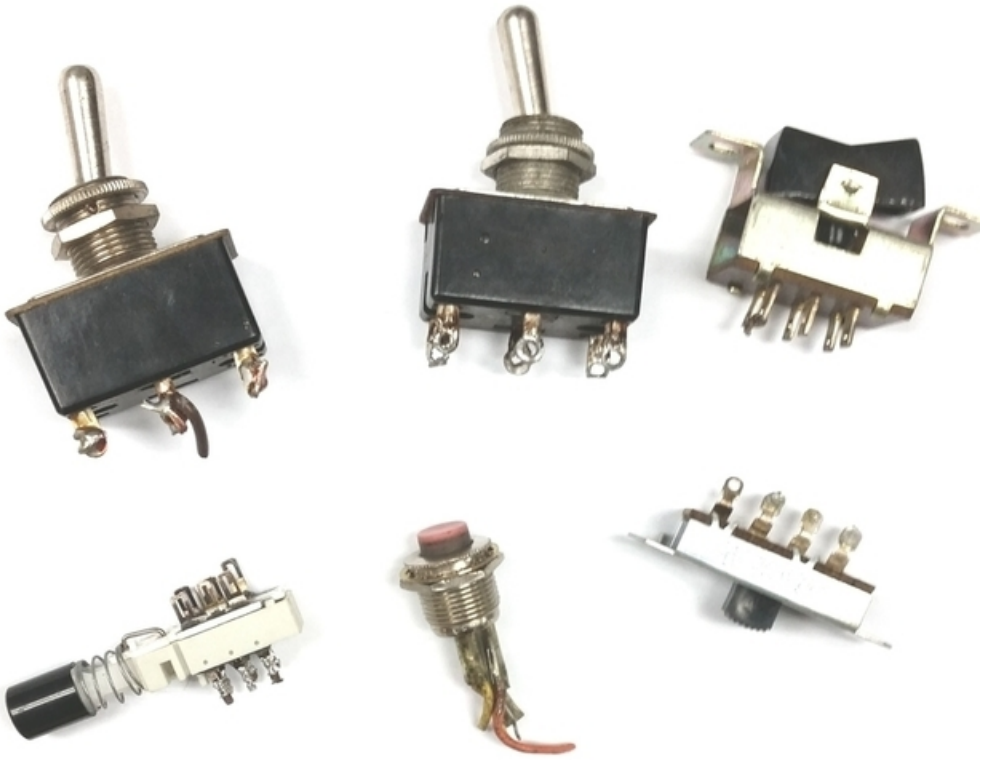
There are many, many more types of switches you may encounter. If you just remember the basics, it's very much the same: use a multimeter, check the terminals before and after switching, jot things down on paper to remember.

Use a manufacturer's diagram if one is available.

For switches with built-in lights, you'll find contacts that do not change when switched. Those most likely are the lighting terminals. If you can, try to find the manufacturer's details on the switch so you know what is the switch and what is the lighting contacts. The information on the lighting will tell you polarity (if any) and if there are any other components inside like an LED current limiting resistor. This will help you design your project's circuitry.

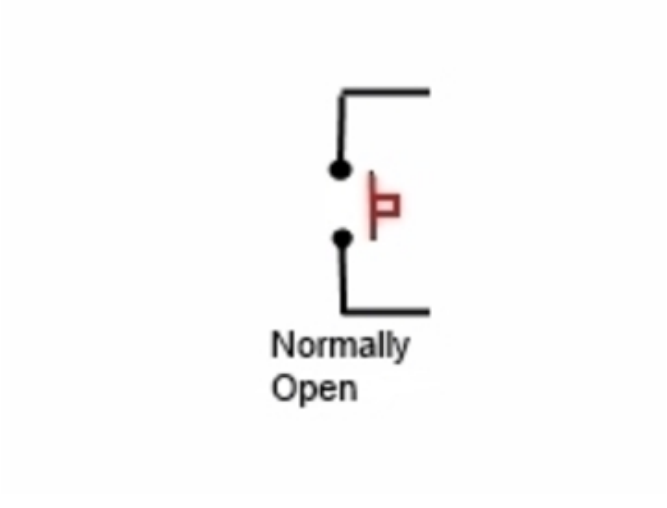
Multi-pole Switches

Some switches have multiple, independent switches that are activated by the same action.



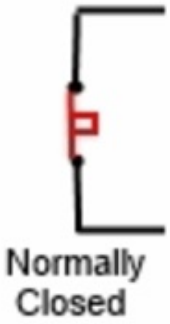
You can see the multiple contacts in the different types of switches above. Usually the center pole is in the center of the pin group if rectangular. Switch one way it connects one set of contacts, the other way connects separate contacts (there may or may not be a "center" where there is no contact).

Use a multimeter (or a battery + resistor + LED) to see which terminals are connected when in various switch positions. Draw a diagram on a scrap of paper (or your Maker notebook) to remember what connects to what.



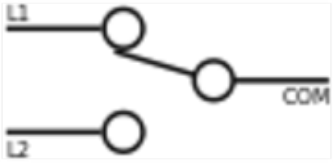
Normally Open Pushbutton (the most common pushbutton)

Normally Closed Push-Button (these are rare!)

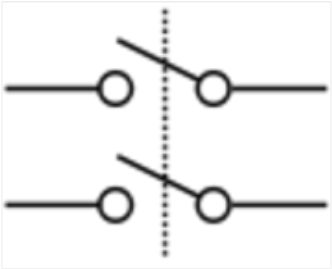


Single Pole Normally Open Switch

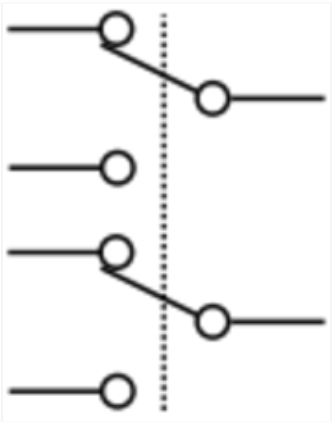
Single Pole, Double Throw Switch



Double Pole, Single Throw Switch



Double pole, Double Throw Switch



Sometimes there is a center position where the switch isn't connected through to any pole, called "center off".

You can see these switches and others on Wikipedia (https://adafru.it/C8e). If you only need a single pole single throw but have one of the switches with additional terminals, you can safely ignore them. Or you can wire something so in one direction you trigger a green light, in the other a red light as an example.

A Special Note on Capacitive Touch "Switches"

You will see many projects using capacitive touch as a switch. Capacitive touch is not a switch in the sense this guide is defining a switch as there is no mechanical motion involved.

The Adafruit Learning System has many guides on the use of Capacitive touch in various circuits. If the switch you are contemplating is capacitive touch instead of mechanical, search the Adafruit Learning System (https://adafru.it/dlu) for more information.

Hook Up Your Switch

The first question that comes up: where do I hook up the switch wires to the microcontroller?

Answer: We'll show you below!

The simplest switch has two terminals which are open when switched one way, and closed when switched the other way. Think of a light switch as an example but they come much smaller for our use.

Using wires, here is a vintage toggle switch (Radio Shack, 1970s) and how the wiring might look:

When the switch is in one position, the contacts are not connected ("off" or "open circuit") and in the other position the circuit is connected ("on" or "closed circuit").

This type of switch is called a normally open single pole single throw switch. More complicated switches are possible but let's understand the basic switch first.

Hookup

You connect both of the terminals to your microcontroller project. Here is the tip on which ones. Select a data pin (digital or analog are ok, it must be a pin that can be read as a digital data pin for input by code).

Connect the other side of the switch to the circuit ground (often marked **GND**).

> Switches are symmetric, so it doesnt matter which switch-pin goes to ground and which switch-pin goes to data

In the drawing, ignore the resistor between the microprocessor pin and the positive voltage - that resistor is internal to the microcontroller board, so you don't need to wire it up yourself. The resistor will connect from the data pin you're using to the **internal logic power voltage** which we label **Vcc**. Depending on your microcontroller **Vcc** may be connected to 5V (Arduino UNO) or 3.3V (which it is on the Circuit Playground) We'll get to that in a second.

Why connect the second connection made to ground? Let's look at the diagram.

Pull "high" or "up" Resistors

The voltage at the micro pins is normally "pulled high" by the resistor when the switch is open due to input pins being high resistance. When the switch closes, the micro pin doesn't see Vcc anymore because the resistor is between it and Vcc, it sees the direct short to ground.

That circuit gives us unambiguous switching between Vcc voltage and zero/ground voltage depending on the switch.

Without the "phantom resistor", the pin would not be connected to anything with the switch open. With electronics, reading an unconnected pin may cause a reading that is undetermined, which is not good in digital on/off readings.

OK, so we know the hookup and know there should be some sort of resistor in there (what I called a phantom, still coming).

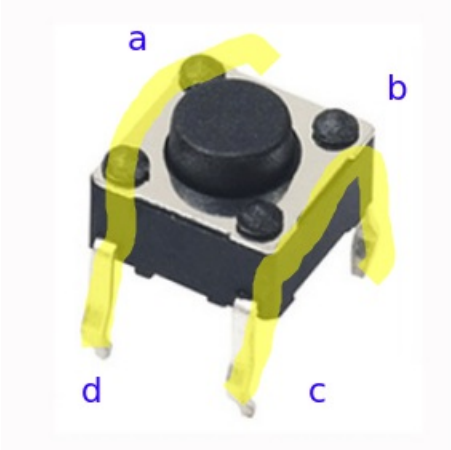
How about those 4 legged breadboard switches. They work fine if you know what they do. See the next page.

What About Breadboard Switches?

Several push buttons are popular in the Maker arena, including smaller and larger buttons that have legs that fit in a breadboard.

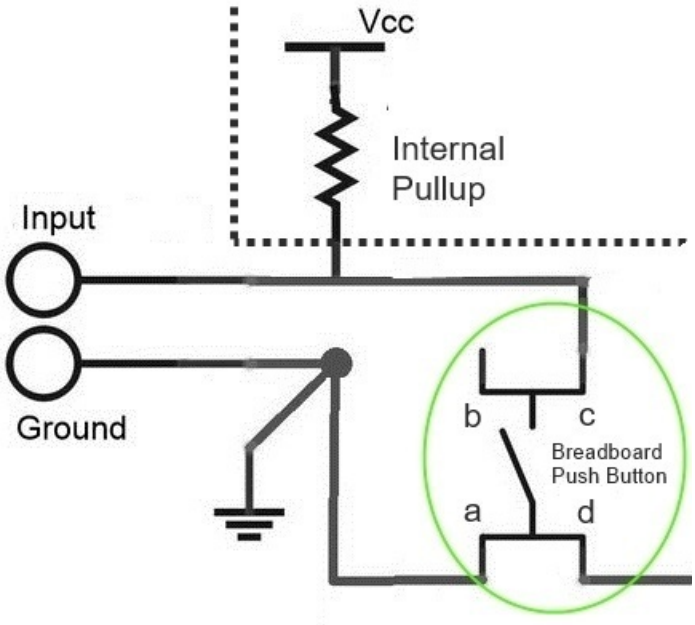
There's four legs so its mechanically more secure when soldering in place on a circuit board, but it doesn't have two switches in it!

There are connections between some of the legs. **a** is already connected to **d** and **b** is already connected to **c**.



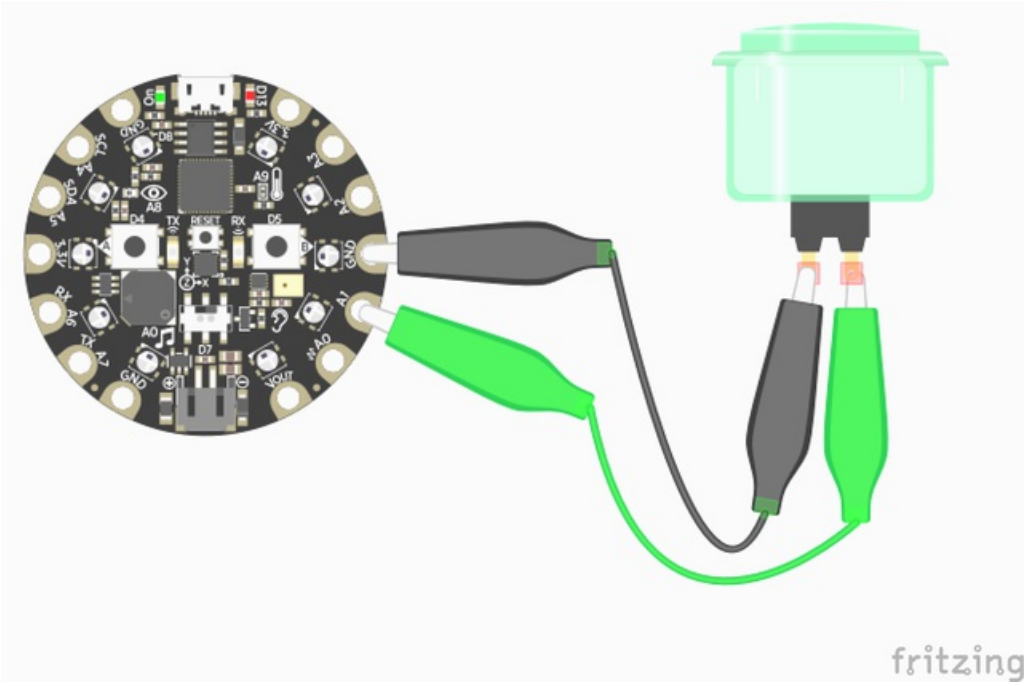
> It's hard to remember which pair are connected - even after years of using them, so the best advice is to use opposite corners as the connections - that way you'll always get one from each pair!

In the diagram below, you can see the hookup of the switch (the switch is circled in green) as a standard normally open switch. The opposite corners are connected (a and c) to ensure we have a single pole normally open switch.


Vcc
Internal
Pullup
Input
Ground
b
c
Breadboard
Push Button
a
d

Code Your Micro

It's fairly easy to code reading a switch in most languages. Refer to the following circuit as we go through the code:



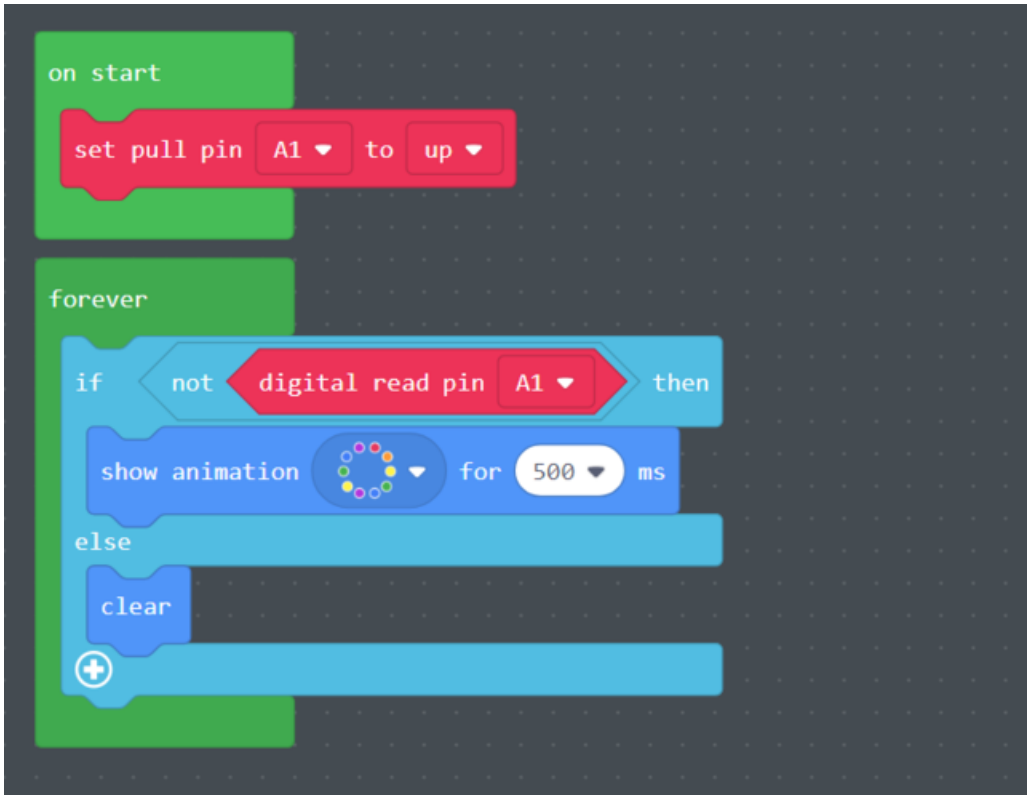
> Though this tutorial has examples of using a switch on an analog pin, all the examples in the tutorial can use either analog or digital pins.

Microsoft MakeCode

MakeCode has reading switches basically baked in. You can read the pins that are available as inputs.

Below is a loop that will read the switch on **A1** on a Circuit Playground Express. This works with any MakeCode compatible board on any pin that can act like a digital input pin (which includes analog pins for modern microcontrollers) which has pullup resistors.

The rainbow effect will be on the NeoPixels on a Circuit Playground Express if activated, otherwise the lights will be off.



https://adafru.it/C8g

https://adafru.it/C8g

Setting pull up resistors

`on start` is the code that sets up our board when everything starts. The `set pull pin A1 to up` is the magic we hinted at earlier. Many modern microcontrollers have the ability to set a resistor pull up (to Vcc) or pull down (to ground) **built in**. The `set pull pin A1 to up` statement (in the red `PINS` group of blocks) enables an internal resistor between the pin (`A1` for my circuit) and Vcc. No external resistor was needed here which saves us on parts and wiring!

Checking Switch Connection

The `forever` loop reads the pin A1 digitally. A1 will always be `high` due to the pull up unless the switch is activated when it will read `low`. For the `if` statement, we need to put a `not` statement in front of the read to activate the statement on a digital low (when the switch grounds the pin). If the program reads `false` (which is low), then `not false` is true and the program shows an animation on the boards LEDs. Your code can have your board do any action you want.

With Pull-Up resistors, you'll always have to check if the pin is connected to ground (LOW, off or False) to determine when the switch has been pressed - which is backwards from what we normally think! Watch out for this switcheroo.

CircuitPython

```
from digitalio import DigitalInOut, Direction, Pull
import board

# button
button_1 = DigitalInOut(board.A1)
button_1.direction = Direction.INPUT
button_1.pull = Pull.UP

while True:
    if not button_1.value:
        print("Button activated")
```

CircuitPython has a library that defines the pins on CircuitPython compatible boards called `boards`. When you `import board` you can use the defined objects. Analog pins are `board.A0`, `board.A1`, etc. and digital pins `board.D4`, `board.D5`, etc.

The `digitalio` library allows the definition of pins for `INPUT` or `OUTPUT` and to have a `Pull.UP`, `Pull.DOWN` if you want it.

As the button is pulled up, it will return `HIGH` when not pushed. You want to look for a `LOW` which indicates the button is activated to do something, here printing to the console.

See the CircuitPython Basics Tutorial (https://adafru.it/C86) for more on using switches.

Arduino

You'll see a good number of Arduino examples on the web. You might be a bit confused there appear to be different ways to code a read. Here is the simple method corresponding to the examples above.

```
void setup() {
  pinMode(A1, INPUT_PULLUP);  // set our pin to an input with a pullup resistor
}

void loop() {
  // read the state of the pushbutton value - if not activated low then loop
  if( ! digitalRead(A1) ) {
     // do something like blink pin 13 LED, or whatever
  }
}
```

Noisy Switching: Bounce and Debounce

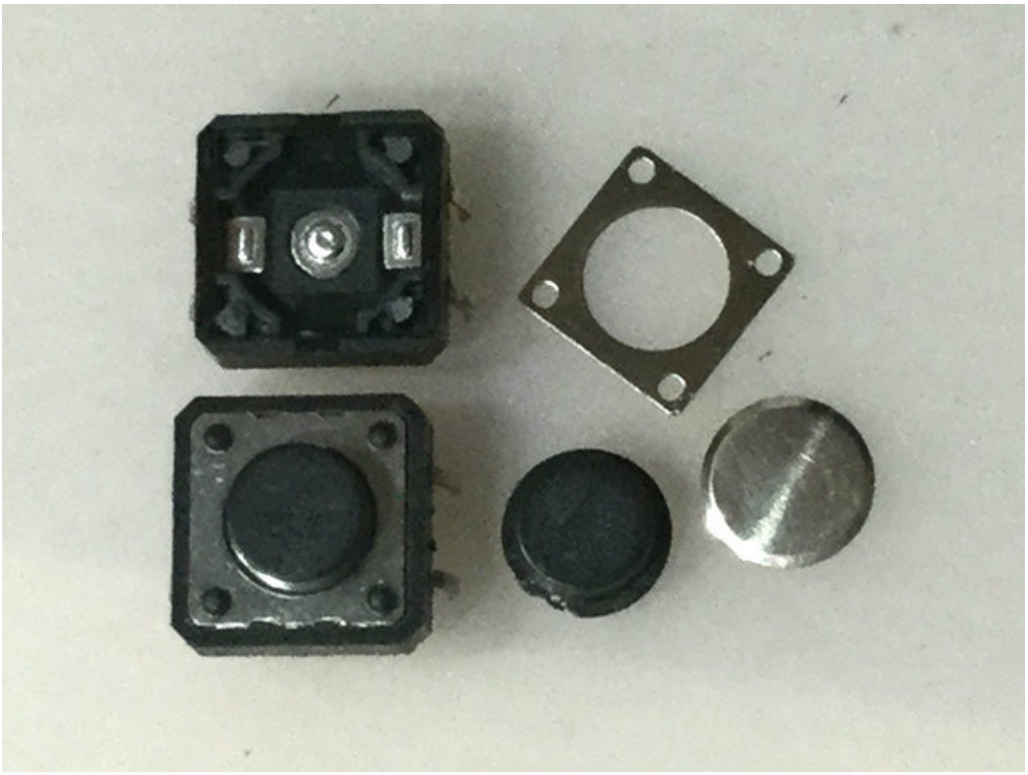
The simple examples above may be all you need - to turn a light on or some other long-lived action. Go for it!

But if you are relying on a switch activation for something timed very fast (less than 10 milliseconds), you may find that the mechanical switch may make contact on and off for a few microseconds before settling down to the state you want. This is called switch bounce, see the next page for more.

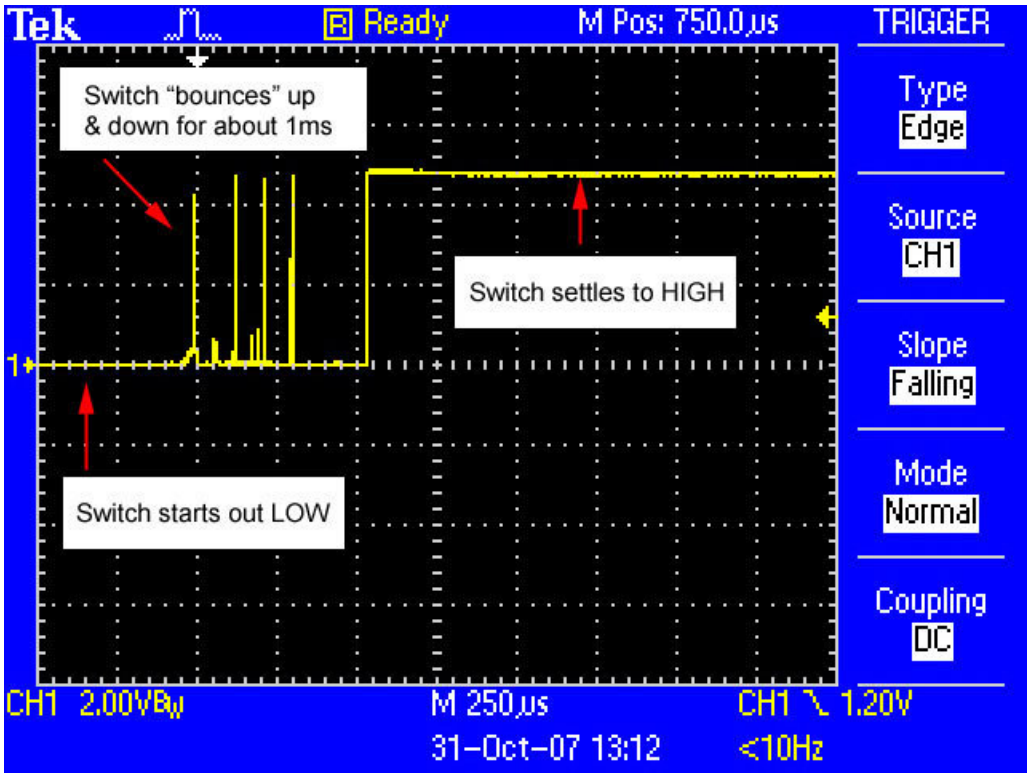
Debouncing

Inside a tactile switch are metal contacts that go to the legs, and then a domed disc that sits on top, when pressed, the dome collapses down and connects the two contacts.

Below you can see the switch body, silver metal disc, black switch cap and the metal square that holds the switch together



The metal disc is springy/bouncy. When you press it down, it wont make *immediate* contact, it will shift around a little bit!



If you are relying on a switch activation for something exact, you may find that the mechanical switch may make contact on and off for a few milliseconds before settling down to the state you want. This is called switch bounce and is considered unwanted noise at best, a failure in some very fast circuits.

> The bounces don't occur when the button is held down or not pressed. They only occur during the press or release of a button.

Here are some tricks to use code to debounce a switch (so you don't have to add more hardware).

MakeCode

Internally, MakeCode handles debouncing (https://adafru.it/C87), so you don't have to do anything more for a noisy switch!

CircuitPython

CircuitPython is interpreted so it doesn't run incredibly fast even in a short polling for a switch press loop. This is to our advantage with debouncing, as the timing on reading an input and any follow-up code should be long enough to not detect any switch noise. If you don't poll the switch less than 10 milliseconds apart (hard to do!) you're fine.

If you somehow get a really fast processor and you are trying to catch presses very fast, you might do something like:

```
if switch.value:
    time.sleep(0.05)  // wait 50 ms see if switch is still on
    if switch.value:
        # ok, it's really pressed
```

but again, this is rarely necessary, if ever.

Arduino

Arduino code does run fast enough where you might poll a switch multiple times within a few milliseconds. So the code might catch the highs and lows of switching noise.

The software debounce can be done a number of ways but there is an example in the standard IDE installation which is listed below. I've modified it for a pullup inside the microcontroller and the pin to A1 like other examples.

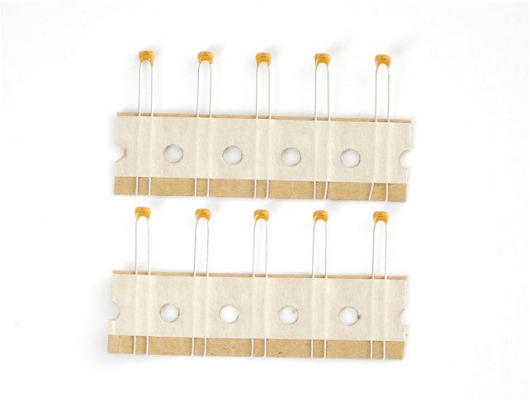
```
/*
  Debounce

  Each time the input pin goes from LOW to HIGH (e.g. because of a push-button
  press), the output pin is toggled from LOW to HIGH or HIGH to LOW. There's a
  minimum delay between toggles to debounce the circuit (i.e. to ignore noise).

  The circuit:
  - LED attached from pin 13 to ground
  - pushbutton attached from pin A1 to ground

  - Note: On most Arduino boards, there is already an LED on the board connected
    to pin 13, so you don't need any extra components for this example.

  created 21 Nov 2006
  by David A. Mellis
  modified 30 Aug 2011
  by Limor Fried
  modified 28 Dec 2012
  by Mike Walters
  modified 30 Aug 2016
  by Arturo Guadalupi

  This example code is in the public domain.

  http://www.arduino.cc/en/Tutorial/Debounce
*/

// constants won't change. They're used here to set pin numbers:
const int buttonPin = A1;   // the number of the pushbutton pin
const int ledPin = 13;      // the number of the LED pin

// Variables will change:
int ledState = HIGH;         // the current state of the output pin
int buttonState;             // the current reading from the input pin
int lastButtonState = HIGH;  // the previous reading from the input pin

// the following variables are unsigned longs because the time, measured in
// milliseconds, will quickly become a bigger number than can be stored in an int.
unsigned long lastDebounceTime = 0;  // the last time the output pin was toggled
unsigned long debounceDelay = 50;    // the debounce time; increase if the output flickers

void setup() {
  pinMode(buttonPin, INPUT_PULLUP);
  pinMode(ledPin, OUTPUT);

  // set initial LED state
  digitalWrite(ledPin, ledState);
```

```
  digitalWrite(ledPin, ledState);
}

void loop() {
  // read the state of the switch into a local variable:
  int reading = digitalRead(buttonPin);

  // check to see if you just pressed the button
  // (i.e. the input went from LOW to HIGH), and you've waited long enough
  // since the last press to ignore any noise:

  // If the switch changed, due to noise or pressing:
  if (reading != lastButtonState) {
    // reset the debouncing timer
    lastDebounceTime = millis();
  }

  if ((millis() - lastDebounceTime) > debounceDelay) {
    // whatever the reading is at, it's been there for longer than the debounce
    // delay, so take it as the actual current state:

    // if the button state has changed:
    if (reading != buttonState) {
      buttonState = reading;

      // only toggle the LED if the new button state is HIGH
      if (buttonState == HIGH) {
        ledState = !ledState;
      }
    }
  }

  // set the LED:
  digitalWrite(ledPin, ledState);

  // save the reading. Next time through the loop, it'll be the lastButtonState:
  lastButtonState = reading;
}
```

A Quick Note on Hardware Debouncing

Hardware costs money so many manufacturers do not want to spend extra money for hardware to debounce switches. A 0.1 uF capacitor across the switch contacts is often used in simple circuits with better hardware used depending on the circuit requirements. Hardware debouncing is its own art and you'll find other guides on the Internet and in books discussing it.

It is worth it to put your homework into hardware switch debouncing if you really require a near-perfect switch transition detection.



0.1uF ceramic capacitors - 10 pack

$1.95
IN STOCK

ADD TO CART

If Your Microcontroller Doesn't Have Internal Resistors

All of our examples so far show and use an internal pullup.

There are also internal pull down resistors on many microcontroller inputs. In this case you connect your switch's non-digital-pin to Vcc instead of Ground. Vcc might be 5V or 3.3V depending on the 'logic level' of your microcontroller.

But what if your microcontroller does not have an internal pullup or pulldown?

No worries! You can use an external resistor!

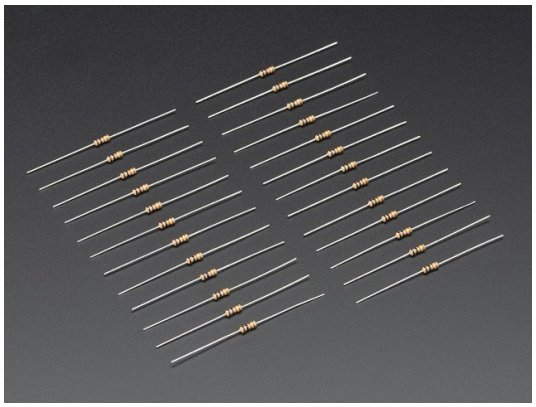
Connect one end to the same microcontroller input pin you use for the switch, and the other end to the positive voltage (usually 5 or 3.3 volts). Resistors can be connected either way, they are not polarized so you cannot make the connections "backwards".

The value of the resistor is not usually critical. any resistor between 1,000 ohms (1 KΩ) and 100,000 ohms (100KΩ) should work. If the resistor is too small, you may waste current pushing the button. If too large, the pull up effect may be so weak as to not be detected by the pin electronics.

A value of 10,000 ohms (10KΩ) or so is a good value and has become a standard for pull up resistors. That's why you'll see so many of them.

You can buy resistors nearly everywhere or find them in salvaging old, unused electronics.

Or you can get them at the Adafruit shop



Through-Hole Resistors - 10K ohm 5% 1/4W - Pack of 25

$0.75
IN STOCK

ADD TO CART

Wiring Example

Above, an external 10K ohm resistor is connected between the microcontroller pin used (**#2** pin) and Vcc, which is +5 volts for this board (**5V** pin). The switch (a push button in this vase) has one connection to ground, another to the resistor and the microcontroller pin.

The microcontroller pin, if read in code, should be `HIGH` if the switch is not pressed, `LOW` if the switch is pressed.

This method can be used for nearly any type of digital circuit looking for a high or low depending on a switch state.