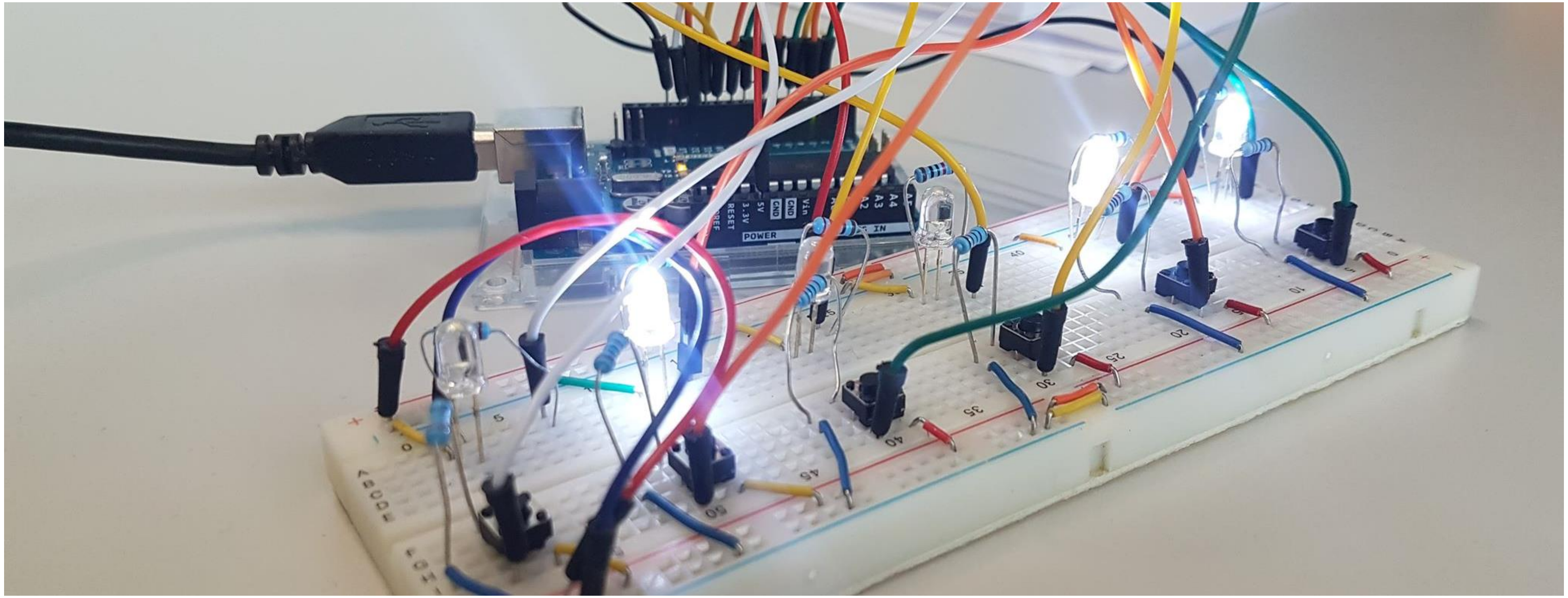
IAT 884 – Week 4 – Workshop 4

Alissa Antle and Annemiek Veldhuis (*ahv1@sfu.ca*)

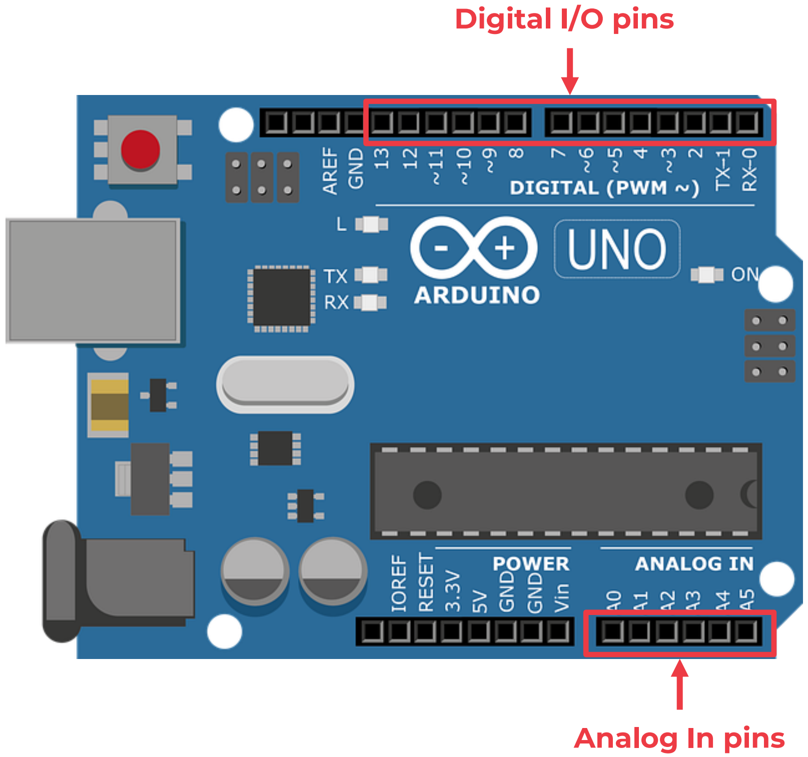


Microcontrollers

Input

Microcontrollers

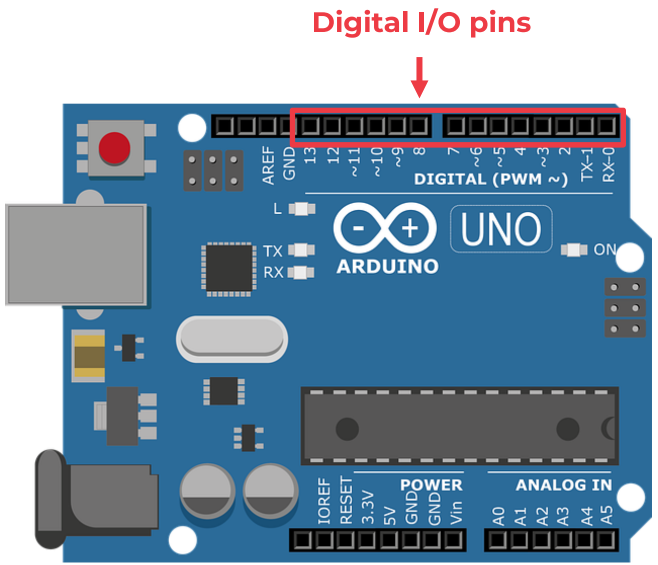
Input pins



Microcontrollers
Digital in

Signals are interpreted as **high** or **low**.
High = 5V
(Actually, >3V is interpreted as HIGH)
Low = 0V

Process:
1. Read input from one of the digital pins (0-13).
2. Detect the type of signal using an **IF** statement.



Microcontrollers
Digital in

```
pinMode(2, INPUT);              // Set pin 2 as input
val = digitalRead(2);           // Read the input pin
```

Microcontrollers

Digital in

```
int inPin = 7;                          // pushbutton connected to digital pin 7
int val = 0;                            // variable to store the read value

void setup(){
        Serial.begin(9600);
        pinMode(inPin, INPUT);          //sets the digital pin 7 as input
}

void loop(){
        val = digitalRead(inPin);       // read the input pin
        Serial.println(val);            //print the incoming value
        if(val==HIGH){
                Serial.println("High");
        } else {
                Serial.println("Low");
        }
}
```

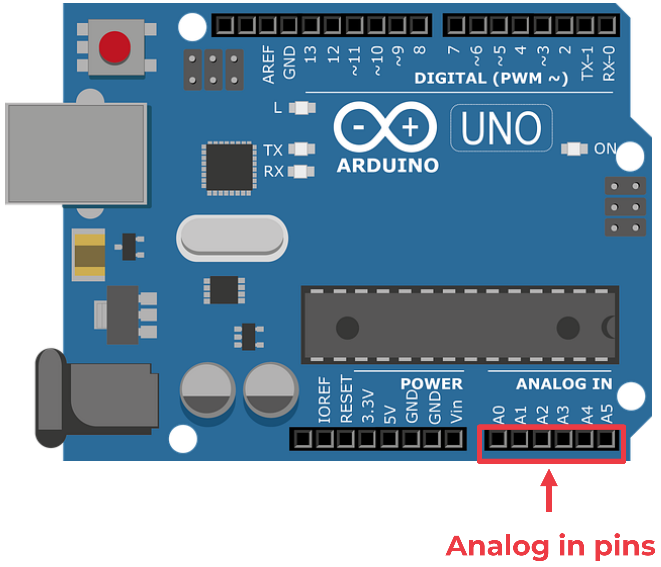
Microcontrollers

Analog in

The Arduino will convert any analog input signal between 0-5V into a number between **0-1023**. (10 Bit ADC)

Remember that Analog Out send signals using 0-255

There are 6 Analog pins, and they are separate from the digital pins.



Microcontrollers

Analog in

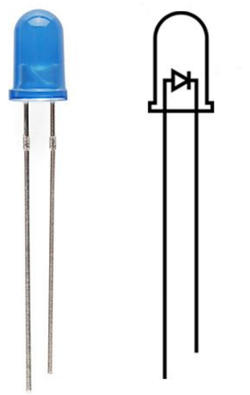
```
val = analogRead(analogPin);          // read the input pin
```

Microcontrollers

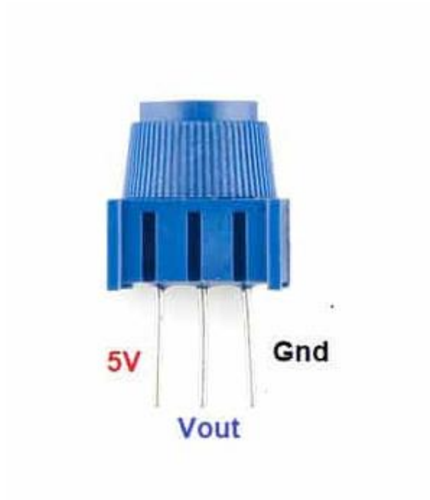
Analog in

```
int analogPin = 3;
int val = 0;                                          // variable to store the read value

void setup(){
            Serial.begin(9600);
}

void loop(){
            val = analogRead(analogPin);              // read the input pin
            Serial.println(val);                      // print the incoming value
}
```

Microcontrollers
I/O summary

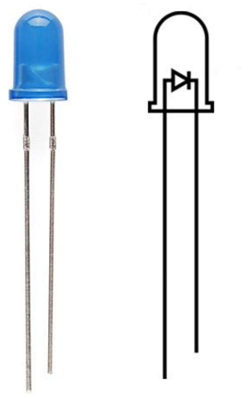


digital

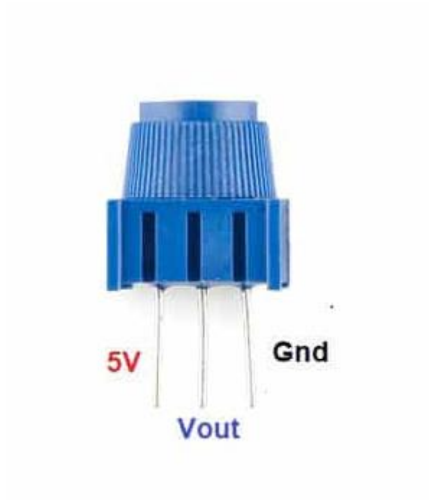


analog

Microcontrollers
I/O summary



digitalRead
digitalWrite
analogWrite



analogRead
analogWrite

Microcontrollers

I/O summary

digital**Read**(pin);

input of 0V (<3V) = 0

input of 5V (>3V) = 1

digital**Write**(pin, KEYWORD);

output LOW = 0V

output HIGH = 5V

analog**Read**(pin);

Reads a number between 0-1023

0V is 0

5V is 1023

analog**Write**(pin, value);

Writes a number between 0 – 255

0 is 0V

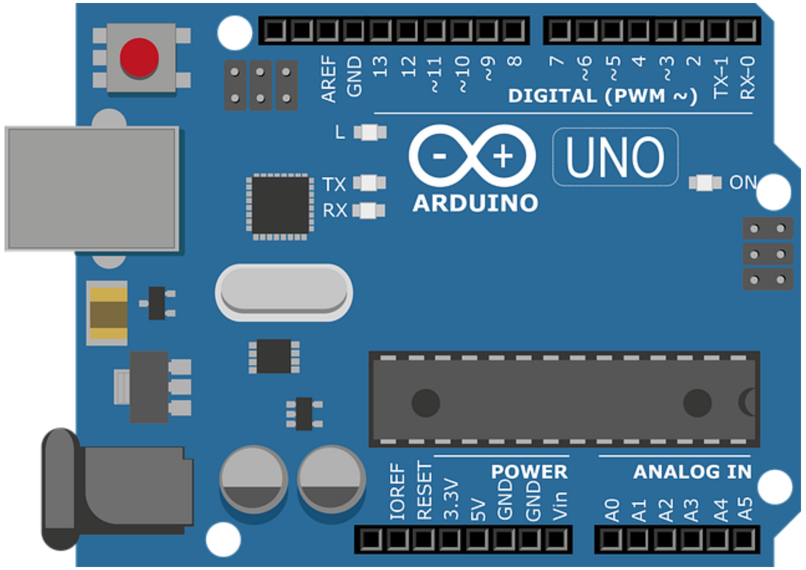
255 is 5V

Pull-up resistors

Pull-up resistors
Floating

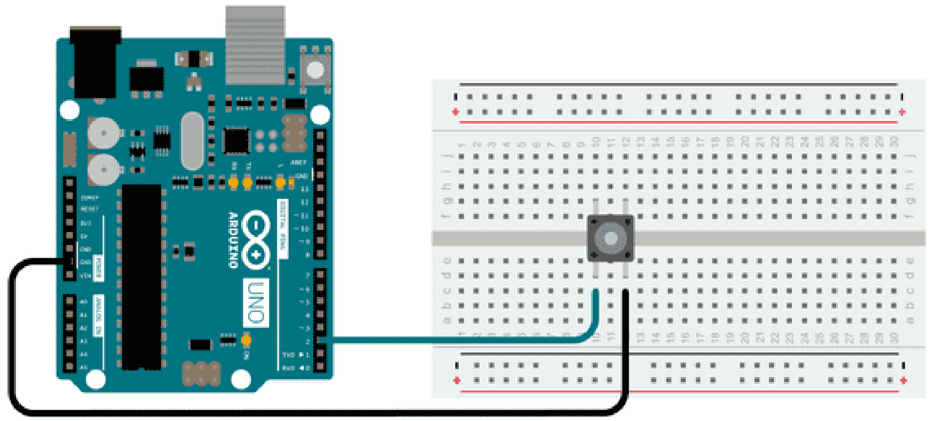
If there is nothing connected to a pin and your program reads the state of the pin, will it be high (pulled to VCC) or low (pulled to ground)?

It is difficult to tell. This phenomena is referred to as *floating*.



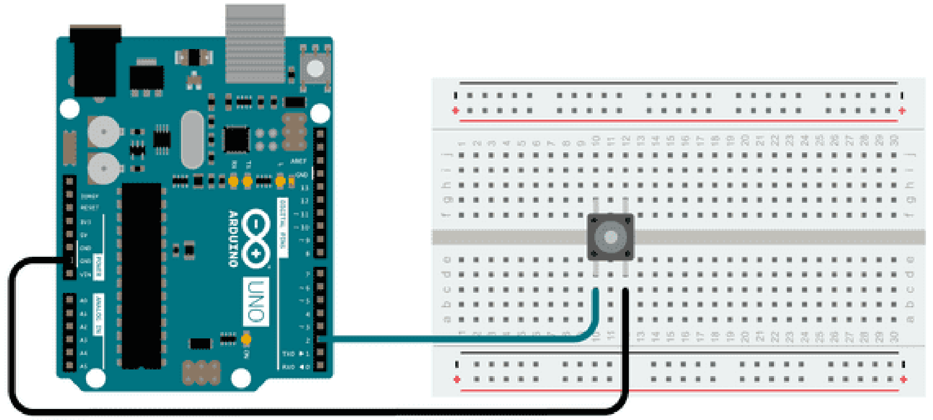
Pull-up resistors
Floating

Does this circuit have a floating pin?



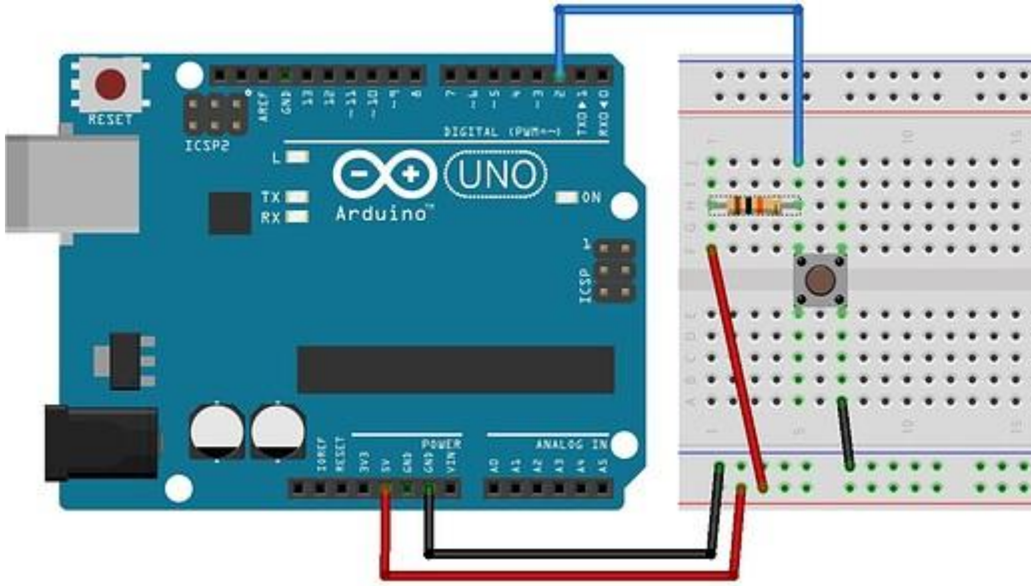
Pull-up resistors

To prevent floating, a **pull-up** or **pull-down resistor** will ensure that the pin is in either a high or low state, while also using a low amount of current.



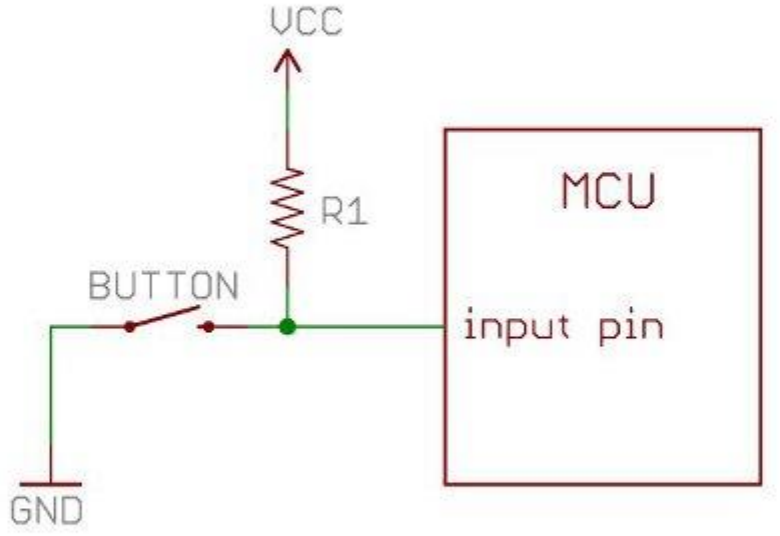
Pull-up resistors

To prevent floating, a **pull-up** or **pull-down resistor** will ensure that the pin is in either a high or low state, while also using a low amount of current.



Pull-up resistors

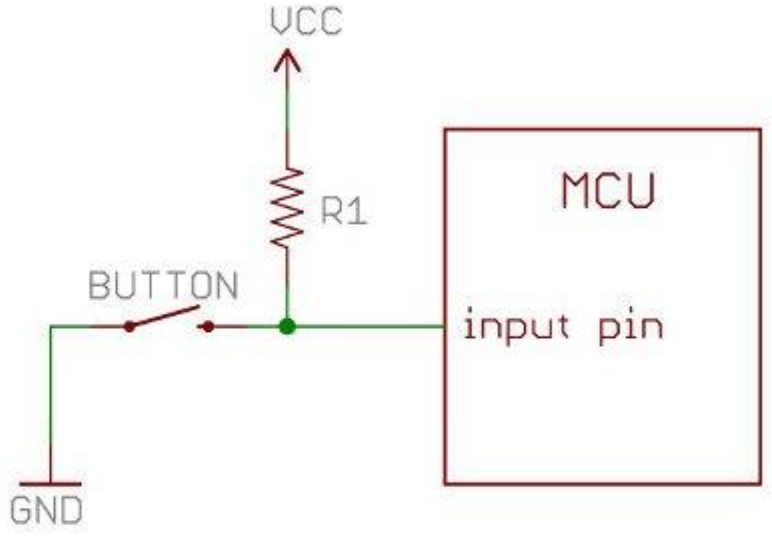
To prevent floating, a **pull-up** or **pull-down resistor** will ensure that the pin is in either a high or low state, while also using a low amount of current.



Pull-up resistors

If the button isn't pressed, what does the Arduino read?

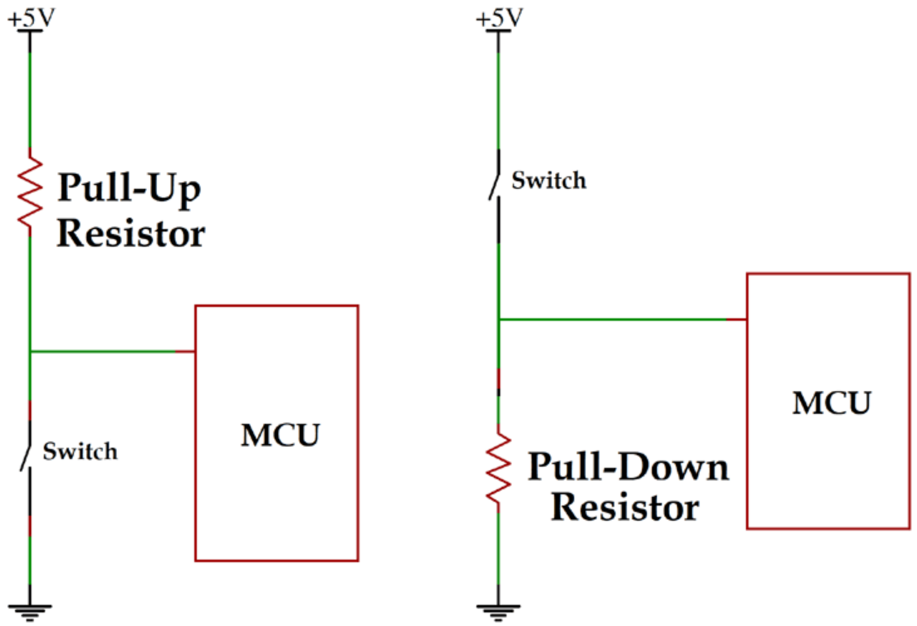
How can you switch this around?



Pull-up resistors

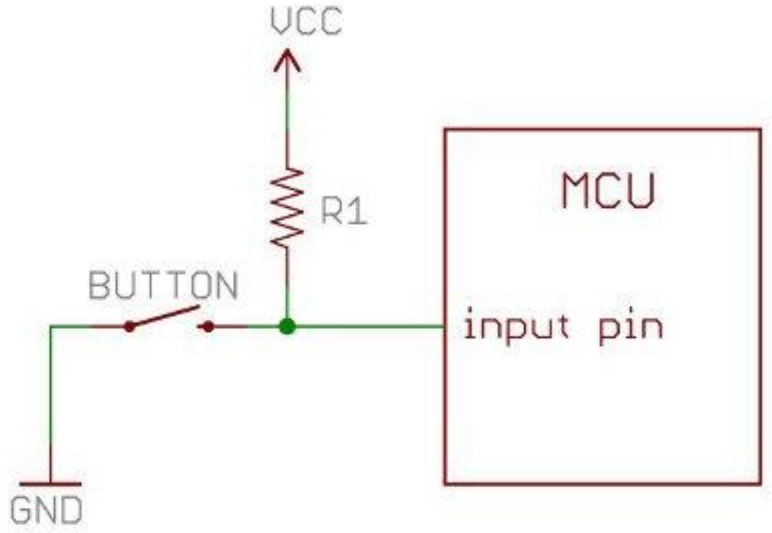
If the button isn't pressed, what does the Arduino read?

How can you switch this around?



Pull-up resistors

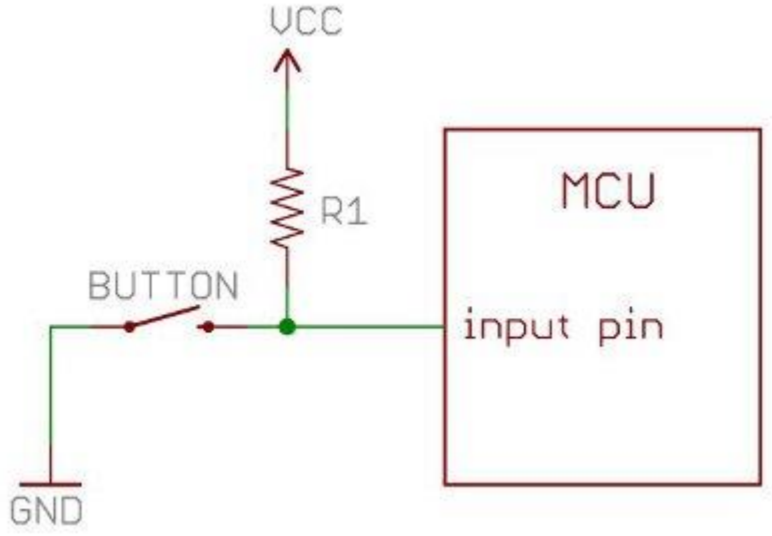
Why can’t we directly connect the button to VCC (+)?



Pull-up resistors

Use Ohm’s law to calculate the Pull-up resistor value.

I = the maximum of current you allow to flow through the circuit when the button is pressed (e.g., 1mA).



Pull-up resistors

Arduino also has build-in pullup resistors that can be enabled through code:

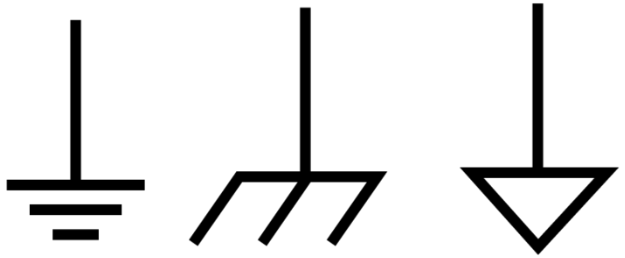
pinMode(buttonPin, **INPUT_PULLUP**);

Common Ground

Whenever you are sending signals from one circuit to another (i.e., from a sensor to the Arduino), you must make sure everything shares a COMMON GROUND.

This means that the negative wires from all components must be connected.
If you are using a separate power supply that provides 12v (remember that the Arduino runs on 5v) you still need to connect the negative wires together. This creates the common ground.

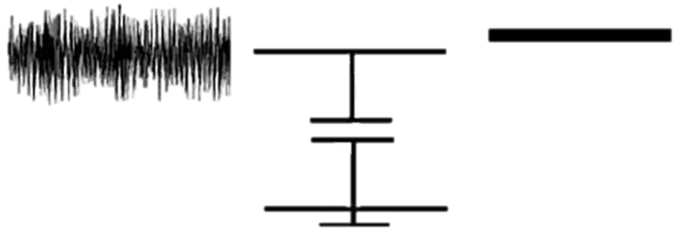
All INPUTS, OUTPUTS, and SOURCES must share a common ground.



Decoupling Capacitors

Decoupling capacitors are low strength capacitors placed between voltage and ground. Decoupling capacitors smooth signals and can greatly improve readings on sensors.

Decoupling capacitors are generally between 0.1 uF and 22 uF (Micro Farrads).



Scaling Functions

A common task you will need to perform is the conversion of an input value into a range that is usable. For example, you receive an analog input value between 0-1023 and you want to send out a proportional analog signal to some device (0-255).

Scaling functions help us do this.

The Arduino Library has a built-in scaling function called map()

map(value, fromLow, fromHigh, toLow, toHigh);

Scaling Functions

```
mapped_value = map(initial_value, 0, 50, 0, 1024);
```

Will convert a number between 0 – 50 to a number between 0 -1024

if initial_value = 0 then mapped_value = 0

if initial_value = 50 then mapped_value = 1024

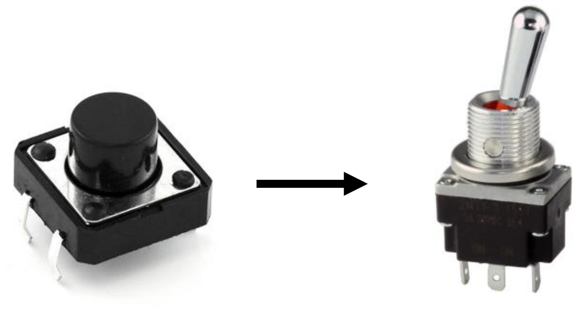
if initial_value = 25 then mapped_value = 512

Edge Detection

Edge Detection is a method that allows you to use a pushbutton as a switch.
It uses **input signal change** (e.g., from LOW to HIGH) as the trigger for some action.

After receiving a signal change **the system compares the current signal with the previous signal.** If the signals are different then we know the button state has changed.

Using Edge Detection, you can count the number of times a button has been pressed.



Edge Detection

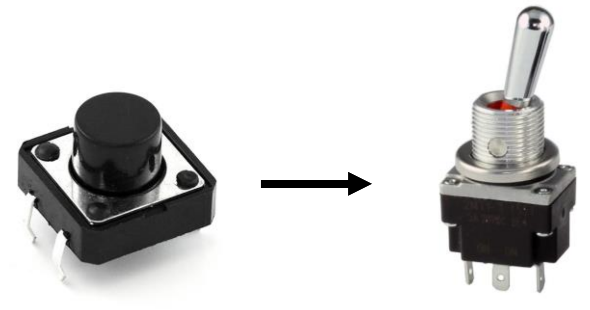
```
int switchPin = 2;
int switchCounter = 0;
int switchState = 0;
int lastSwitchState = LOW;

void setup(){
        pinMode(switchPin, INPUT);
        Serial.begin(9600);
}

void loop(){
        switchState = digitalRead(switchPin);                   // read the switch
        if (switchState != lastSwitchState){                    // compare to previous state
                if (switchState == HIGH){
                        switchCounter++;                        // if the state has changed, increment
                        Serial.println(switchCounter);          // print number
                }
                lastSwitchSyaye = switchState;                  // save the current state as the last state
                                                                // for next time through the loop
        }
}
```

Debouncing

Debouncing is similar to Edge Detection. Debouncing an input removes faulty readings due to the mechanical properties of the button.

When a button is pushed, even though it is in one state, there can be faulty readings where the button's spring "bounces" quickly back and forth between on and off.

To prevent extra state changes from being counted we make sure that the button has been in a state a specific amount of time before registering the button press.



```
int inPin = 7;                                          // the number of the input pin
int outPin = 13;                                        // the number of the output pin
int state = HIGH;                                       // the current state of the output pin
int reading;                                            // the current reading from the input pin
int previous = LOW;                                     // the previous reading from the input pin
long time = 0;                                          // the last time the output pin was toggled
long debounce = 200;                                    // the debounce time, increase if the output flickers

void setup() {
        pinMode(inPin, INPUT);
        pinMode(outPin, OUTPUT);
}
void loop(){
        reading = digitalRead(inPin);
        if (reading == HIGH && previous == LOW && millis() - time > debounce) {
                if (state == HIGH){                     // … invert the output
                        state = LOW;
                } else {
                        state = HIGH;
                        time = millis();                // remember when the last button press was
                }
                digitalWrite(outPin, state);
                previous = reading;
        }
}
```

Debouncing

```
int inPin = 7;                    // the number of the input pin
int outPin = 13;                  // the number of the output pin
int state = HIGH;                 // the current state of the output pin
int reading;                      // the current reading from the input pin
int previous = LOW;               // the previous reading from the input pin
          // the follow variables are long's because the time, measured in miliseconds,
          // will quickly become a bigger number than can be stored in an int.
long time = 0;                    // the last time the output pin was toggled
long debounce = 200;              // the debounce time, increase if the output flickers
```

Debouncing

```
void loop() {
        reading = digitalRead(inPin);
                // if we just pressed the button (i.e. the input went from LOW to HIGH),
                // and we've waited long enough since the last press to ignore any noise
        if (reading == HIGH && previous == LOW && millis() - time > debounce) {
                        // ... invert the output
                if (state == HIGH) {
                        state = LOW;
                } else {
                        state = HIGH;
                                // ... and remember when the last button press was
                        time = millis();
                }
        digitalWrite(outPin, state);
        previous = reading;
}
```

Debouncing

You can also use the Bounce2 library for debouncing

Exercises

Exercises

Download the Week 4 Handout on the wiki.

Get as far as possible with the exercises during the lecture time. Complete them at home if you can't finish.

During remote learning: create the circuits through Tinkercad circuits.

Send a document with your name, screenshots of your circuits, and answers to the questions to ***ahv1@sfu.ca***.
Deadline: Sunday 11.59pm