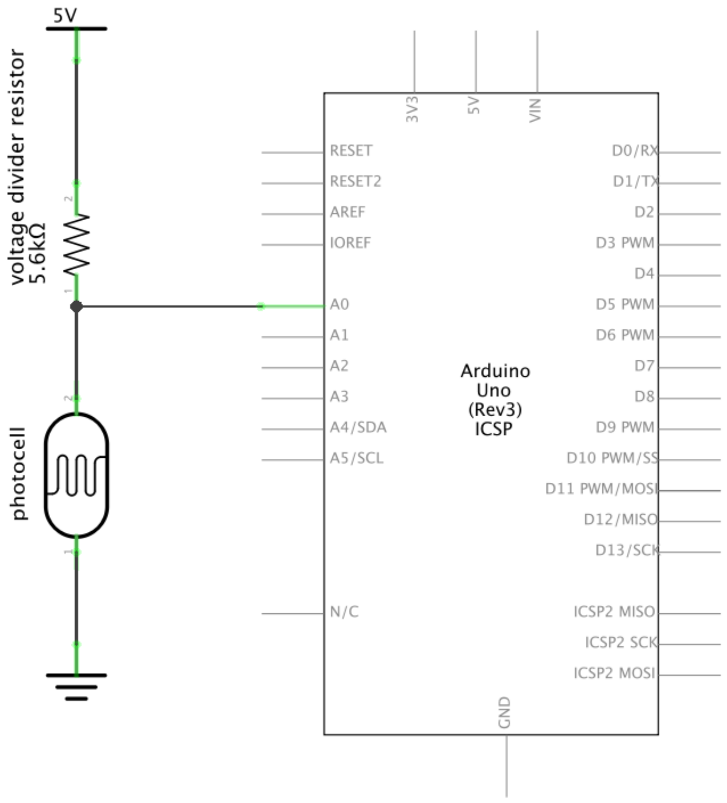
Basic photocell use on an Arduino



A standard photocell is very easy to wire, as shown in the schematic.

The purpose of the 5.6kΩ resistor in series with the photocell is to make it so that the photocell's changing resistance is reliably detectable.

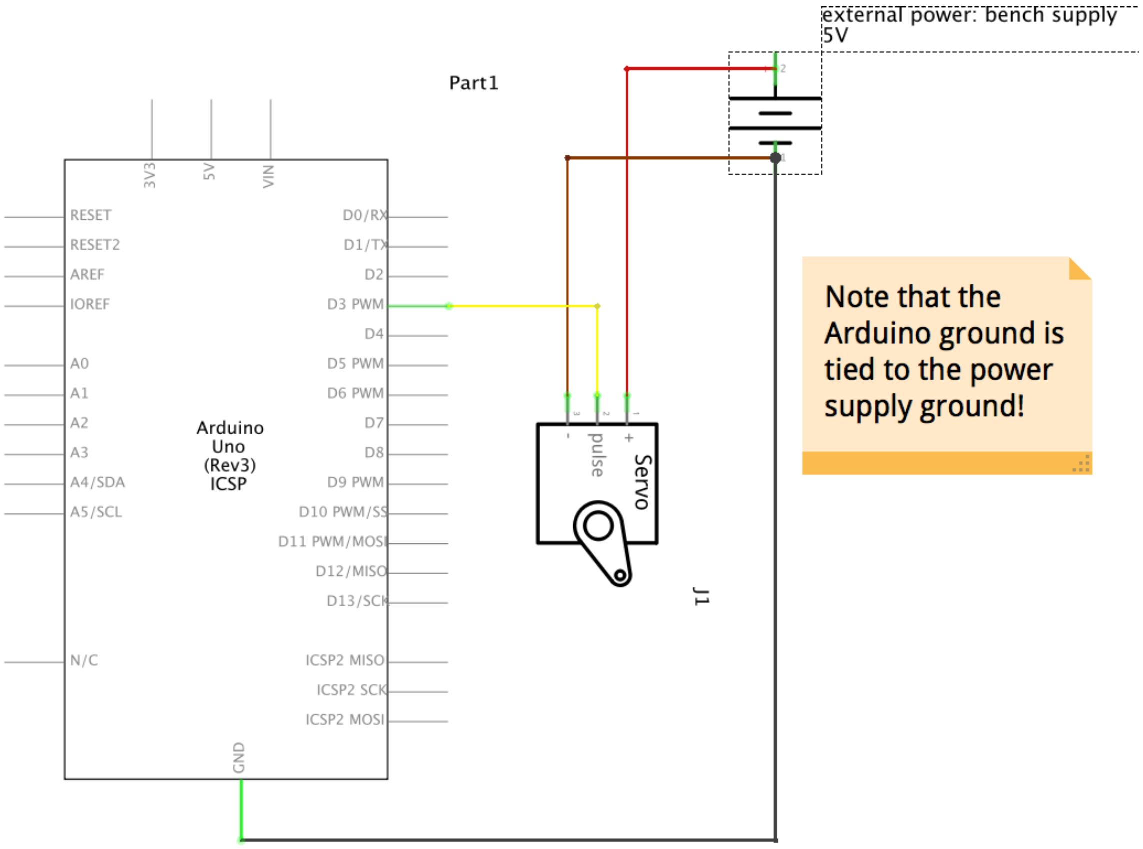
You might be tempted to attach 5V to one side of the photocell, and the other side to A0. But! Consider that at very high photocell resistances this is like leaving A0 with a floating input (which would give you unpredictable values); and at any lower values, it's just wiring 5V right into A0 (which should just read as 1023 all the time). That's why instead of doing that you build a "voltage divider" circuit like the one shown.

Sample Arduino sketch to use serial monitor feedback to display the photocell value

After uploading this code to the Arduino, in the desktop software go Tools–>Serial Monitor, or starting in Arduino 1.6.0, you can try using Tools–>Serial Plotter to see a graph of the incoming data over time.

```
int photocell = A0; // shortcut to refer to the photocell pin later

void setup() {
  pinMode(photocell, INPUT); // we will be reading the photocell pin
  Serial.begin(9600); // starts serial communication at 9,600 baud (the rate)
}

void loop() {
  int readVal; // initialize a new integer to store the photocell value
  readVal = analogRead(photocell); // do the analog read and store the value
  Serial.println(readVal); // push the most recent value to the computer
  delay(50); // slow the loop down a bit before it repeats
}
```

Basic servo use on an Arduino



Note that the Arduino ground is tied to the power supply ground!

Servo wires:

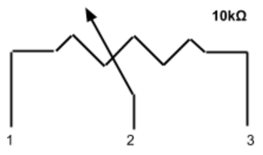
red: ~+5V
brown: ground
yellow: digital signal

You *can* use the Arduino 5V supply to run the servo, but it will be slow and might cause the Arduino to “brown out,” i.e. suffer a voltage droop. That’s why it’s good practice to use an external power source like a battery or bench supply.

Sample Arduino sketch to move the servo back and forth between 10º and 270º

```
#include <Servo.h> // this allows you to use the Servo library

Servo myLittleMotor; // you can call the servo whatever you want
int servoPin = 3; // pin the servo data line is plugged into

void setup(){
     myLittleMotor.attach(servoPin); // set up the servo on that data pin
}

void loop(){
     myLittleMotor.write(10); // tell the servo to go to 10°
     delay(500); // wait a half second
     myLittleMotor.write(170); // tell the servo to go to 170°
     delay(800); // 8/10ths of a second
}
```

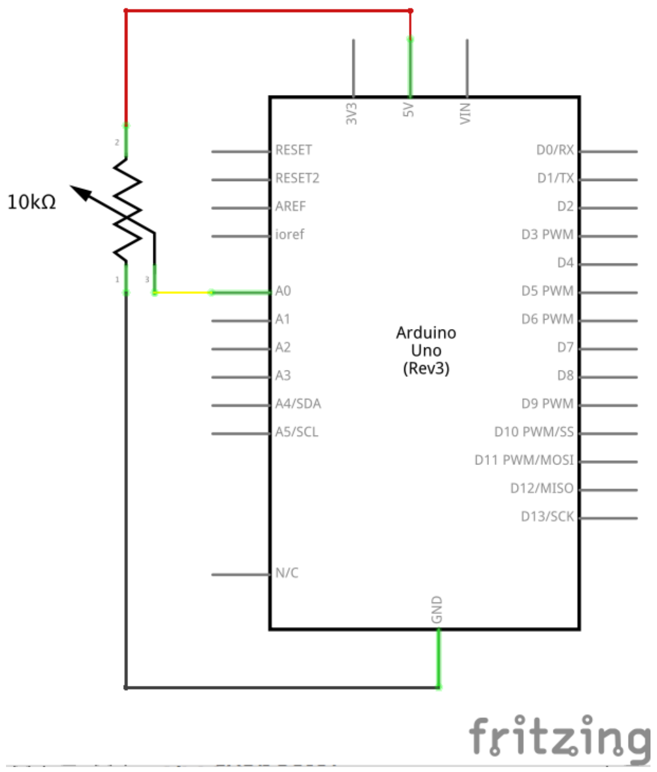
Basic potentiometer use on an Arduino



A potentiometer is a variable resistor with three attachment points. We can take a look at a schematic drawing of the device to better understand what's going on inside:

There are two outside legs, here numbered 1 and 3. These are connected to each other by a resistor, whose value is written near the potentiometer: in this case, it's 10kΩ (10,000 Ohms). This resistance doesn't change.

If you only used pins 1 and 3, this potentiometer would simply be a 10kΩ resistor. It's the last leg, #2 in this diagram, where the device actually does something. Leg #2 is called the *wiper*, and it can be moved physically (by turning a knob or sliding a slider) so that it electrically attaches to the resistor inside the potentiometer in different locations.



When this wiper is all the way to the left, there is no resistance between it and pin 1: in this case, pins 1 and 2 have 0Ω between them, and pins 2 and 3 have 10kΩ between them—vice versa if it's turned to the right.

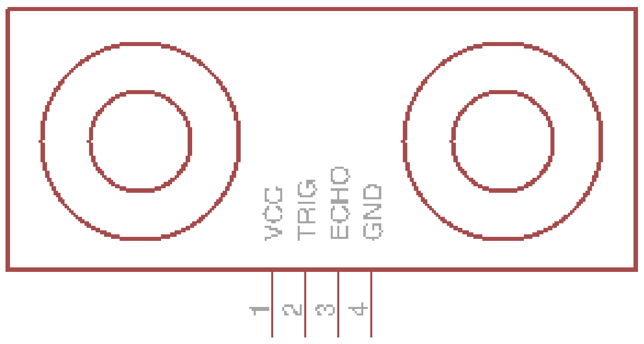
The real magic happens in the middle: if the wiper were, for instance, perfectly centered, then there would be 5kΩ of resistance between pins 1 and 2, and also 5kΩ of resistance between pins 2 and 3. By moving the wiper to any intermediate point between far left and far right, any resistance between 0Ω and 10kΩ can be selected.

The typical Arduino wiring is to attach pin 1 to ground; pin 3 to 5V; and pin 2 (the wiper) to any analog input on the Arduino. Then, performing an `analogRead()` on that pin will produce a value that changes based on the position of the wiper.

Sample Arduino sketch to do `analogRead()` and get `Serial` feedback

```
int POTPIN = A0; // variable to store the pin the potentiometer is wired to

void setup() {
  pinMode(POTPIN, INPUT); // setting the potentiometer pin as an input
  Serial.begin(9600); // starts serial communication at 9,600 baud (the rate)
}

void loop() {
  int readVal; // initialize a new integer to store the potentiometer value
  readVal = analogRead(POTPIN); // do the analog read and store the value
  Serial.println(readVal); // push the most recent value to the computer
  delay(50); // slow the loop down a bit before it repeats
}
```

Basic ultrasonic ranger use on an Arduino



Ultrasonic ranger wires:

VCC: +5V
trigger and echo are data pins which connect with the pins as described in the sketch
GND: ground

The ultrasonic ranger, like most small simple digital devices that are Arduino inputs, does not use much current and so it's fine to run off of the Arduino's 5V supply.

Sample Arduino sketch to read distance

Use the Arduino IDE's built-in Library Manager to add the NewPing library: Sketch–>Include Library–>Manage Libraries–> and search for "NewPing." Once it's installed, go to File–>Examples–>NewPing–>NewPingExample to load the below example sketch. (Note: I've lightly modified the comments for clarity and brevity.)

```
#include <NewPing.h>

#define TRIGGER_PIN  12  // attach the trigger pin to Arduino pin 12
#define ECHO_PIN     11  // attach the echo pin to Arduino pin 11
#define MAX_DISTANCE 200 // Maximum distance we want to ping for (cm)

NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE); // looks like Servo myservo!

void setup() {
  Serial.begin(115200); // note that this is not 9600, the usual serial rate
}

void loop() {
      delay(50); // Wait 50ms between pings
      Serial.print("Ping: ");
      Serial.print(sonar.ping_cm()); // do pinging, return result in centimeters
      Serial.println("cm");
}
```

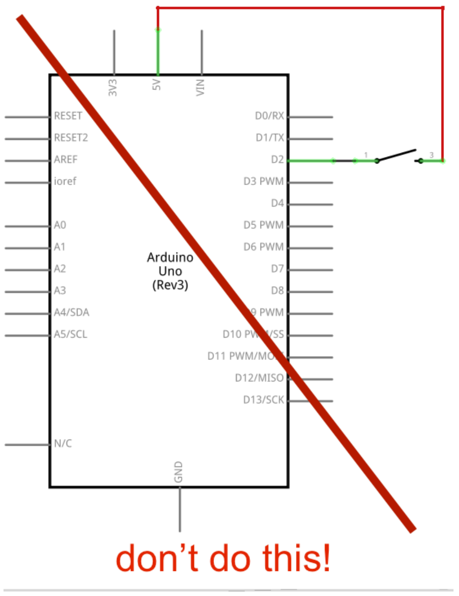
Basic switch/button use on an Arduino

Switches and buttons are easy to wire up once you understand a few basics:

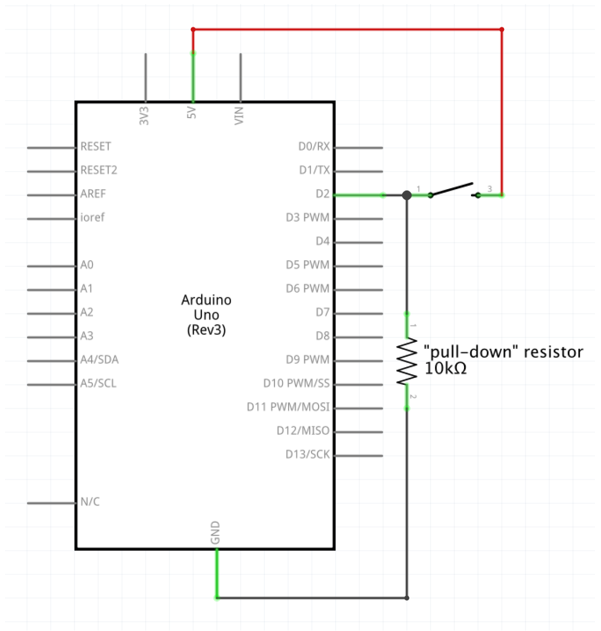
1) For reliable readings, some electrical input must always be attached to the input pin
2) On an Arduino Uno, 5 volts means `HIGH` and 0 volts (ground) means `LOW`
3) Buttons and switches may have non-obvious internal wiring!

An easy mistake

Many beginners often build a circuit like the one to the right—>



Makes sense. When the button is pushed, the input pin sees 5V, and when it's not pushed, it will see 0V so it will be grounded (because nothing is plugged in to it). Right?



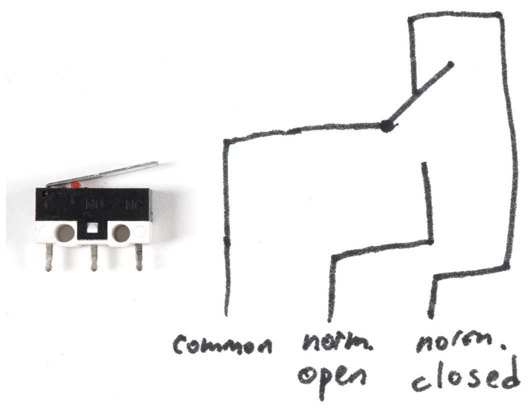
Nope. As point #1 says above, you must wire *some* electrical input to an input pin at all times. When the button on the right is not pushed, pin 2 is connected to nothing but a bit of wire. It will *not* reliably read 0 volts, because of the way digital inputs are read on the Arduino.

The solution? Add a "pull-down resistor" to make it so that when the button is not being pushed, the input pin will be connected to ground, as shown on the left.

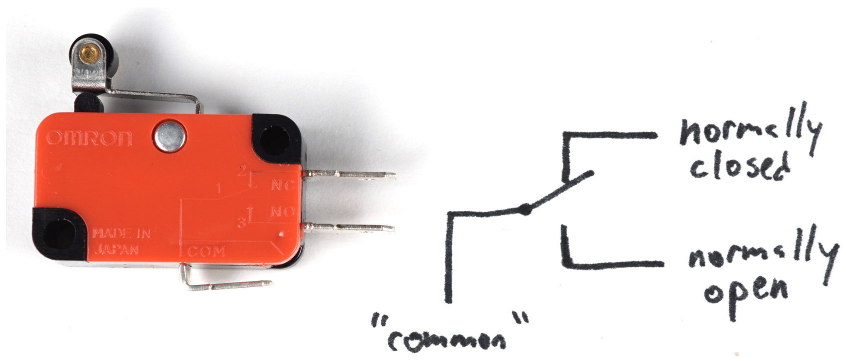
Simple Arduino code to read a button or switch and send the result via Serial monitor:

```
const int BUTTONPIN = 2; // button plugged into pin 2

void setup() {
  pinMode(BUTTONPIN, INPUT); // set the pin up as an electrical input
  Serial.begin(9600); // start serial communication with the computer
}

void loop() {
  int buttonVal = 0; // make a new variable to hold the button state
  buttonVal = digitalRead(BUTTONPIN); // find the current button state
  Serial.println(buttonVal); // send the value back to the computer
  delay(50); // wait a bit before doing it again
}
```

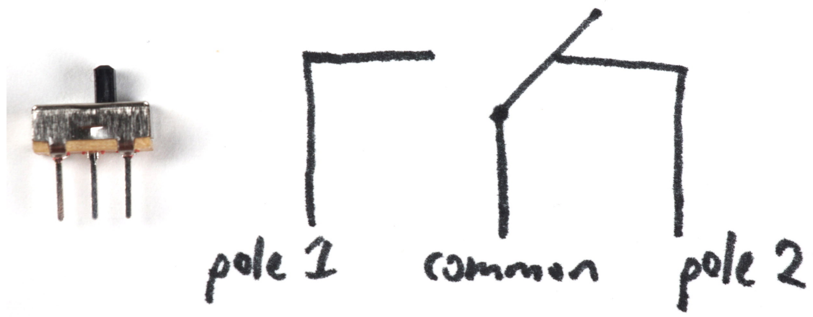
micro lever switch:



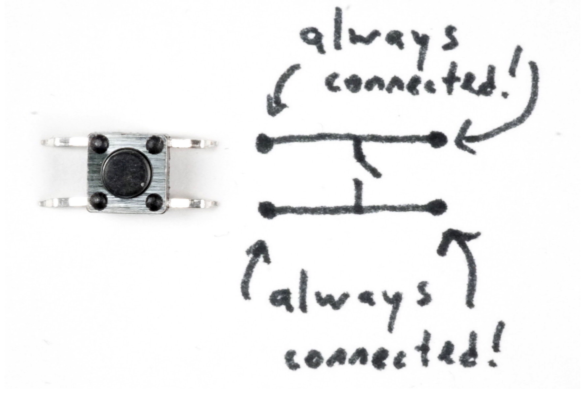
lever switch:



single pole double throw (SPDT) switch:



"tactile" pushbutton switch:


always connected!
always connected!