ARDUINO

Workshop
HCI Technologies

Danica Mast
Chris Heydra
Version 6.1b 25-05-2020

Table of contents

Introduction

The exercises in this reader are intended to get you started with the Arduino. Through these exercises you gain experience in connecting electronics to the micro-controller and you practice programming.

What is Arduino?

Arduino is an open-source electronics platform for building prototypes. It is based on flexible, easy-to-use hardware and software. Intended for artists, designers and anyone interested in creating interactive objects or environments.

Arduino consists of an open source micro-controller and a software environment to program it in.

Arduino projects can run stand-alone (without a computer), but can also communicate with software that runs on a computer (eg Processing, MaxMSP). The Arduino micro-controller can receive input from sensors and send output to actuators.

Examples of sensors:

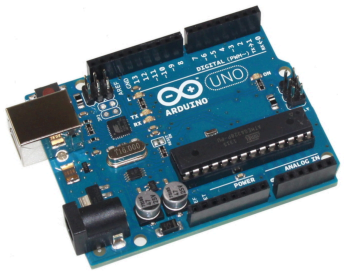
- Push buttons
- Twist knobs
- Light sensor
- Temperature sensor
- Accelerometer
- Infrared sensor
- CO2 sensor
- CO sensor
- Ultrasonic distance sensor
- Alcohol sensor
- Capacitive touch sensor
- Microphone
- Heart rate sensor
- … and lots more

Examples of actuators:

- LED
- Piezo speaker
- Pager vibration motor
- Step motor
- Servo motor
- LCD
- Heating pad
- … and lots more

Hardware

During this workshop you will learn how to use the following hardware:



Arduino & USB kabel

The Arduino board is a mini computer that you can program to control an almost endless amount of hardware.
The board is programmed with a USB cable. This cable is also used to power it, or it can be powered by batteries.

Breadboard

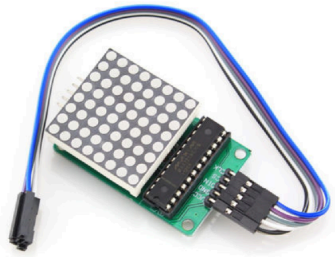
A breadboard is a useful tool to make an electronic circuit prototype without soldering. You simply insert the components into the holes to connect them together. Unfortunately, the connections are not sturdy enough to really use in practice. So if you are content with your circuit on your breadboard, it's important to transfer the circuit to a circuit board and solder it permanently.

Jumper Wires

Jumper wires are conductive. You use these to connect components on your breadboard and to your Arduino board.

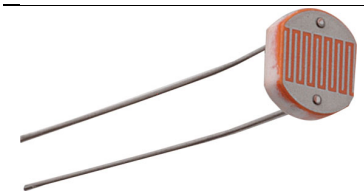
LED

LEDs light up when current runs through them. Make sure you connect the LED the correct way around. The lead on the flat side is the minus and usually goes to ground. With LEDs, the colour matters for how to connect it. For this manual we will use red, yellow or green LEDs. Blue and white are less suitable.



LED matrix

This tiny screen is comprised out of 64 LED's and can show basic images.



LDR (Light sensor)

A LDR allows current to flow freely if there is a lot of light, but resists the passage of current in the dark. This allows you to use an LDR as a light sensor.

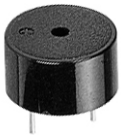
Potentiometer
A potentiometer (often abbreviated to pot-meter or just pot) is a rotary knob that has a variable resistance. The pot we use has a maximum resistance of 10k

---

Microbutton
This is a push button. It closes the circuit when you press it.

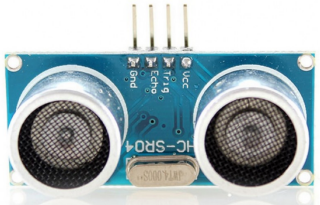
Resistor
A resistor restricts the flow of electric current. The value of the resistor is expressed in Ω (Ohm) or kΩ (kilo Ohm).
Resistors have coloured bands that indicate the value. You do not have to memorize those colour codes. http://www.weerstandcalculator.nl/ is a handy tool to decipher the colour codes.

Servomotor
A Servo is a motor that can turn very accurately. You can orient it to a certain position. For that reason, Servos are used, among other things, to steer remote controlled cars. They can never completely turn round. Other motors are suitable for this.



Piezo speaker
This speaker can, as you might have guessed, make noise.

NTC
This sensor is used to measure temperature. It's resistance decreases as temperature rises.



Ultrasonic sensor
This sensor measures distance by emitting an inaudible sound pulse and measuring the time it takes for the echo to reach the sensor again. Mimicking the way bats use high pitched sound to sense their environment.

Getting started on Mac-OS X

Follow these steps to get your Arduino working on Mac OS.

A. Go to http://arduino.cc/en/Main/Software. Download and install the current version of Arduino IDE.
B. Connect the Arduino board to your computer with the USB cable. One or more lights on the board should now turn on.
C. Open the Arduino IDE software.
D. In the menu navigate to Tools > Board and choose *Arduino Uno*, if not yet selected.
E. In the menu navigate to Tools > Port and choose */dev/tty.usbmodem****xxxx*** or */dev/cu.usbmodem****xxxx*** if not yet selected. The ***xxxx*** stands for a number that's could be different for every computer.

Once you have completed these steps, you are ready to work with the Arduino board. In *Exercise 1: Blink* you will test your board.

Getting started on Windows

Follow these steps to get your Arduino working on Windows.

A. Go to http://arduino.cc/en/Main/Software. Download and install the Arduino IDE.
B. Connect the Arduino board to your computer with the USB cable. One or more lights on the board should now go on. Windows will detect that a new piece of hardware is connected and will try to install it. Wait until the installation is done.
C. Open the Arduino IDE software.
D. In the menu navigate to Tools > Board and choose *Arduino Uno*, if not yet selected.
E. In the menu navigate to Tools > Port and choose *COM****X*** if not yet selected. The ***X*** stands for a number that's could be different for every computer.

Once you have completed these steps, you are ready to work with the Arduino board. In *Exercise 1: Blink* you will test your board.

The Arduino IDE Interface

The Arduino IDE consists of a text editor to write code, a bar with buttons for frequently used functions, a console and a number of menus. The development environment connects to the board to upload your program and to communicate with the board.

Interface

The upper part of the interface contains a number of buttons. This is what each button does.

Verify
Checks your code for errors without uploading your code.

Upload
Compiles and uploads the code to the board. Only when the code is error free.

New
Opens a new sketch.

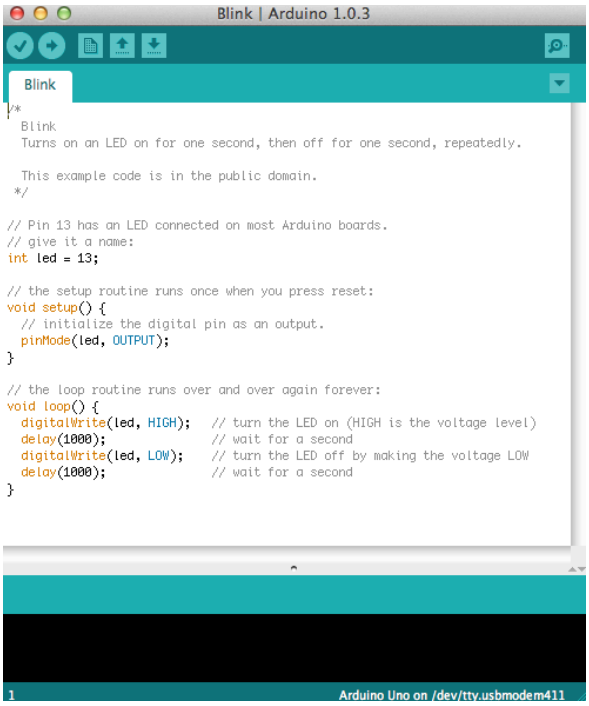
Open
Opens an existing sketch from your sketchbook.

Save
Saves your sketch

Serial Monitor
Opens the Serial Monitor.

Sketches

Software written with Arduino is called a sketch. Sketches are written in the text editor. Sketches are saved with the .ino file extension.



Console and message bar

The black area at the bottom of the screen is the console. It provides feedback during saving and exporting and displays errors made. Just above the console you see a green message bar. This too is used for providing feedback.

Exercise 1: Blink

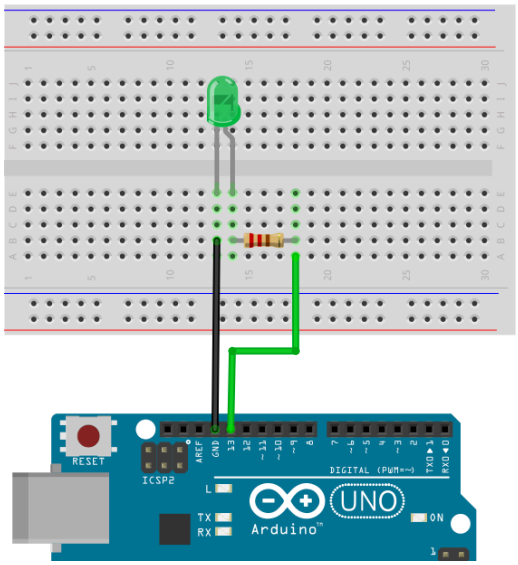
With this first exercise you will start by making a LED blink. We will use a built-in example for this.

A. Go to *File > Examples > 01. Basics* and open the *Blink* example.
B. Click *Verify* and wait until the program is done verifying.
   The message "done compiling" is shown in the message bar.
C. While the Arduino board is connected, click *Upload*.
   On the board the LEDs labelled TX and RX should start blinking to indicate that the board is communicating with the computer. After a while the message "done uploading" will appear in the message bar.
   And finally the LED labelled L should start blinking in 1 second cycles.

On Windows, uploading might fail because the wrong COM-port is selected. Selecting an other port in the Tools > Port menu might solve this problem.

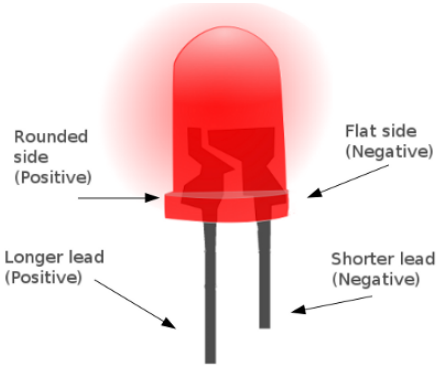
Exercise 2: Blink - Breadboard

In the previous exercise you have already seen that you can make a LED flash on the Arduino board. In this exercise you will connect an external LED using a breadboard and have it blink. Use the image on the right to connect your circuit.



A. Disconnect the Board from your computer. Then recreate the circuit shown in the image on your breadboard.
   You will need:
   - 1x 330Ω (Ohm) resistor. Colour code [orange orange brown gold] or [orange orange black black brown][1]
   - 1x LED (red, green or yellow)
   - Breadboard
   - Arduino board
   - Jumper wires

   Although it is not technically necessary, it's a good idea to use jumper wires with different colours[2].
   Pay attention to the orientation of the LED. The rounded side at the bottom of the LED (or the longer lead) should be on the side of Arduino pin 13. The flat side at the bottom of the LED (or the shorter lead) must go to GND (Ground).
   The orientation of the resistor does not matter in the circuit. Also it can be placed on either side of the LED in your circuit.
   Finally connect the board to your computer. It will power up and run the program you uploaded in exercise 1. The external LED should now also blink.

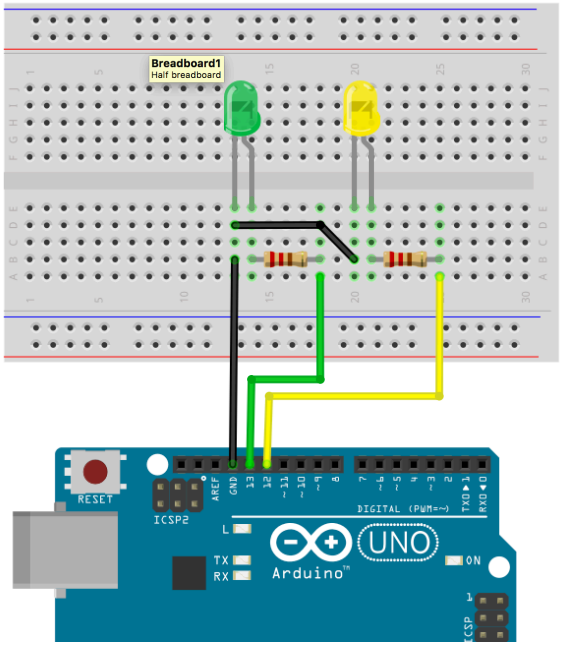


Change your code to the following:
B. The LED blinks slower
C. The LED blinks faster
D. The LED blinks by being off longer than it is on.

Remember to always verify the code before you upload it to the Arduino board.

[1] Any resistor betweeen 180Ω and 500Ω will work here.
[2] The colours of the jumper wires do not matter for the operation of your circuit. However, there are conventions that make it easier to understand your circuit. Black is used for ground (GND) and red (not used here) for the power supply (5V).

Exercise 3: Blink multiple LEDs



Connecting one LED may not be as exciting. But we can extend the example to test some more possibilities.
In the code for exercise 1 and 2 the variable used for the LED-pin has the name LED_BUILTIN. This is a kind of nickname for pin 13. So instead of the name LED_BUILTIN we could also use 13 in those places in the code. Or for that matter any other pin number.

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

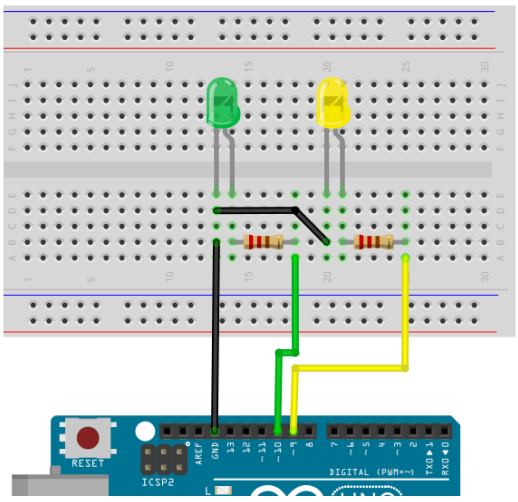
Extend your circuit with a second LED as shown in the image.
As you can see, a second LED (yellow), resistor (also 330Ω) and 2 jumper wires are added.
The second LED is connected to Arduino pin 12.
Change your code to the following:

A. Green and yellow blink at the same time.
B. Green and yellow will alternate.
C. Green blinks twice as fast as yellow.

Exercise 4: Fade LED – Analog Output

Instead of only turning LEDs on and off, we can also control the brightness of LEDs. The analogWrite() function allows us to set the brightness of an LED from 0 (completely off) to 255 (completely on) or anywhere in between.
Not all pins are suitable for this. It must be pins that are suitable for Pulse Width Modulation[3] (PWM). Normally a pin can either be turned on (HIGH) or off (LOW). A pin suitable for PWM can behave as if it is partially on. And by doing so, make it look like the LED is dimmed.
On the Arduino board pins 3, 5, 6, 9, 10 and 11 are suitable for PWM. On the board itself this is indicated by the ~ symbol next to the pin number.

A. We can almost use the same circuit from the previous exercise. The only thing that needs changing is the pins on the Arduino in order to support PWM. In the image you can see that here we chose pin 9 and 10.
Upload the following code to the board. If everything went well, you will see that the green LED alternates between on and off completely. And the yellow LED alternates between fully on and half on.



```
int greenLedPin = 10;
int yellowLedPin = 9;

void setup() {
  pinMode(greenLedPin, OUTPUT);
  pinMode(yellowLedPin, OUTPUT);
}

void loop() {
  analogWrite(greenLedPin, 255);  // turn on the green LED maximally
  analogWrite(yellowLedPin, 128); // turn on the yellow LED half way
  delay(1000);                    // wait one second
  analogWrite(greenLedPin, 0);    // turn the green LED off
  analogWrite(yellowLedPin, 255); // turn on the yellow LED maximally
  delay(1000);                    // wait one second
}
```

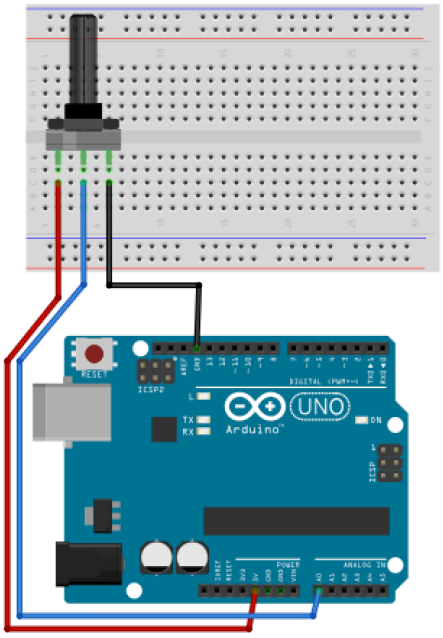
[3] Although the function `analogWrite()` is used, PWM is not truly analog. The pin pulses between HIGH and LOW at a very high frequency. The duration (or width) of the HIGH and LOW pulse determines the output voltage. For more info on PWM check out https://learn.sparkfun.com/tutorials/pulse-width-modulation/all

B. Change the code so that one of the LEDs slowly fades from completely off (0) to completely on (255). To do this, first empty the void loop() function. You can then use a for-loop. See the following code for an example.

```
for (int brightness=0; brightness <256; brightness++){
   analogWrite(greenLedPin, brightness);
   delay(10);
 }
```

C. Extend the code so that one LED slowly fades in and the other fades out at the same time.

Exercise 5: Potmeter value – Analog Input



You have seen in previous exercises that the Arduino board can have an analog output. In this exercise you will work with a rotary knob that gives an analog input to the Arduino.
You need the breadboard, a potentiometer (rotary knob) and jumper wires for this exercise. Make the circuit shown on the image on your breadboard.
Connect the wires of the potentiometer according to the image. The middle lug must be connected to the analog input pin A0 where the board will read the values. One of the two outer pins must be connected to the 5V and the other one goes to GND.
Now you will write the code to read out the value of the potentiometer. Add the following code to a new Arduino sketch and upload it to the Arduino board.

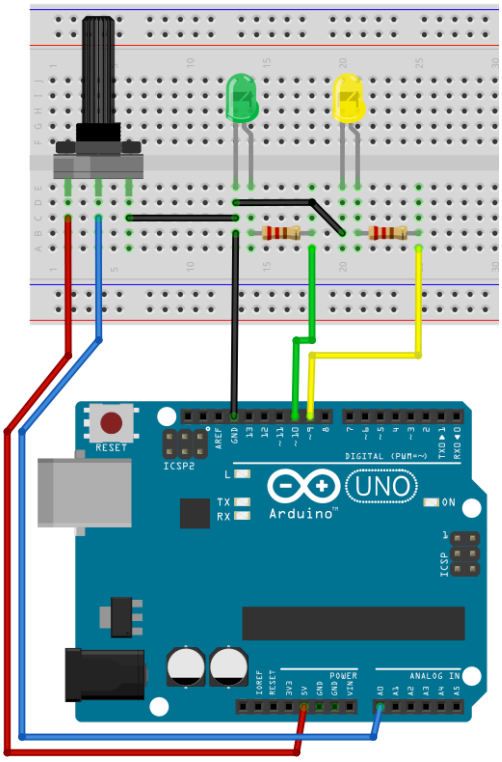
```
int sensorValue = 0;    // variable for sensor value
int sensorPin = A0;     // variable for sensor pin

void setup() {
  pinMode(sensorPin, INPUT);
  Serial.begin(9600);
}
void loop() {
  sensorValue = analogRead(sensorPin); // read the value/voltage on the sensor pin and
                                       // store that value in the variable sensorValue
  Serial.println(sensorValue);         // print out sensorValue to the Serial Monitor
  delay(200);                          // delay for 0.2 seconds
}
```

In the IDE open the Serial Monitor by clicking *Tools > Serial Monitor*. A new screen now opens; the Serial Monitor. It shows all messages that are sent by the Arduino board to the computer. And in this case, because of the program we just uploaded, the values of the potentiometer are shown. Make sure the value in the lower right corner is set to 9600 baud. Carefully turn the knob and you will see the number in the Serial Monitor change. The potmeter now works as a sensor. The potmeter's rotation is now measured and displayed as a number between 0 and 1023.
Another way to visualize the numbers sent over the Serial connection is the Serial Plotter. Open it through the Tools menu and see how it works.

Exercise 6: Controlling a LED with a potmeter

In this exercise we will use the input from the potentiometer from exercise 6 to fade the LEDs from exercise 4.

A. Make sure that your circuit looks like the picture on the right.



The centre lug of the potmeter is connected to pin A0. The LEDs are connected to pin 9 and 10. The point of this exercise is to learn how to use an analog input to control an analog output. You can use the code from exercise 6 as a basis for this exercise.

1. The outcome of exercise 6 was that the Arduino read a value between 0 and 1023, which was determined by rotating the potentiometer. We have seen in exercise 4 that a LED is controlled by a value from 0 to 255. This means that we have to change the code so that the sensor value gets scaled from a 0-1023 range to a 0-255 range.
   We can add the following calculation to the code after the sensorValue has been read:

```
sensorValue = (sensorValue/1023)*255;
```

2. For this calculation, the variable sensorValue must be a float (number with a decimal point) instead of an int (whole number). Change the data type of the variable sensorValue from int to float at the top of the file.

3. Now change the code so that the correct voltage is sent to the LEDs by doing the following steps.
   Add two new variables to contain the LED pins.

```
int greenLedPin = 10;
int yellowLedPin = 9;
```

   Add the following to the setup() section of the code:

```
pinMode(greenLedPin, OUTPUT);
pinMode(yellowLedPin, OUTPUT);
```

   In the loop() section add the following lines after you calculated the sensorValue.

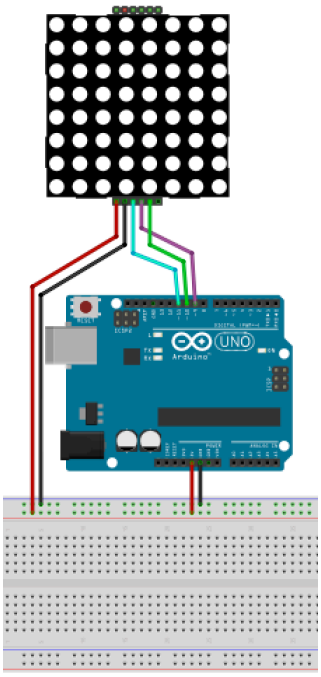
```
analogWrite(greenLedPin, sensorValue);
analogWrite(yellowLedPin, sensorValue);
```

4. Verify and upload your code to the board. If you now rotate the potmeter, the LEDs should change their brightness. You might notice that the fading isn't really fluently. To remedy this, shorten the delay. A good value for the delay is 20.

B. On the Arduino.cc website you will find a list of all built-in functions of Arduino under Resources > Reference. One of them is the function map(). Look up how this function works. Change the code of exercise A so that you incorporate the function map().

C. Change the code so that one LED fades in while the other fades out and vice versa.

Exercise 7: LED Matrix



In this exercise you wil use the 8*8 LED Matrix that needs a special library. To install the library, click *Sketch > Include Library > Manage Libraries*. In the menu use the search bar to find the *LedControl* library by *Eberhard Fahle* and install it.
Hook up the matrix to your to your Arduino board. The figure on the right of this text is an example of a similar circuit, but uses a different type led matrix and different pins! Make sure you connect your led matrix breakout board according to the following table:

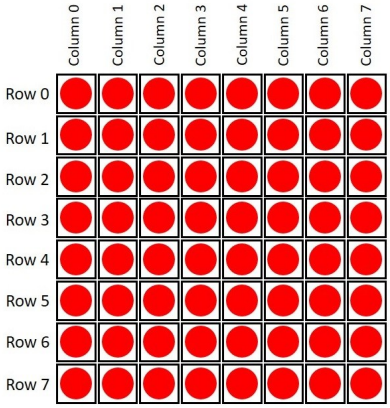
LED matrix pin	Wiring to Arduino Board
GND	GND
VCC	5V
DIN	Pin 12
CS	Pin 10
CLK	Pin 11

The easiest way to display something on the dot matrix is by using the functions `setLed()`, `setRow()` or `setColumn()`. These functions allow you to control one single led, one row or one column at a time.

```
setLed(addr, row, col, state);
```

addr is the address of your matrix, for example, if you have just 1 matrix, the int addr will be zero.
row is the row where the led is located
col is the column where the led is located
state (true or false)
- It's true or 1 if you want to turn the led on
- It's false or 0 if you want to switch it off

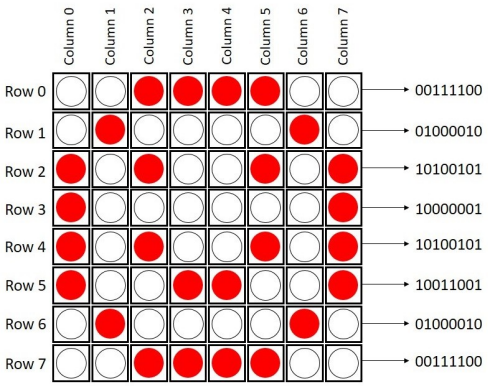
```
setRow(addr, row, valueArray);
setCol(addr, column, valueArray);
```



This matrix has 8 columns and 8 rows. Each one is indexed from 0 to 7. Here's a figure for better understanding:

Smiley LED Matrix

A. If you want to display something in the matrix, you just need to know if in a determined row or column, the LEDs that are on or off. For example, if you want to display a happy face, this image shows what you need to do:

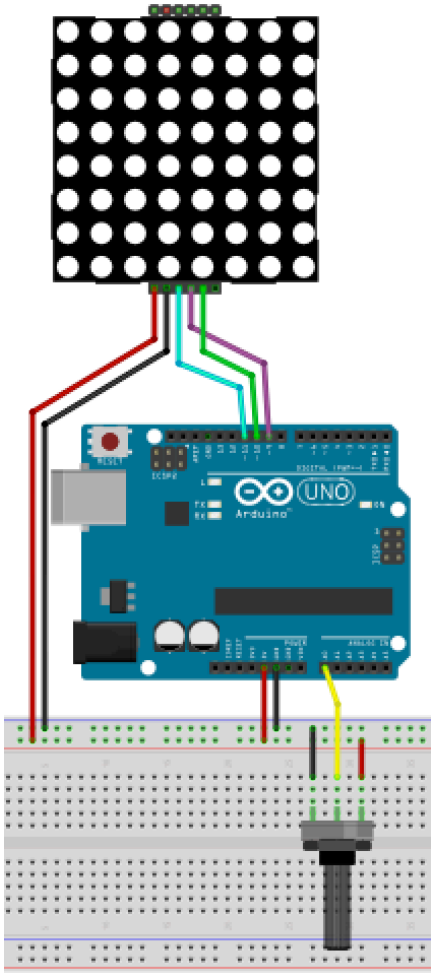


Copy, upload and execute the code on the next

page to your Arduino board. A smiley face and a frowney face should alternate every second.

```
#include "LedControl.h"

/*
 DIN connects to pin 12
 CLK connects to pin 11
 CS connects to pin 10
*/
LedControl lc = LedControl(12,11,10,1); // create a new LedControl object

void setup() {
  lc.shutdown(0,false); // Turn matrix on, no power saving
  lc.setIntensity(0,8); // Set brightness to a medium value
  lc.clearDisplay(0);   // Clear the display
}

void loop(){
  lc.setRow(0,0,B00111100); // set the first row of the matrix
  lc.setRow(0,1,B01000010); // set the second row of the matrix
  lc.setRow(0,2,B10100101); // etc.
  lc.setRow(0,3,B10000001);
  lc.setRow(0,4,B10100101);
  lc.setRow(0,5,B10011001);
  lc.setRow(0,6,B01000010);
  lc.setRow(0,7,B00111100);
  delay(1000);

  lc.setRow(0,0,B00111100);
  lc.setRow(0,1,B01000010);
  lc.setRow(0,2,B10100101);
  lc.setRow(0,3,B10000001);
  lc.setRow(0,4,B10011001);
  lc.setRow(0,5,B10100101);
  lc.setRow(0,6,B01000010);
  lc.setRow(0,7,B00111100);
  delay(1000);
}
```

B. Expand the circuit to include a potentio meter like in this image.
Copy, upload and execute the code the code below to your Arduino board. When you rotate the pot, the matrix should light up accordingly.

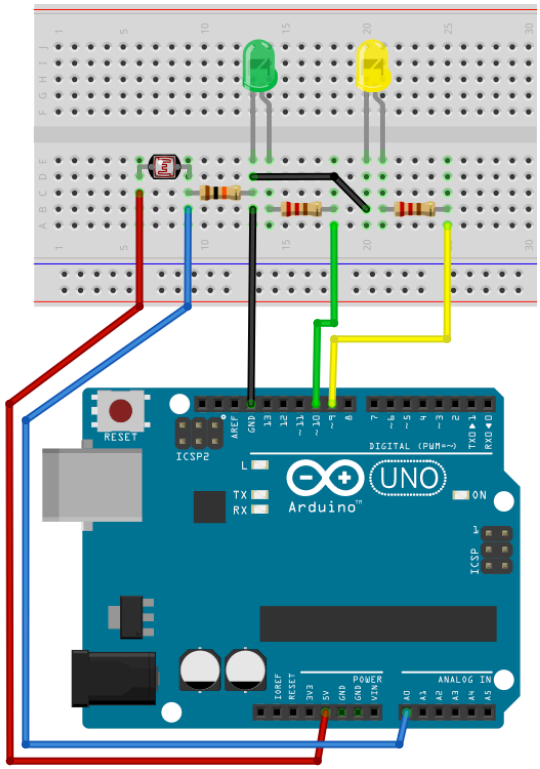


```
#include "LedControl.h"

/*
 DIN connects to pin 12
 CLK connects to pin 11
 CS connects to pin 10
*/
LedControl lc = LedControl(12,11,10,1);

void setup() {
  lc.shutdown(0,false); // Turn matrix on, no power saving
  lc.setIntensity(0,8); // Set brightness to a medium value
  lc.clearDisplay(0);   // Clear the display

  Serial.begin(9600);
}

void loop(){
  int sensorValue = analogRead(A0); // read sensor, 0-1023
  Serial.println(sensorValue);
  sensorValue = map(sensorValue, 0, 1000, 0, 7);
         // remap the value 0-7. (1000 eliminates noise)

  lc.clearDisplay(0);                    // Clear the display
  lc.setRow(0, sensorValue, B11111111);  // Turn one line on

  delay(20);
}
```

C. Change the code so that the value of the potentio meter is visualised with the matrix in an other interesting way.

Exercise 8: Control the LED with light

In exercise 6 you dimmed a LED with a potentiometer. In this exercise you will control the LED with a different kind of sensor. There are many more sensors that you can use with the Arduino. One is an LDR (Light Dependent Resistor), a variable resistor that measures the amount of light. This is similar to the sensor that is used to turn street lights on when it gets dark outside.



A. You will use this sensor to dim the LEDs. Build the circuit in the image shown. Connect one leg of the LDR to 5V. And the other leg on pin A0. On the leg that is connected to A0, you also connect a 10kΩ resistor [brown black orange gold] or [brown black black red brown]. The other side of the resistor is connected to GND. It does not matter which way around you orient the LDR or resistor.

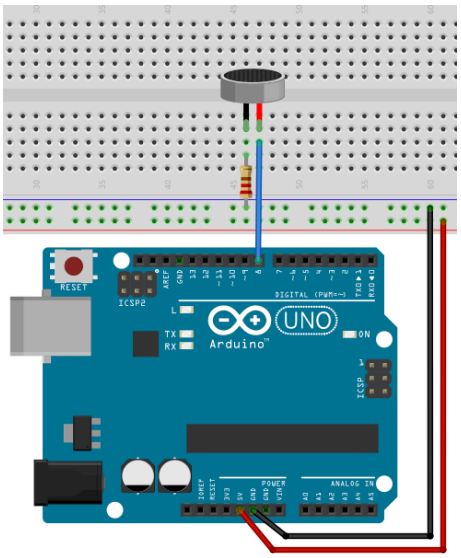
1. Use the code you wrote in exercise 6A. Comment out the line you wrote to scale the sensor value. Do this by adding // at the beginning of the line. This will temporarily disable that line.

```
// sensorValue = (sensorValue/1023)*255;
```

2. Verify the code and upload to the board.
3. Open the Serial Monitor. What you see now is the amount of light that the LDR perceives expressed in a number. Under ideal circumstances this range is between 0 and 1023. But in practice, the measured range is a lot smaller. If you hold you hand over the LDR, you should see a drop in this value. Write down the highest (unobscured/uncovered) and lowest (obscured/covered) value. This the actual range the sensor has under these lighting conditions.
4. Uncomment the line you commented (in step 1) and change it to rescale the sensor value in the proper way. The object here is to rescale the actual range you measured in step 3 to the 0-255 range the LED's accept. (You could also use the map() function if this makes things easier)
5. Verify the code and upload to the board. If all went well, the LEDs should now fade according to the amount of light that the LDR perceives.

You might notice the LEDs flickering when sensor value is out of range. The light will completely turn on when the sensor value is a negative number. If you want to remedy this you could use the constrain() function to make sure the number is never lower than 0. Look up the constrain() function in the reference and implement it in your code.

Exercise 9: Sound



When you connect a speaker, the Arduino board can produce sound. This happens with a so-called Piezo speaker (looks like a small black cylinder) that consists of a plate of two different metals that can vibrate and thusly produce sound.
Connect the circuit as shown in the image. When connecting, use a 330Ω resistor [orange orange brown gold] or [orange orange black black brown]. The speaker has a polarity indicated by a + sign next to one of the lugs. The + goes towards the Arduino's pin 8. The resistor has no polarity, so it does not matter how you orient it.
You can test the speaker with code provided below. In the code you can see that the function tone() is used. This function actually makes the tone. The first parameter states which pin the speaker is connected to. The second parameter determines the frequency (in hertz). The final parameter states the duration (in milliseconds). The last parameter is optional and when it is omitted, the tone will ring indefinitely until a new tone starts or the tone is stopped with the noTone() function.

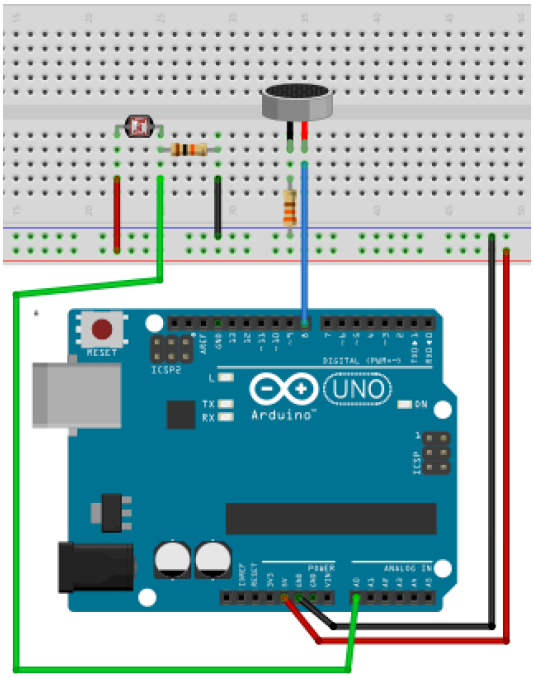
```
int speakerPin = 8;

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

void loop() {
  tone(speakerPin, 262, 200);  // plays a tone of 262Hz for 200ms
  delay(250);                  // wait 250 ms

  tone(speakerPin, 294, 200);  // plays a tone of 294Hz for 200ms
  delay(250);                  // wait 250 ms

  tone(speakerPin, 330, 200);  // plays a tone of 330Hz for 200ms
  delay(250);                  // wait 250 ms

  tone(speakerPin, 262, 200);  // plays a tone of 262Hz for 200ms
  delay(500);                  // wait 500 ms
}
```

Note	Frequency (Hz)
C_4	262
$C^{\#}_4/D^{b}_4$	277
D_4	294
$D^{\#}_4/E^{b}_4$	311
E_4	330
F_4	349
$F^{\#}_4/G^{b}_4$	370
G_4	392
$G^{\#}_4/A^{b}_4$	415
A_4	440
$A^{\#}_4/B^{b}_4$	466
B_4	493
C_5	523

A. Upload and test the code.
B. Change and expand the code so that it plays a creative melody. You can use the diagram to choose your frequencies.
C. Make a siren. For a siren the frequency always increases until a certain upper frequency is reached and then the frequency is decreased again. Use a loop to achieve this. You can choose your own frequency range, but please keep it

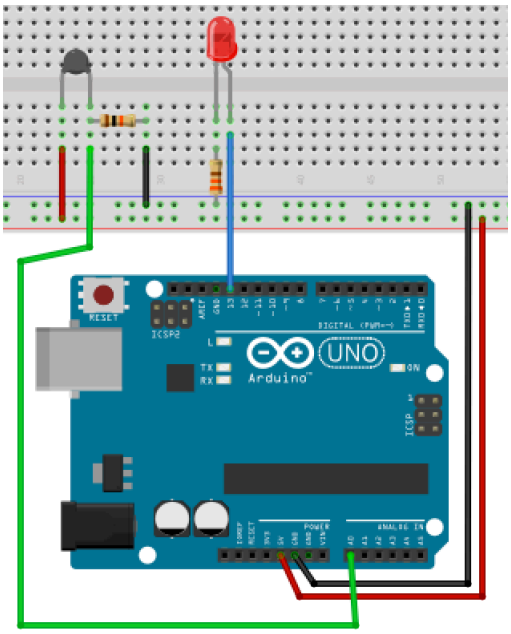
somewhere between 100Hz en 2000Hz. If your siren creaks, try the `tone()` function without the third parameter.

D. For this final sound exercise, you will make a musical instrument resembling a Theremin[4]. The purpose of this exercise is to control the pitch of the sound by moving your hand over the LDR.
    1. Add the LDR to the circuit as shown in this image.
    2. Read the value from the LDR like you did in exercise 8.
    3. Remap the sensor value to a manageable frequency range between 100Hz and 2000Hz.
    4. Create a tone. Use the sensor value as the tone's pitch.
    5. Test your Theremin by waving your hand over it.



[4] A Theremin is a retro futuristic musical instrument that might be best known for the opening theme for the Star Trek series from the sixties. It actually worked by manipulating the electro magnetic field around two antennas to control the pitch and volume of the sound. https://en.wikipedia.org/wiki/Theremin. Watch a performance of the Star Trek theme here https://www.youtube.com/watch?v=x0NVb25p1oU. Or watch Leon Theremin play his own creation here https://www.youtube.com/watch?v=w5qf9O6c20o

Exercise 10: Sensing temperature



An NTC is used to measure temperature. Just like a LDR it is a variable resistor, but the NTC is sensitive to temperature. The NTC in your kit can measure temperatures ranging from -30C ~ +150C.

A. Build the circuit in the image shown. The NTC in your kit might not be black, but green. And probably has 103 written on it.
Connect one leg of the NTC to 5V. And the other leg on pin A0. On the leg that is connected to A0, you also connect a 10kΩ resistor [brown black orange gold] or [brown black black red brown]. The other side of the resistor is connected to GND. It does not matter which way around you orient the NTC or resistor.

Connecting the LED to pin 13 is practically the same as in exercise 2. The resistor value here is 330Ω [orange orange brown gold] or [orange orange black black brown]. Don't forget about the LED's polarity.

B. Upload the following code and open the Serial monitor.

```
int ledPin = 13;
int sensorPin = A0;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(sensorPin, INPUT);
  Serial.begin(9600);
}

void loop() {
  int sensorValue = analogRead(sensorPin); // read sensorpin
  Serial.println(sensorValue);             // print to serial monitor

  delay(10);                               // wait
}
```

The value you see in your serial monitor represents to the ambient temperature. Write down what that value approximately is.

C. You are now going to warm up the NTC by using your body heat. Carefully pinch the NTC making sure you don't touch the leads to much (or you'll measure the capacity of your skin instead of the temperature). You should see the value in the Serial monitor rise. Once the value stabilises, write down the approximate value.

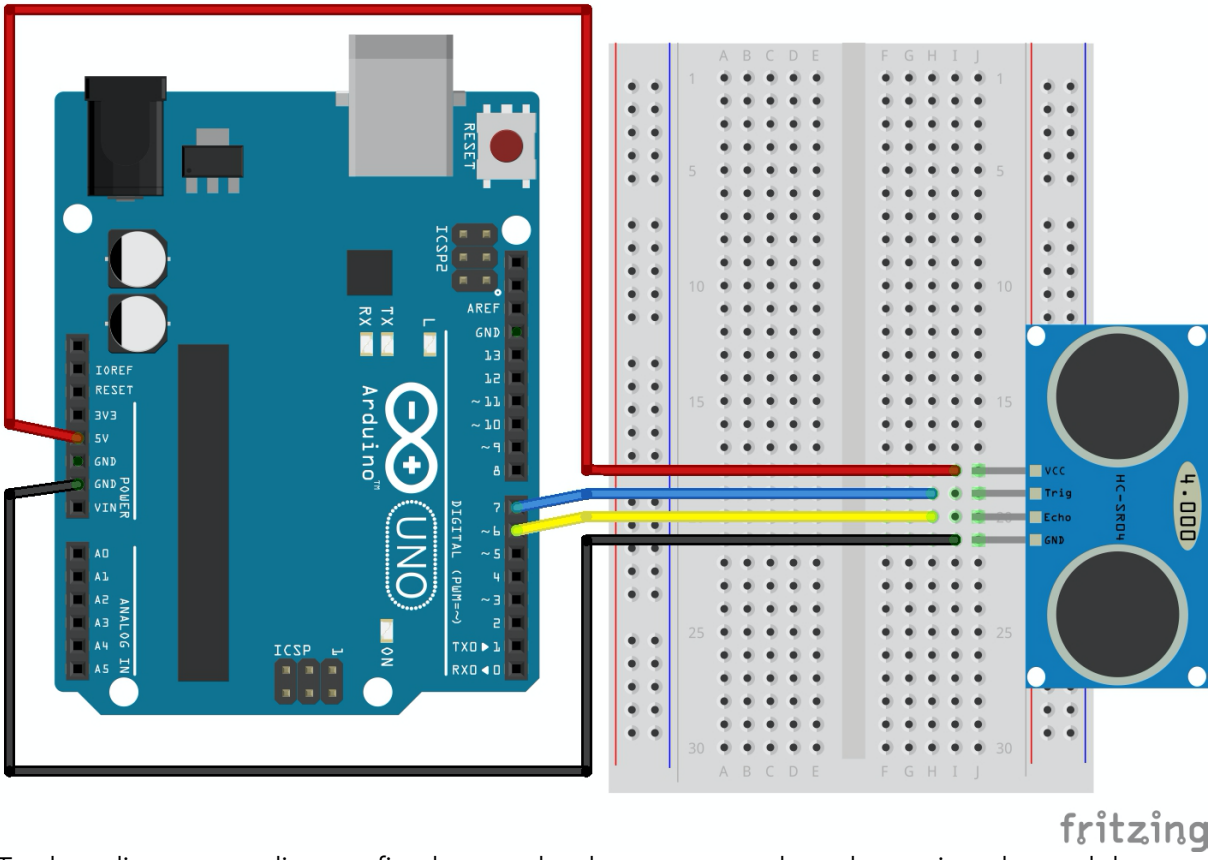
D. You should now have written down two numbers. The value exactly in the middle of these two numbers will be your threshold value. Add an if-statement to the code to turn on the LED when the sensor value exceeds the threshold value. The LED should turn off when the sensor value is lower then the threshold.

Exercise 11: Distance

For measuring distance, we will use an Ultrasonic Proximity Sensor. These sensors work by transmitting a pulse of ultrasonic sound (sounds that are higher in frequency than humans are capable of hearing) and then listening for the return echo of this sound as it bounces of objects in front of the sensor. Ultrasonic sensors are commonly used in parking assist for cars.[5]

The sensor we will be using is a HC-SR04, with 4 pins: VCC, TRIG, ECHO, and GND. VCC connects to the Arduino 5V supply, GND connects to one of the Arduino GND pins. TRIG and ECHO are used to communicate with the sensor.

A. Build the circuit in the image below. TRIG is connected to pin 7 and ECHO to pin 6.



To do a distance reading we first have to let the sensor send an ultrasonic pulse and then listen for the echo. We will do this by sending a short signal (pulse) to the TRIG pin, the sensor will then send the ultrasonic pulse. Then we wait for this pulse to return, with the function pulseIn(). The Arduino will wait and count (in microseconds) the time it takes for the ECHO pin to go HIGH and then LOW again. That duration is the time it took for the ultrasonic pulse to travel from the sensor to an object and back again. The longer the duration, the further away the object is.

[5] This exercise was inspired by: https://rmit.instructure.com/courses/47702/pages/ultrasonic-proximity-sensor?module_item_id=1241442

B. Copy the code below and upload it to your Arduino and run the sketch. (Open your Serial monitor to view the sensor readings)

The Arduino will send a signal to the sensor, the sensor will send an ultrasonic pulse, the Arduino reads the measured duration from the ECHO pin, recalculate it to cm and print this value to the serial monitor.

The calculation from duration to distance is done using the knowledge that the speed of sound in air is 340 m/s or 29 microseconds per centimetre. The sound travels out and back so to get the distance to the object, this value is divided by 2.

```
const int echoPin = 6;
const int trigPin = 7;

void setup() {
 pinMode(echoPin, INPUT);
 pinMode(trigPin, OUTPUT);

 Serial.begin(9600);
}

void loop() {
 // send a pulse
 digitalWrite(trigPin, LOW);
 delayMicroseconds(2);
 digitalWrite(trigPin, HIGH);
 delayMicroseconds(5);
 digitalWrite(trigPin, LOW);

// wait for the echo
 long duration = pulseIn(echoPin, HIGH);

// convert the time into a distance, the speed of sound is 29 microseconds per
centimeter,
//the pulse traveled forth and back, so we divided the distance by 2
 int cm = duration / 29 / 2;

Serial.print(cm);


delay(100);
}
```

C. Extend your circuit so it will have a yellow LED (with the corresponding resistor), connected to pin 8. Like you did before in exercise 2. Add the following line at the top of your code:

```
int ledPin = 8;
```

Add the following line to the setup:

```
pinMode(ledPin, OUTPUT);
```

Add the following lines to your loop, below the lines that print the distance to the serial monitor.Run your code. The yellow LED should start blinking when something is within 50 cm from the sensor.

```
//if the measured distance is lower than 50 cm, turn on LED
if(cm<50){
 digitalWrite(ledPin, HIGH);
}

//else turn off LED
else{
  digitalWrite(ledPin, LOW);
}
```

Social Distance Indicator

D. Make a social distance indicator. Create a traffic light (with a red, yellow and green LED) that indicates how far away something (or someone!) is standing.

Measured distance	LED on
<140cm	Red
>140cm & <160cm	Yellow
>160cm	Green

The green light indicates that someone is at a safe (160 cm or further) distance. The yellow light should give a warning about being within 10 cm of the 150cm threshold. A red light indicates there's too little distance.

Exercise 12: Arduino and Processing

Until this moment you have seen Arduino work standalone. Meaning that after the program is uploaded, the program runs by itself on the Arduino board. The USB cable then only acts as a power supply.
You could even replace the USB cable for a USB power bank or a 9V battery to make your circuit work completely wireless.
For this exercise though, we will use the Arduino board as an input device for our computer by creating a connection between Arduino and Processing. Processing is a programming environment that is great for prototyping interactive experiences. We will also be using Processing in the next workshop on Computer Vision. Processing and Arduino work well together. You can use them together to make real world electronic input-devices for a digital environment or vice versa.
In this exercise the input from the previous exercise is passed through the Arduino to a Processing sketch.

A. Download and install Processing: https://processing.org/download/
B. (Re)build the circuit of exercise 5 or 10.
C. Copy, paste and upload the Arduino code below onto your Arduino
D. Copy and paste the Processing code in a new processing sketch. Change the serial port where indicated in the Processing code and then run it by pressing the play button in Processing.

Make sure the Arduino Serial Monitor isn't open before you start the processing code.
You should now see a circle on the screen. The size of the circle is determined by input of the sensor.

E. Now try to read the Processing code. We do not expect you to understand each line at first glance, but you might recognise some functions. Change the Processing code to use the sensor value in another interesting way. For example, rotate a rectangle, move a triangle across the screen or let text appear.

Arduino Code:

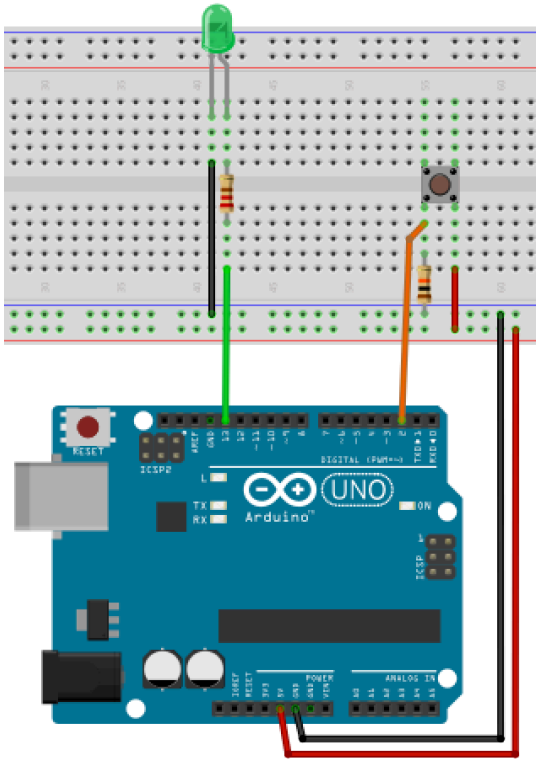
```
float sensorValue = 0;           // variable for sensor value
int sensorPin = A0;              // variable for sensor pin
int greenLedPin = 10;            // variable for green LED pin

void setup() {
  Serial.begin(9600);            // Start the Serial connection at a
                                 // speed of 9600 bps
  pinMode(sensorPin, INPUT);     // Input pin for potmeter or LDR
  pinMode(greenLedPin, OUTPUT);  // Output pin for LED
}

void loop() {
  sensorValue = analogRead(sensorPin);  // Read the value/current on the sensor pin and
                                        // store that value in the variable sensorValue
  sensorValue = (sensorValue/1023)*255; // Rescale the sensor's value.
```

```
                                    // Change the values 1023 and 255 to calibrate
the sensor.
  analogWrite(greenLedPin, sensorValue);// Send power to LED
  Serial.println(sensorValue);      // Print the sensorValue to the serial
                                    // connection
  delay(100);                       // Wait 0.1 seconds
}
```

Processing Code:

```
import processing.serial.*;

Serial myPort;
String sensorReading="";

void setup() {
  size(400, 400);
  myPort = new Serial(this, Serial.list()[5], 9600); // instead of 5, choose what ever
serial port the Arduino connects to
  myPort.bufferUntil('\n');
}

void draw() {
  background(255);
  fill(0);
  text("Sensor Reading: " + sensorReading, 20, 20);
  ellipse(width/2, height/2, float(sensorReading), float(sensorReading));
}

void serialEvent (Serial myPort) {
  sensorReading = myPort.readStringUntil('\n');
}
```

Exercise 13: Buttons



We've so far used a variety of sensors. The simplest sensor is a button. A button functions by closing a circuit and this can be sensed by the Arduino. In this exercise you will practice using buttons in different ways.

One button

The link below is for the example Button on the Arduino website. We will use this example as the basis for this exercise.

http://arduino.cc/en/Tutorial/Button

In the image on the site, you notice that there no LED is used. Therefore, use the circuit shown here.

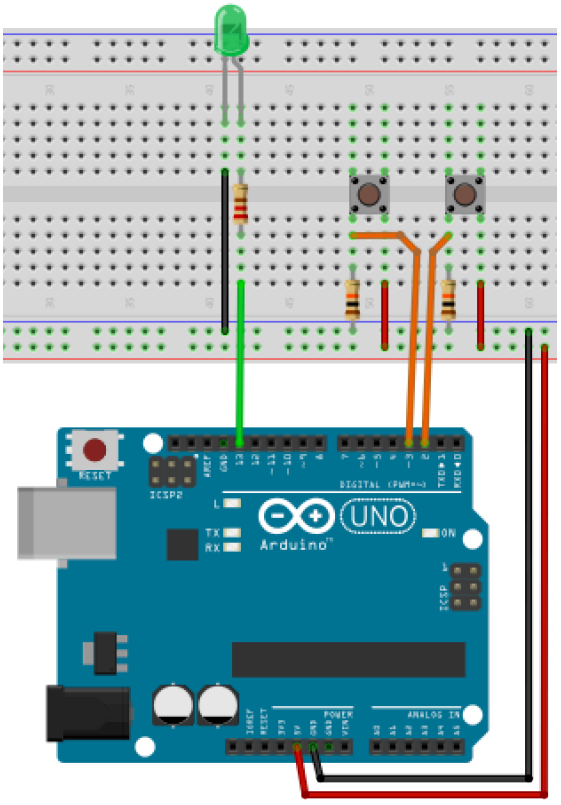
Make sure the LED is oriented the correct way. The two resistors have different values. The resistor closest LED is 330Ω [orange orange brown gold] or [orange orange black black brown]. And the resistor closest to the button is 10kΩ [brown black orange gold] or [brown black black red brown].

A. Upload the code from the example. You will notice the LED lights up when you press the button.
B. Change the code to do the exact opposite. The LED should turn off while the button is pressed.
C. The code in the example can be substantially shortened. The variable `buttonState` isn't per sé necessary. Shorten the code.

Two buttons

Two buttons are more fun than one button. Add a second second button as shown in the image. Now you can use the two buttons to interact with. Write 3 programs that work with this circuit.

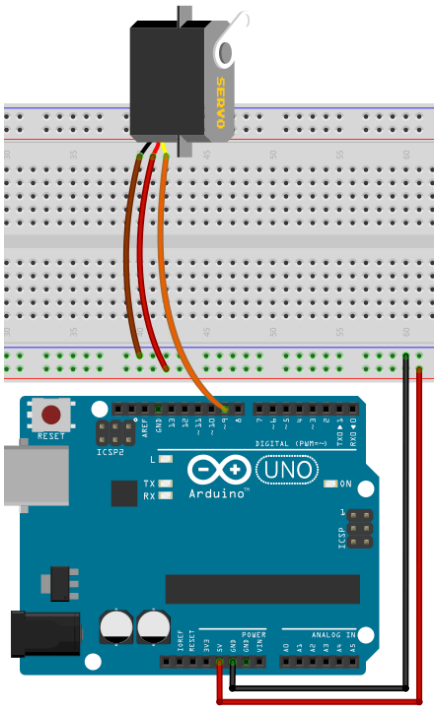
D. The LED lights up when both buttons are pressed simultaneously.
E. The LED lights up when one button of both buttons are pressed.



You can use the code below as a starting point for these programs.

```
int ledPin =  13;
int buttonPinL = 3;
int buttonPinR = 2;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPinL, INPUT);
  pinMode(buttonPinR, INPUT);
}

void loop(){

  ........

}
```

Exercise 14: Servomotor



Servomotors are motors that turn to a precise position. They are applicable for many purposes, but the most striking application is that of remote controlled vehicles. For example, they are used to steer the wheels of a remote controlled car, or to turn the helm of a remote controlled boat. Another application is to run a security camera to focus on something specific. Most servos can not rotate fully. Usually they have a reach of ± 180°. In this exercise you will control a servo to do a pre-programmed movement.

ATTENTION: NEVER TURN THE SERVO FORCEFULLY WITH YOUR HANDS. THIS DESTROYS THE INTERNAL MECHANISM.

Connecting the servo

The servo has three wires that connect to your Arduino board. Connect the servo as shown in the image. Pay attention to the colour wire that come out of the servo. Brown goes to GND, red goes to 5V and orange goes to pin 9. The servo is controlled by a special code library. That library can be imported by the following line:

```
#include <Servo.h>
```

A. Use the following code and upload it to the board.

```
#include <Servo.h>
Servo myServo;                // Declare a servo object for us to control
int servoPin = 9;             // Var for the pin the servo connects to
int pos = 0;                  // Var to keep track of the servo's position

void setup() {
  myServo.attach(servoPin);  // Tell the servo to what pin it's connected to
}

void loop() {
  for(pos = 0; pos < 160; pos += 1){ // Loop, pos is added to (from 0 to 160)
    myServo.write(pos);              // Turn the servo to the position in pos
    delay(15);                       // Wait 15ms
  }
  for(pos = 160; pos >= 1; pos -= 1){// Loop, pos is deducted from (from 160 to 1)
    myServo.write(pos);              // Turn the servo to the position in pos
    delay(15);                       // Wait 15ms
  }
}
```

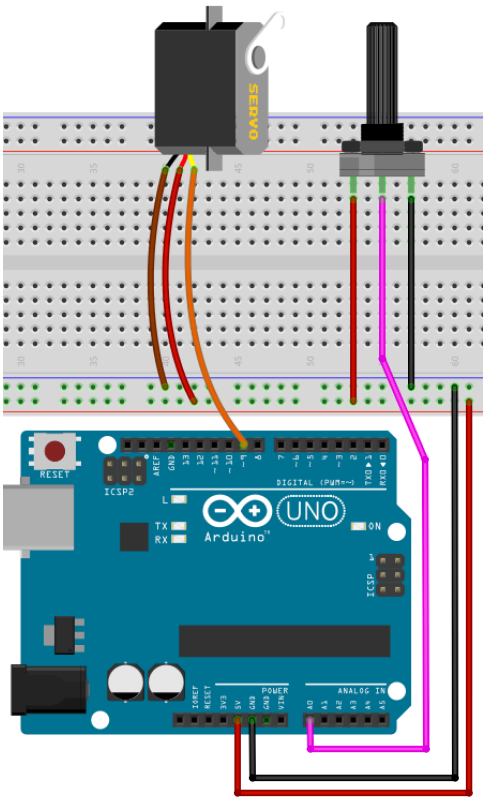
If all went well, your servo should now move from left to right and back again.

B. Change the code in a way that the servo will move in a more interesting way. Try to make the servo move in a funny, rhythmical way. (or teach it how to dance)

Servo with knob

You have now seen that you can turn a servo to a certain position. You can use that and combine it with the previous knowledge you have about potentiometers and buttons.

C. Build the circuit shown in the image. Write the code that lets you move the servo in accordance with the potmeter. Remember that the potmeter will be read as a value in the range 0 - 1023. This range should be scaled to the range 0 - 160 before sending the position to the servo.

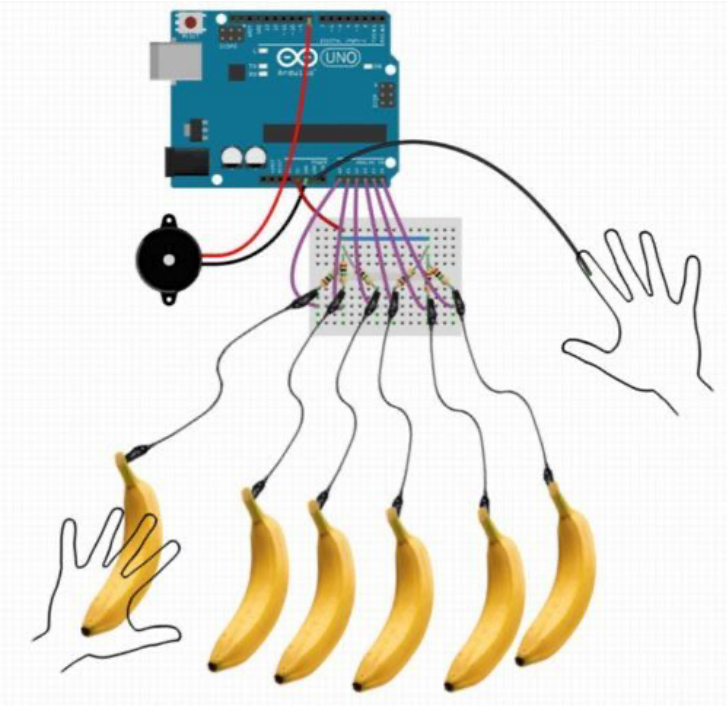


If all else fails, take a look at http://arduino.cc/en/Tutorial/Knob for tips.

Exercise 15: Capacitive Touch

In this exercise you will be making your own sensor that detects touch.

When making a capacative sensor with Arduino you will have an output that transmits a pulse, and an input which receives the pulse and compares it to the transmitted pulse. When you put your finger on or near the sensor it creates a delay in the pulse, and this delay is recalculated by the CapSense library and generates a value you can use.



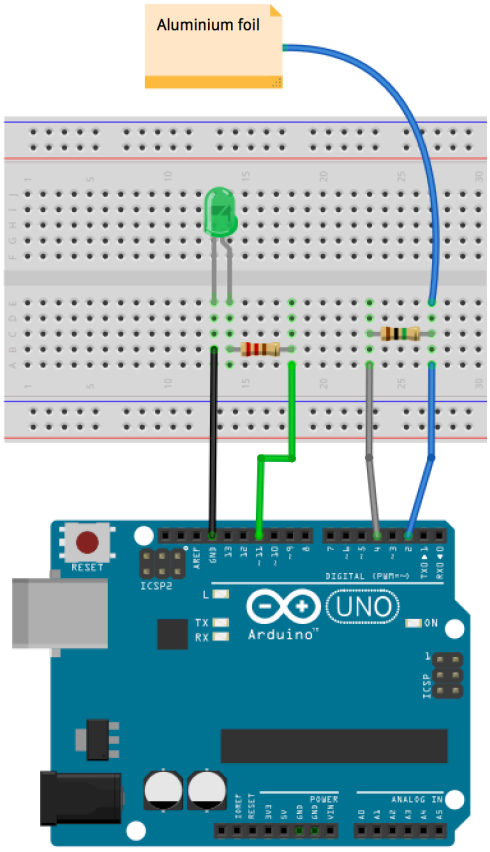
The capacitiveSensor library turns two or more Arduino pins into a capacitive sensor, which can sense the electrical capacitance of the human body. All the sensor setup requires is a 1MΩ resistor [brown black green gold] or [brown black black yellow brown] and a piece of wire and a piece of aluminium foil (or something else that conducts electricity) on the end. At its most sensitive form, the sensor will start to sense a hand or body inches away from the sensor.

Capacitive sensing may be used in any place where low to no force human touch sensing is desirable. An Arduino and the library can be used to sense human touch through more than a quarter of an inch of plastic, wood, ceramic or other insulating material (not any kind of metal though), enabling the sensor to be completely visually concealed.
A capacitive sensor covered with paper or other insulator also acts as fairly good (human touch) pressure sensor with an approximately logarithmic response. In this regard it may surpass force sensing resistors in some applications.[6]

IMPORTANT!
The grounding of the Arduino board is very important in capacitive sensing. The board needs to have some connection to ground. Capacitive sensing has some quirks with laptops unconnected to mains power. The laptop itself tends to become sensitive and bringing a hand near the laptop will change the returned values. Connecting the charger to the laptop will usually be enough to get things working correctly. Connecting the Arduino ground to an earth ground (for example, a radiator pipe) could be another solution.
Connect your computer/laptop to the power outlet!

[6] https://playground.arduino.cc/Main/CapacitiveSensor/
[7] https://www.instructables.com/id/Turn-a-pencil-drawing-into-a-capacitive-sensor-for/

Capacitive sensor



A. Build the circuit shown in the image. The LED is connected to pin 11, the capacitive touch is connected to pins 2 and 4. The resistor is 1MΩ (1 mega-ohm) [brown black green gold] or [brown black black yellow brown].
The blue wire (at the top of the image) goes nowhere, that is your sensor.

1. For this exercise you need to install a specific library.
   Go to 'Sketch' > 'Include Library' > 'Manage Libraries' and search for and install the library 'CapacitiveSensor' by Paul Bagder and Paul Stoffregen.

2. Copy, paste and upload the code below

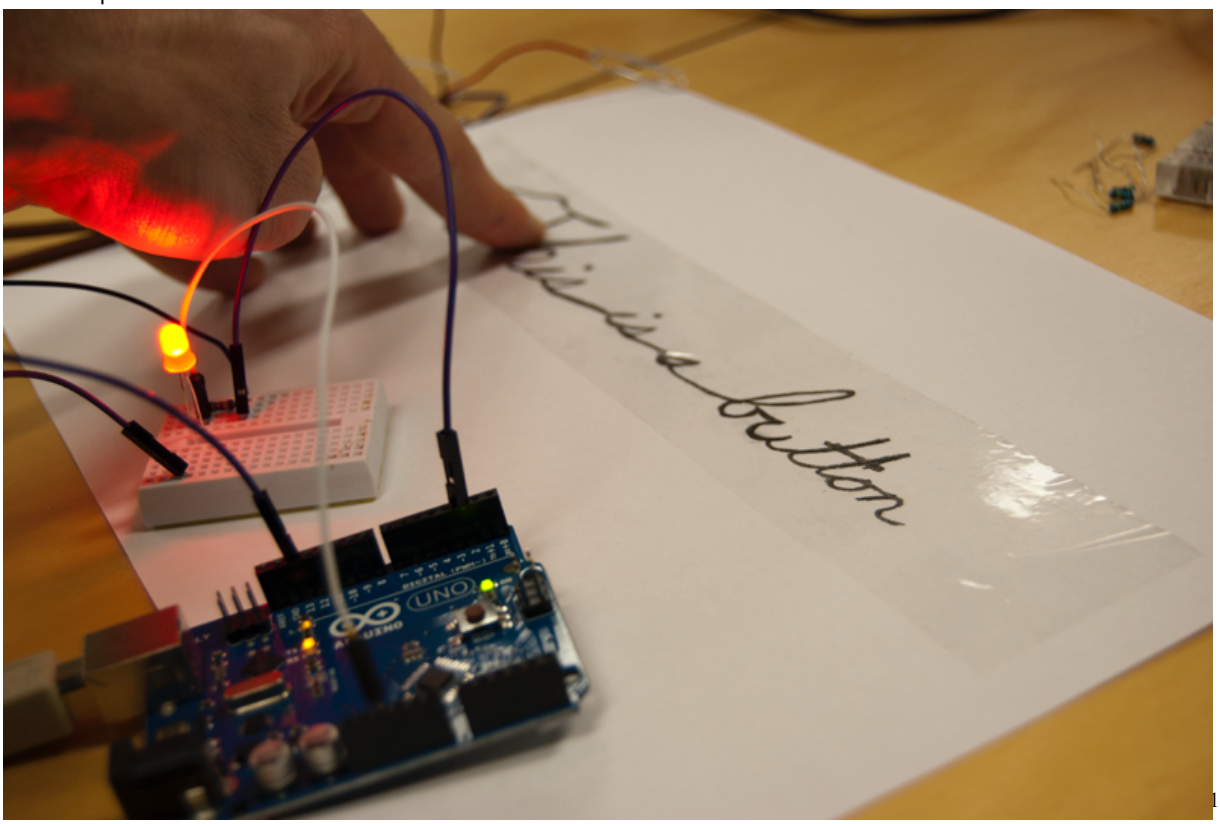
```
#include <CapacitiveSensor.h>

CapacitiveSensor touchSwitch = CapacitiveSensor(4, 2);

int LEDpin = 11;

void setup(){
    pinMode(LEDpin, OUTPUT);
    Serial.begin(9600);
}

void loop(){
    long val = touchSwitch.capacitiveSensor(100);

    if (val > 100) {
        digitalWrite(LEDpin, HIGH);
    } else {
        digitalWrite(LEDpin, LOW);
    }

    Serial.println(val); // print sensor output
}
```

3. Open your serial monitor or serial plotter and see what happens when you touch (or come near) the bare end of your sensor wire.

   The value should increase and the LED turns on/off according to the threshold value.
   The threshold value is 100 in the example code, change this if your sensor is too sensitive or not sensitive enough.

Draw a sensor

B. Basically, you can attach anything that's conductive to your sensor wire to make a sensor out of it.In this exercise we'll be drawing something on paper and use that for input.



1. Take a piece of paper and a soft (grey) pencil and draw something (see the picture above for an example).
   You'll want to follow these rules:
   - Make a big filled area at the edge of the paper - this is where you'll connect your bare sensor wire to the drawing via paperclip or tape.
   - Make sure all parts of your drawing are connected – you can only read from parts that are touching.
   - Re-trace over your lines at least once or twice - you'll want to get a nice, thick layer of graphite on the paper.

2. Change your code into the code below. The brightness of the LED will indicate the pressure of touching the sensor.
   Touch your drawing, the brightness of the LED will correspond with the pressure of your touch

```
#include <CapacitiveSensor.h>
CapacitiveSensor touchSwitch = CapacitiveSensor(4, 2);

int LEDpin = 11;

void setup(){
```

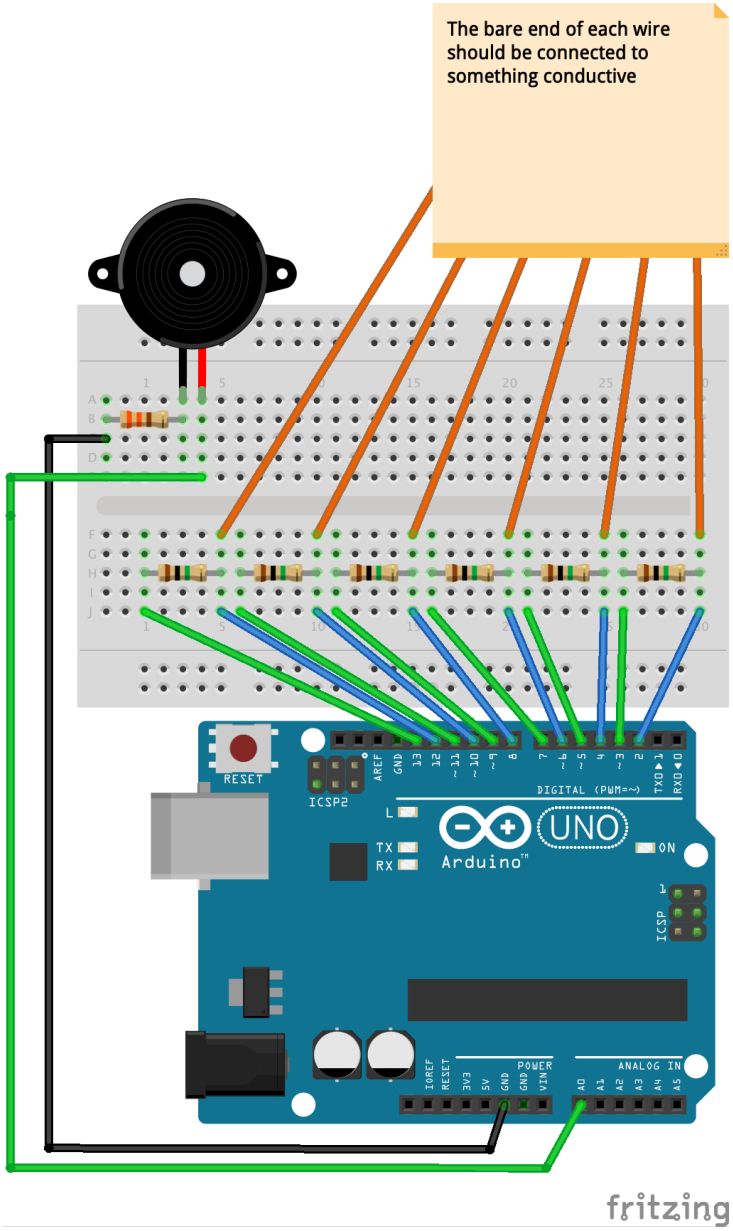
```
  pinMode(LEDpin, OUTPUT);
  Serial.begin(9600);
}


void loop(){
  long val = touchSwitch.capacitiveSensor(30);
  Serial.println(val); // print sensor output
  val = map(val, 0, 20000, 0, 255);
  val = constrain(val, 0, 255);

  analogWrite(LEDpin, val);
}
```

Paper piano

1. Add a piezo speaker to your circuit (see exercise 9) and further extend your circuit so it has 6 capacitive touch sensors. They should all be attached to different pins (see the image on the right and the code for pin numbers.
   You don't need the LED for this exercise.

2. Copy, paste and upload the code below.

   You'll now have an Arduino 'piano with 6 capacitive inputs, by touching the bare end of each wire, the piezo will play a different tone. (feel free to change the tones, see exercise 9 for a tone table)




```
#include <CapacitiveSensor.h>

CapacitiveSensor touchSwitchA =
CapacitiveSensor(13, 12);
//1 megohm resistor between pins 13 & 12,
pin 12 is sensor pin

CapacitiveSensor touchSwitchB =
CapacitiveSensor(11, 10);
//1 megohm resistor between pins 11 & 10, pin 10 is sensor pin

CapacitiveSensor touchSwitchC = CapacitiveSensor(9, 8);
//1 megohm resistor between pins 9 & 8, pin 8 is sensor pin
```

```
CapacitiveSensor touchSwitchD = CapacitiveSensor(7, 6);
//1 megohm resistor between pins 7 & 6, pin 6 is sensor pin

CapacitiveSensor touchSwitchE = CapacitiveSensor(5, 4);
//1 megohm resistor between pins 5 & 4, pin 4 is sensor pin

CapacitiveSensor touchSwitchF = CapacitiveSensor(3, 2);
//1 megohm resistor between pins 3 & 2, pin 2 is sensor pin

int speakerPin = A0;

void setup()
{
  Serial.begin(9600);
  pinMode(speakerPin, OUTPUT);
}

void loop()
{

  //read sensor A
  long valueA = touchSwitchA.capacitiveSensor(30);
  valueA = constrain(valueA, 150, 3000);
  valueA = map(valueA, 150, 3000, 0, 255);
  Serial.println(valueA);

  if (valueA>100){   //if sensor value is above threshold, play tone
    tone(speakerPin, 440, 200);  // plays a tone of 440Hz for 200ms
  }

  //read sensor B
  long valueB = touchSwitchB.capacitiveSensor(30);
  valueB = constrain(valueB, 150, 3000);
  valueB = map(valueB, 150, 3000, 0, 255);
  Serial.println(valueB);

  if(valueB>100){  //if sensor B value is above threshold, play tone
    tone(speakerPin, 493, 200);  // plays a tone of 493Hz for 200ms
  }

  //read sensor C
  long valueC = touchSwitchC.capacitiveSensor(30);
  valueC = constrain(valueC, 150, 3000);
  valueC = map(valueC, 150, 3000, 0, 255);
  Serial.println(valueC);

  if (valueC>100){   //if sensor value is above threshold, play tone
    tone(speakerPin, 262, 200);  // plays a tone of 262Hz for 200ms
  }

  //read sensor D
  long valueD = touchSwitchD.capacitiveSensor(30);
  valueD = constrain(valueD, 150, 3000);
  valueD = map(valueD, 150, 3000, 0, 255);
```

```
  Serial.println(valueD);

  if (valueD>100){   //if sensor value is above threshold, play tone
    tone(speakerPin, 330, 200);  // plays a tone of 330Hz for 200ms
  }

  //read sensor E
  long valueE = touchSwitchE.capacitiveSensor(30);
  valueE = constrain(valueE, 150, 3000);
  valueE = map(valueE, 150, 3000, 0, 255);
  Serial.println(valueE);

  if (valueE>100){   //if sensor value is above threshold, play tone
    tone(speakerPin, 330, 200);  // plays a tone of 262Hz for 200ms
  }

  //read sensor F
  long valueF = touchSwitchF.capacitiveSensor(30);
  valueF = constrain(valueF, 150, 3000);
  valueF = map(valueF, 150, 3000, 0, 255);
  Serial.println(valueF);

  if (valueF>100){  //if sensor value is above threshold, play tone
    tone(speakerPin, 349, 200);  // plays a tone of 262Hz for 200ms
  }

}
```

C. Find something cool to use as input for your piano. You can use (almost) anything that conducts electricity.
Connect each 'thing' to a different bare sensor wire (with tape, a paperclip, by sticking it in…).