# Miniband

Type: Musical instrument
Difficulty: Medium
Time: 2-3 sessions


*The Hackday judges with the Miniband*

# Introduction

Make your own band! You'll get to create some musical instruments from a few simple electronic parts and plug them into your Rasperry Pi. You can then make beautiful music. Or a dreadful racket.

# Resources

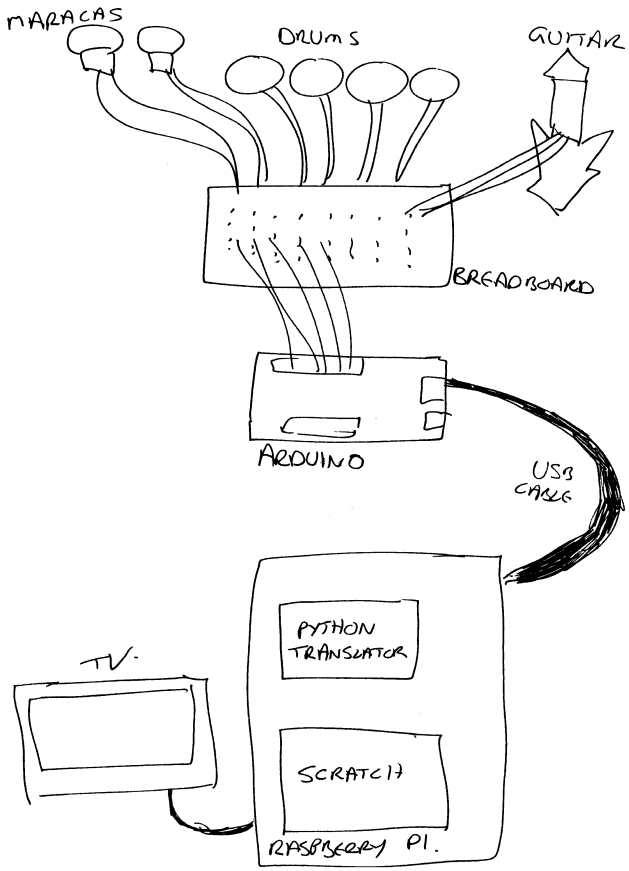
This project needs quite a few bits to make all the instruments. You'll need:

- 1 Arduino board with a USB cable (we used an Arduino Uno, but any type should do)

- 1 large breadboard or 3 small breadboards
- 1 x strip potentiometer (for the guitar)
- 2 x tilt sensors (for the maracas)
- 4 x piezo buzzers (the type in greetings cards) (for the drum kit)
- 7 × 10KΩ resistors
- Jumper leads, including some long female-female ones for connecting instruments to breadboards
- Plasticine (or some other modelling material, such as Sugru)
- Card
- Electrical tape
- Plastic drinking straws
- Pencil
- Coloured pens

# Overview

This project is complicated and there are few parts you need to build. This is how they all fit together.




Each instrument has its components plugged into a breadboard and connected to an Arduino. The Arduino is connected by a USB cable to the Raspberry Pi. The Arduino has a small

program on it that detects when the instruments are played and sends a signal, via the USB cable, to the Pi. A small Python program on the Pi reads those messages and translates them into a form that Scratch and understand. The Scratch project reads the signals and plays the sounds, and does some cool animation as well.

We'll make one instrument work first, then add the others. Getting the first instrument working takes quite a while; the others are a lot quicker.

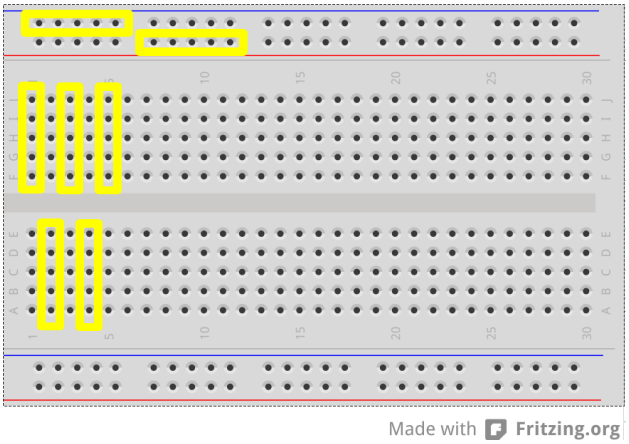
### Wiring and circuit diagrams

The wiring and circuit diagrams show the different instruments individually. This is just for clarity. In reality, there are lots of wires attached to the same Arduino.

### Breadboard wiring

If you've not used a breadboard before, you may be confused by the mess of holes. Generally, groups of about six holes are connected electrically; if you plug wires into two of these holes, the wires are connected. These groups are generally labelled with numbers and are shown as columns in the diagrams here.

Your breadboard might also have two rows of holes at the end with red and black lines nearby. These span over several columns and are often used for power. The red side is generally used as +5V and the black side as ground (GND).




The yellow highlights show which groups of holes are electrically connected.

# Step 1: Make the Maracas

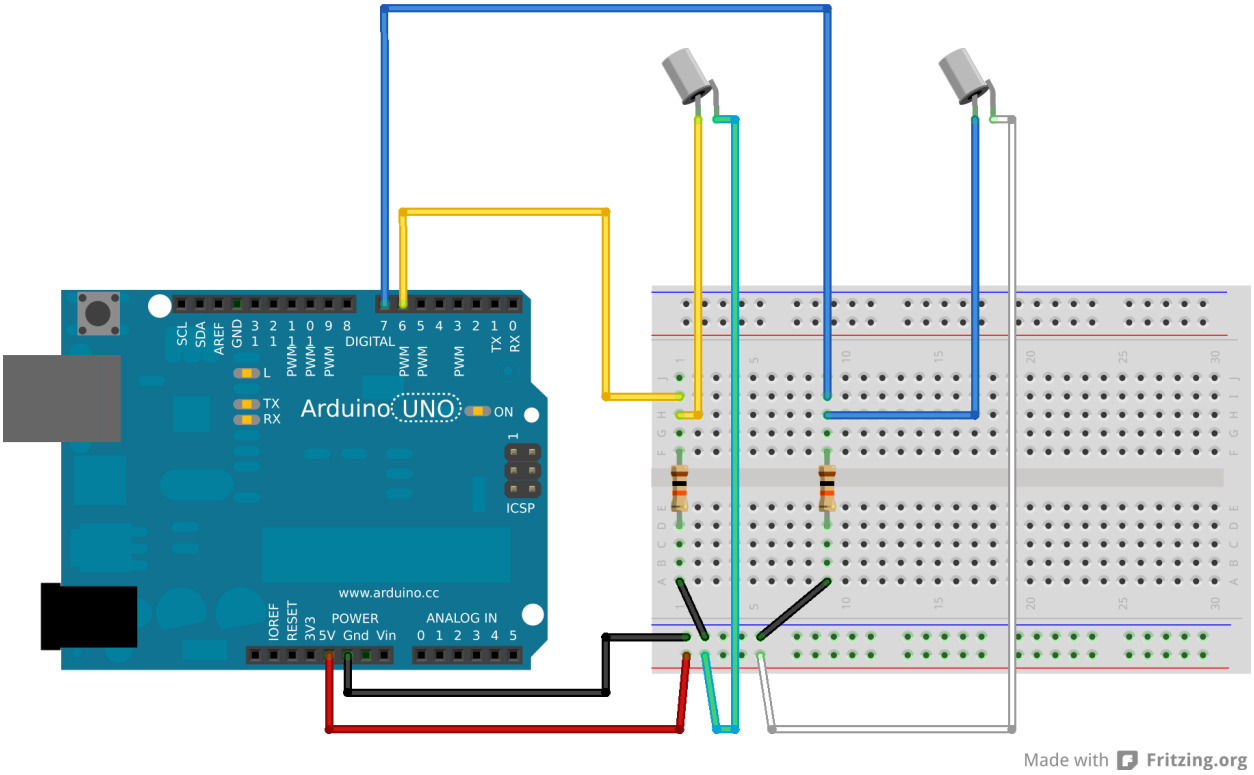
## You will need:

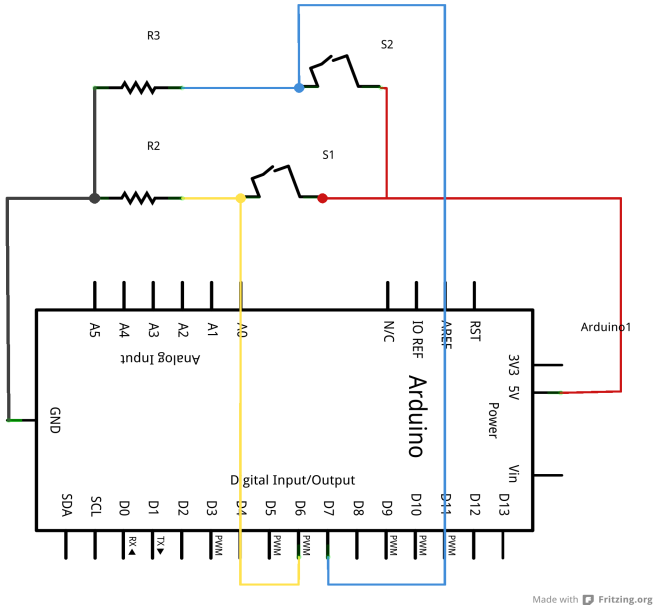
2 tilt switches
long female-female jumper leads
various other jumper leads

Two 10KΩ resistors
Straw
Plasticine

## Wire up the maracas

1. Connect each tilt switch to two long female-female jumper leads. Connect ordinary male-male jumper leads at each end.
2. Use a jumper lead to connect the +5V and GND pins on the Arduino to the breaboard.
3. Plug one end of each maraca into the breadboard, next to the +5V lead
4. Connect two 10KΩ resistors across the gap in the centre of the breadboard
5. Connect a jumper lead between the ground row and the column of one end of the resistor
6. Plug the end of the other maraca leads into the breadboard on the same column as the other end of the resistor.
7. Use another jumper lead to connect between near the resistor and pins 6 and 7 on the Arduino.
8. Wire in the other jumper leads following the diagram below




## Install the Arduino code

Open the Arduino environment on the Raspberry Pi and add this script:

```
const int NUMBER_OF_MARACAS = 2;
```

```
const int MARACAS_PINS[] = {6, 7};
const int MARACAS_DELAY = 5000;

int maracas_states[] = {-1,-1};
int maracas_delays[] = {0, 0};

void setup() {
  for (int i = 0; i < NUMBER_OF_MARACAS; i++) {
    pinMode(MARACAS_PINS[i], INPUT);
  }
  Serial.begin(9600);
}

void loop() {
  maracas();
  update_delays();
}


void maracas() {
  for (int i = 0; i < NUMBER_OF_MARACAS; i++) {
    int newState = digitalRead(MARACAS_PINS[i]);
    if (newState != maracas_states[i] && maracas_delays[i] < 1) {
      Serial.print("maracas,");
      Serial.print(i);
      Serial.println();
      maracas_states[i] = newState;
      maracas_delays[i] = MARACAS_DELAY;
    }
  }
}

void update_delays() {
  for (int i = 0; i < NUMBER_OF_MARACAS; i++) {
    if (maracas_delays[i] > 0) {
      maracas_delays[i] -= 1;
    }
  }
}
```

Save the script as `instrument`
Upload the script to the Arduino

<test your project>
Open the serial port monitor in the Arduino environment and shake the tilt switches. Do you get messages when you move one of the switches? Do the readings stop when you stop moving them?

If nothing happens, it's probably that you've made a mistake in the wiring. Make sure that everything is in the right holes: it's very easy to put things in the wrong row. Make sure that the everything (especially the resistors) is pushed fully into the holes, but don't push too hard.

### Assemble the maracas

1. Cut two lengths of straw and put a large blob of Plasticine on one end of each.
2. Slide the tilt switch into the straw and push it into the plasticine so it sticks.

<test your project>
Open the serial port monitor again if you closed it earlier. Do the maracas still work? Do you get messages when you move one of the switches? Do the readings stop when you stop moving them?

# Step 2: Add the Arduino/Scratch translator reader

This is a complicated piece of Python that does a very simple thing. Basically, it monitors a bunch of USB ports and listens for any signals from them. If it reads a signal one one, it translates it and sends it on to Scratch.

Open `codeclub/miniband/miniband.py` with the Leafpad editor. Change the `DEVICES =` line at the top of the script to refer to your Arduino. It might be

```
DEVICES = ['/dev/ttyACM0']
```

or

```
DEVICES = ['/dev/ttyACM2']
```

depending on what you've plugged in.

Don't run this script yet: it wants to connect to Scratch as well as the Arduino, and we've not done that bit yet.

#### How do I know what the Raspberry Pi thinks my Arduino is called?

The Raspberry Pi gives each Aduino device a name when it's plugged in. Both the Arduino programming environment and the Pyhon translator need to know these names.
The Arduino tool shows you a list of available devices in the Tools | Serial Port menu.
The other way to find out is to open an LXTerminal window, plug in the Adruino, and they type

```
dmesg
```

and look at the output. You should see something like

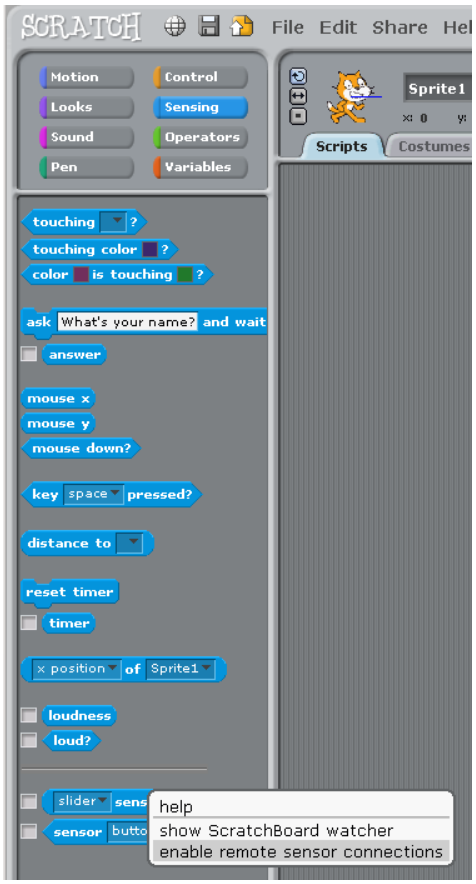
```
[255958.705055] usb 1-1.3.3: new full-speed USB device number 13 using dwc_otg
[255958.810055] usb 1-1.3.3: New USB device found, idVendor=2341, idProduct=0043
[255958.810088] usb 1-1.3.3: New USB device strings: Mfr=1, Product=2, SerialNumber=220
[255958.810105] usb 1-1.3.3: Manufacturer: Arduino (www.arduino.cc)
[255958.810120] usb 1-1.3.3: SerialNumber: A4131363139351117181
```

```
[255958.811917] cdc_acm 1-1.3.3:1.0: ttyACM0: USB ACM device
```

and “`/dev/ttyACM0`” is your Adruino.

# Step 3: Create the maracas player in Scratch

1. Start a new Scratch project.
2. Change the sprite to have the `miniband/maracas` costume (you may want to make it smaller). Import the `miniband/maracas` sound. You can delete the existing cat costumes and the meow sound.
3. Open the "Sensing" palette and right-click on the [slider] sensor value block at the bottom. Select the "Enable remote sensor connections" on the menu that pops up. This allows Scratch to listen to the Python translation script.




4. Now start that script. Open an LXTerminal window and type

```
cd ~/codeclub/miniband
python miniband.py
```

You should get some output that looks like this

```
INFO:root:Connecting to Scratch...
INFO:root:Connecting to Scratch
INFO:root:Connected to Scratch
INFO:root:Started listener on port /dev/ttyACM0
INFO:root:Listeners running....
```

```
INFO:root:Instrument: maracas, Value: 0
INFO:root:broadcast maracas
INFO:root:Instrument: maracas, Value: 1
INFO:root:broadcast maracas
```

The last two lines will repeat every time you move the maracas. (The "value" is which maracas is shaken.)

You can stop the Python `miniband` script by pressing Ctrl+C.

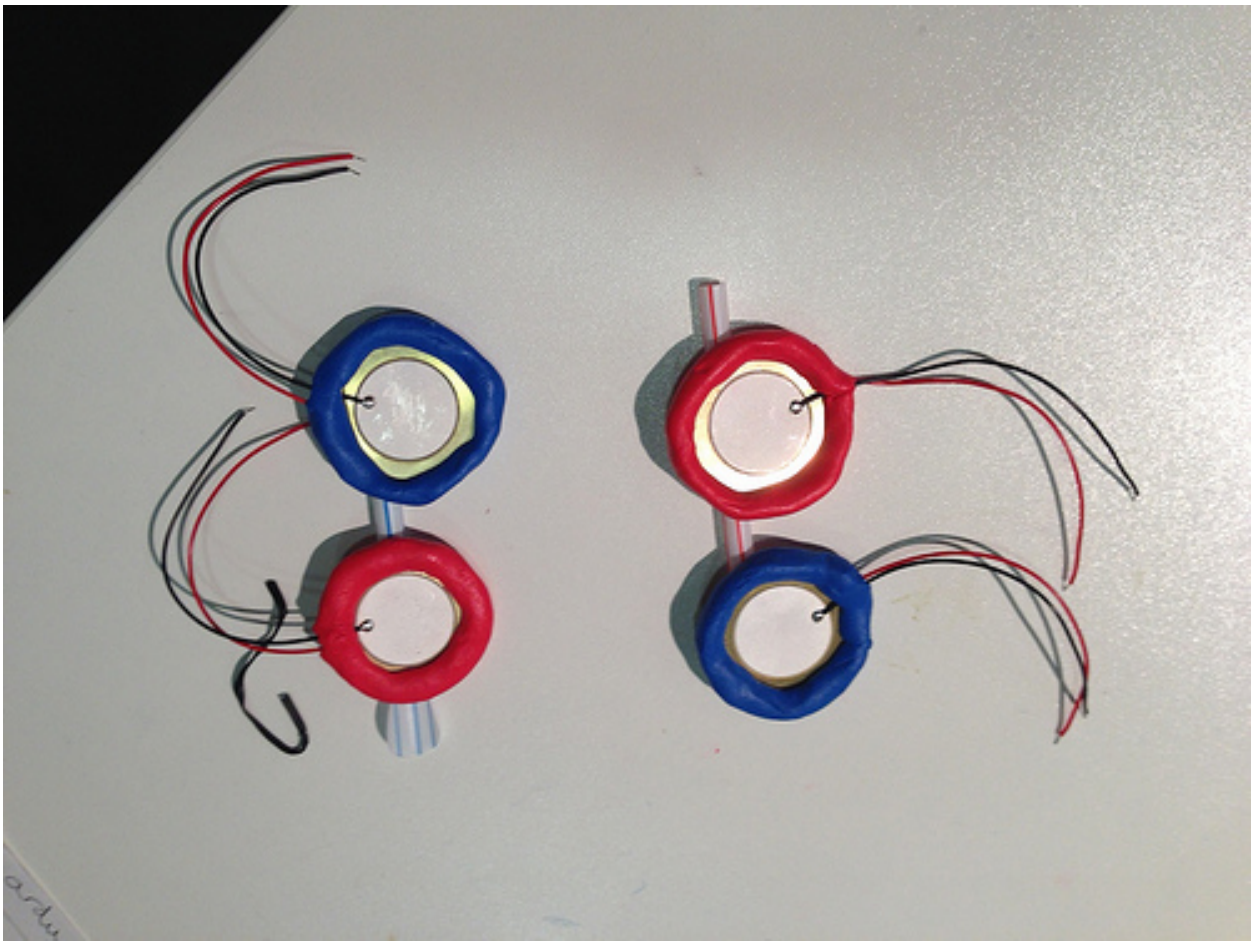
<test your project> Shake the maracas while the Python `miniband` script is running. Do you see extra "broadcast maracas" lines when you do?

5. Go back to Scratch and in the Control palette you should see a `when I receive [maracas]` hat block. Use that to make this script:


<test your project> Double-click on the maracas script in Scratch. Do you hear the sound? Now shake the maracas instrument you made. Do you hear the sound? Does you still see extra "broadcast maracas" lines when you shake them?

# Step 4: Create the drum kit

You will need:
4 piezo-electric speakers
Four 10KΩ resistors
Various jumper leads
Plasticine
Straw
Card

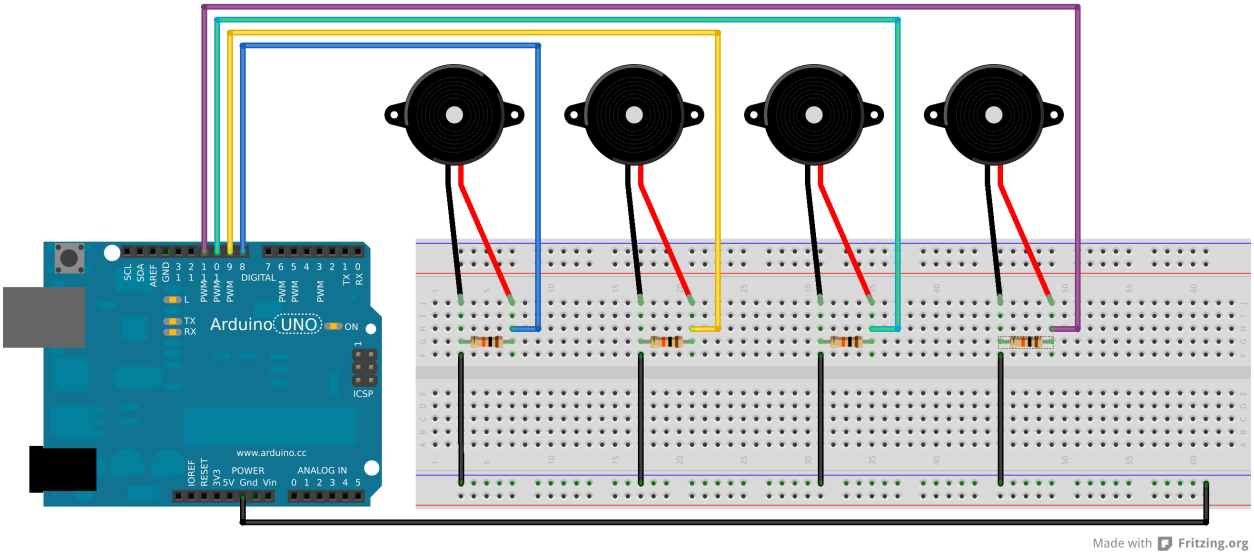


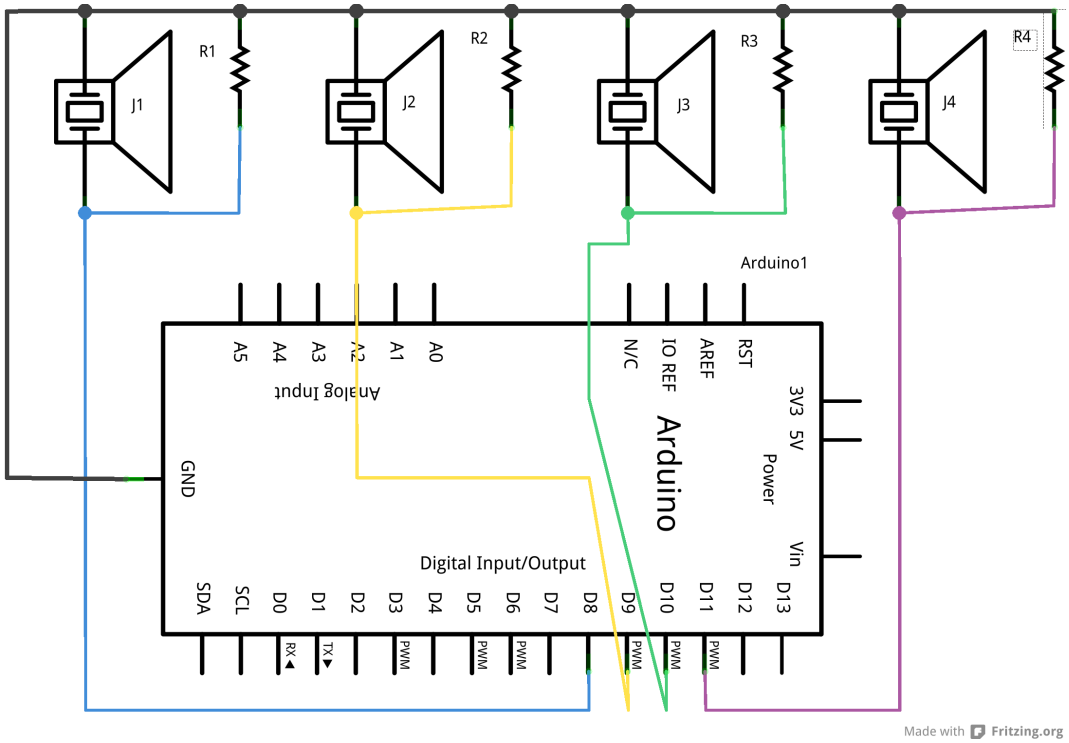
## Wire up the drums

Piezo-electric crystals change shape when an electric current is sent through them. This makes them work as simple loudspeakers. But if you change their shape (by tapping them), they make an electric current. We'll use that current to signal that a drum has been hit.

1. Use a long jumper to connect the GND pin on the Arduino to the ground rail on the breadboard.
2. Connect a jumper between the ground rail and a column of the breadboard.
3. Connect the black lead of the speaker to the same column.
4. Connect a 10KΩ resistor between that column and one nearby.
5. Connect the red lead of the speaker to this second column.
6. Connect a long jumper from this second column to pin 8 on the Arduino.
7. Repeat steps 2 to 6 for the other three speakers, connecting them to Arduino pins 9, 10, and 11.

Be careful! The connections to the speakers are quite delicate. If you pull too hard on the leads, it's very easy to snap them off.




## Update the Arduino program

Modify your `instrument` Arduino program. The changed lines are highlighted.

```
const int NUMBER_OF_MARACAS = 2;
const int MARACAS_PINS[] = {6, 7};
const int MARACAS_DELAY = 5000;

const int NUMBER_OF_DRUMS= 4;
const int DRUM_PINS[] = {8, 9, 10, 11};
```

```
const int DRUM_DELAY = 2500;

int maracas_states[] = {-1,-1};
int maracas_delays[] = {0, 0};

int drum_delays[] = {0, 0, 0, 0};

void setup() {
  for (int i = 0; i < NUMBER_OF_MARACAS; i++) {
    pinMode(MARACAS_PINS[i], INPUT);
  }
  for (int i = 0; i < NUMBER_OF_DRUMS; i++) {
    pinMode(DRUM_PINS[i], INPUT);
  }
  Serial.begin(9600);
}

void loop() {
  maracas();
  drums();
  update_delays();
}


void maracas() {
  for (int i = 0; i < NUMBER_OF_MARACAS; i++) {
    int newState = digitalRead(MARACAS_PINS[i]);
    if (newState != maracas_states[i] && maracas_delays[i] < 1) {
      Serial.print("maracas,");
      Serial.print(i);
      Serial.println();
      maracas_states[i] = newState;
      maracas_delays[i] = MARACAS_DELAY;
    }
  }
}

void drums() {
  for (int i = 0; i < NUMBER_OF_DRUMS; i++) {
    if (digitalRead(DRUM_PINS[i]) == HIGH && drum_delays[i] < 1) {
      Serial.print("drum,");
      Serial.print(i);
      Serial.println();
      drum_delays[i] = DRUM_DELAY;
    }
  }
}
```

```
void update_delays() {
  for (int i = 0; i < NUMBER_OF_MARACAS; i++) {
    if (maracas_delays[i] > 0) {
      maracas_delays[i] -= 1;
    }
  }
  for (int i = 0; i < NUMBER_OF_DRUMS; i++) {
    if (drum_delays[i] > 0) {
      drum_delays[i] -= 1;
    }
  }
}
```

Save your code.
Upload  your program to the Arduino.

<test your project>
Open the serial port monitor in the Arduino environment and tap the speakers. Do you get messages when you tap one of the drums? You may have to tap them quite sharply, but be careful not to bend them. Do you get different messages for different drums? Do you get a message for each drum? Do the readings stop when you don't tap anything?

### Assemble the drums

This is easy. Just put a ring of plasticine around the rim of each drum to hold it in place. Make sure the metal disk isn't touching the table: it make the drum more sensitive. You can make card cylinders in the shape of drums and place the disks on top.

If you can, arrange the disks with the white part facing down so you don't tap it every time. It will last longer.

<test your project>
Open the Arduino serial port monitor again if you closed it earlier. Do the drums still work?

## Step 5: Test the Python translator

<test your project>
Ensure Scratch is still running the miniband script then run
`python miniband.py`

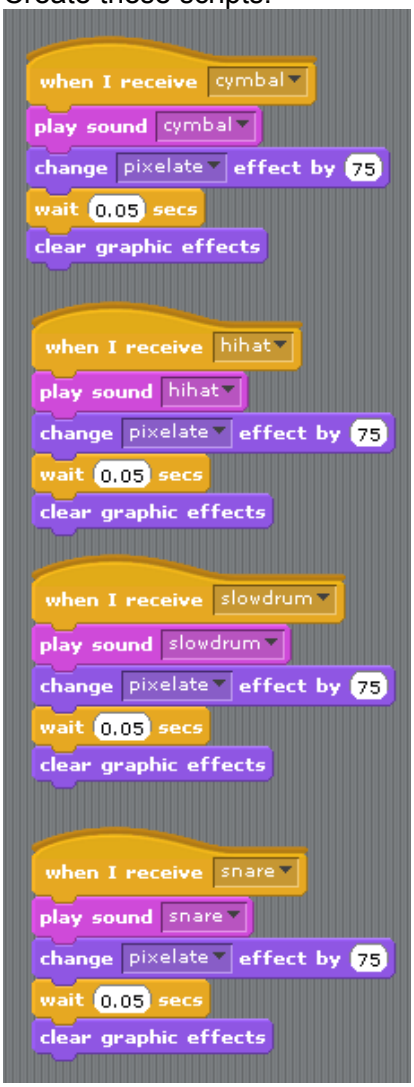
Wait for it to connect to both Scratch and the Arduinos. Then tap the drums and watch the output. You should get something like this:

```
INFO:root:Instrument: drum, Value: 0
INFO:root:broadcast cymbal
INFO:root:Instrument: drum, Value: 1
INFO:root:broadcast hihat
INFO:root:Instrument: drum, Value: 2
INFO:root:broadcast slowdrum
INFO:root:Instrument: drum, Value: 3
INFO:root:broadcast snare
```

If it doesn't work, check that Scratch is running and that you've got the correct names for your Arduino devices. (Use `dmesg` to check after you unplug them and plug them in again.)

# Step 6: Update the Scratch player

1. Create a new sprite called drum. Use the standard Scratch drum costume.
2. Import into drum the sounds `miniband/cymbal`, `miniband/hihat`, `miniband/slowdrum`, and `miniband/snare`.
3. Create these scripts:

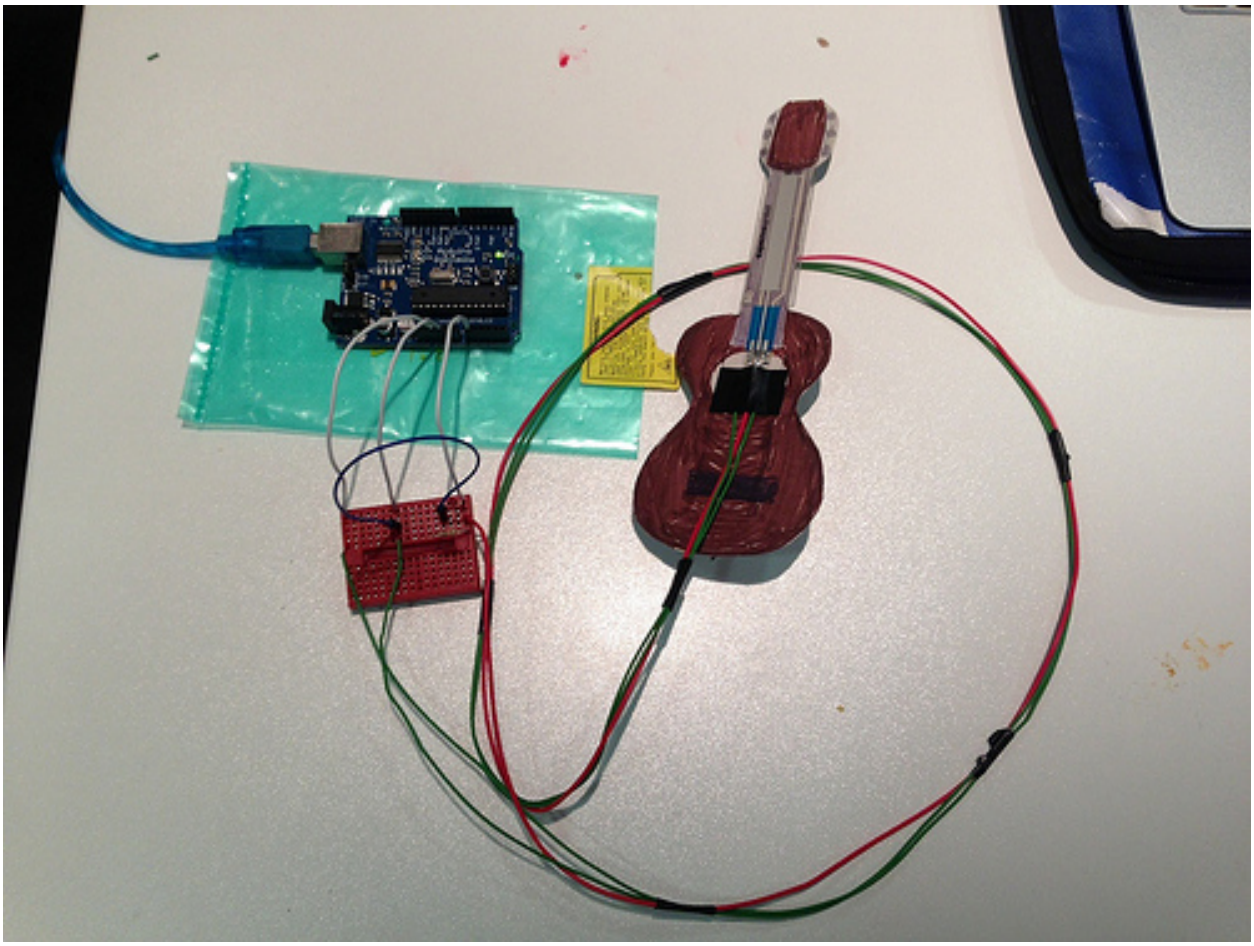


4. Make sure the Python translator script is running.

<test your project>
Double-click on each of the drums scripts in Scratch. Do you hear the right sound? Now tap the drums instrument you made. Do you hear the sound? Do you still see extra "broadcast slowdrum" lines when you tap them them?

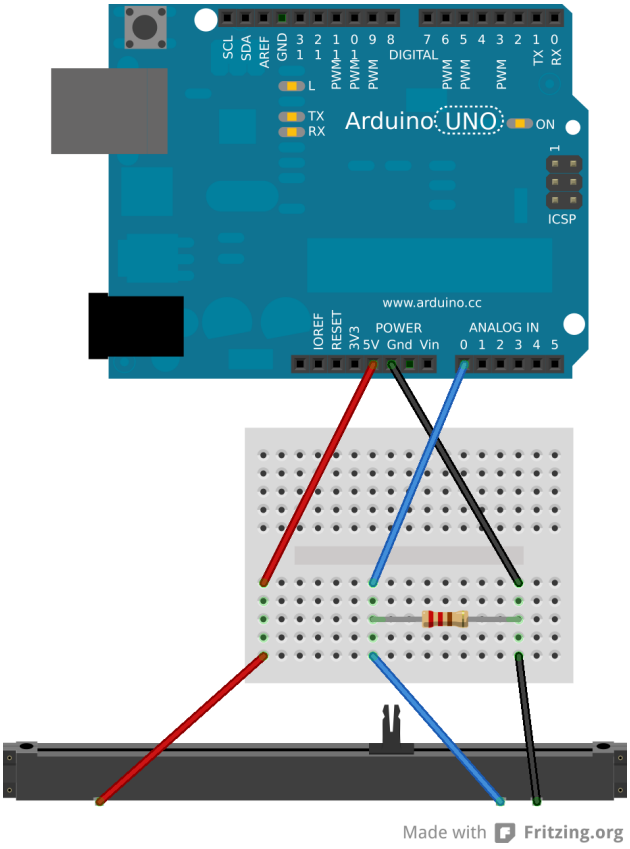
# Step 7: Make the guitar

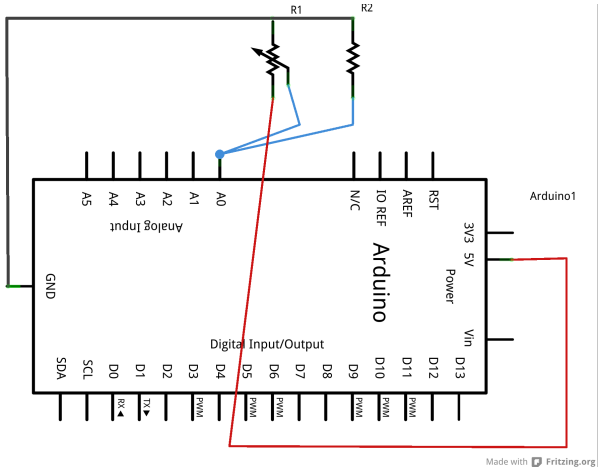
You will need:
strip potentiometer
Jumper leads
One 10KΩ resistor
Pencil
Straw
Card



## Wire up the guitar

1. Connect the potentiometer to three long jumper leads. Plug the other ends into three different columns on the breaboard.
2. Use a jumper lead to connect the +5V rail on the breadboard to one of the outer terminals of the potentiometer
3. Use a jumper lead to connect the GND rail on the breadboard to the other outer terminal of the potentiometer
4. Use a jumper lead to connect the A0 analogue pin on the Arduino to the centre terminal of the potentiometer
5. Connect a 10KΩ resistor between the centre and GND terminals of the potentiometer




## Install the Arduino code

Open the Arduino environment and modify your `instrument` script (the changes are highlighted):

```
const int NUMBER_OF_MARACAS = 2;
const int MARACAS_PINS[] = {6, 7};
const int MARACAS_DELAY = 5000;

const int NUMBER_OF_DRUMS= 4;
```

```
const int DRUM_PINS[] = {8, 9, 10, 11};
const int DRUM_DELAY = 2500;

const int GUITAR_THRESHOLD = 20;
const int GUITAR_PIN = A0;
const int GUITAR_DELAY = 5000;

int maracas_states[] = {-1,-1};
int maracas_delays[] = {0, 0};

int drum_delays[] = {0, 0, 0, 0};

int guitar_delay = 0;

void setup() {
  for (int i = 0; i < NUMBER_OF_MARACAS; i++) {
    pinMode(MARACAS_PINS[i], INPUT);
  }
  for (int i = 0; i < NUMBER_OF_DRUMS; i++) {
    pinMode(DRUM_PINS[i], INPUT);
  }
  Serial.begin(9600);
}

void loop() {
  maracas();
  drums();
  guitar();
  update_delays();
}


void maracas() {
  for (int i = 0; i < NUMBER_OF_MARACAS; i++) {
    int newState = digitalRead(MARACAS_PINS[i]);
    if (newState != maracas_states[i] && maracas_delays[i] < 1) {
      Serial.print("maracas,");
      Serial.print(i);
      Serial.println();
      maracas_states[i] = newState;
      maracas_delays[i] = MARACAS_DELAY;
    }
  }
}

void drums() {
  for (int i = 0; i < NUMBER_OF_DRUMS; i++) {
```

```
    if (digitalRead(DRUM_PINS[i]) == HIGH && drum_delays[i] < 1) {
      Serial.print("drum,");
      Serial.print(i);
      Serial.println();
      drum_delays[i] = DRUM_DELAY;
    }
  }
}

void guitar() {    //
  int pitch = analogRead(GUITAR_PIN);
  if (pitch >= GUITAR_THRESHOLD && guitar_delay < 1) {
    Serial.print("guitar,");
    Serial.print(pitch);
    Serial.println();
    guitar_delay = GUITAR_DELAY;
  }
}

void update_delays() {
  for (int i = 0; i < NUMBER_OF_MARACAS; i++) {
    if (maracas_delays[i] > 0) {
      maracas_delays[i] -= 1;
    }
  }
  for (int i = 0; i < NUMBER_OF_DRUMS; i++) {
    if (drum_delays[i] > 0) {
      drum_delays[i] -= 1;
    }
  }
  if (guitar_delay > 0) {
    guitar_delay -= 1;
  }
}
```

Save the script
Upload the script to the Arduino.

<test your project>
Open the serial port monitor in the Arduino environment and press the potentiometer at different places. Do you get different readings appear in the serial port monitor? Do the readings stop when you stop pressing?

## Assemble the guitar

1. Draw a guitar shape on the card. Colour it in and cut it out. It needs to be large enough that the strip potentiometer fits on the neck of the guitar.

2. Stick the pencil to the back of the guitar to stiffen it. You need to make sure that the pencil goes all the way along the neck.
3. Stick the potentiometer to the fretboard of the guitar.

<test your project>
Open the serial port monitor again if you closed it earlier. Does the guitar still work? Do you still get different readings when you press the guitar in different places? Does the output stop when you're not touching it?

# Step 8: Test the Python translator

<test your project>
Ensure Scratch is still running the miniband script then run
`python miniband.py`

Wait for it to connect to both Scratch and the Arduinos. Then tap the press the potentiometer and watch the output. You should get something like this:

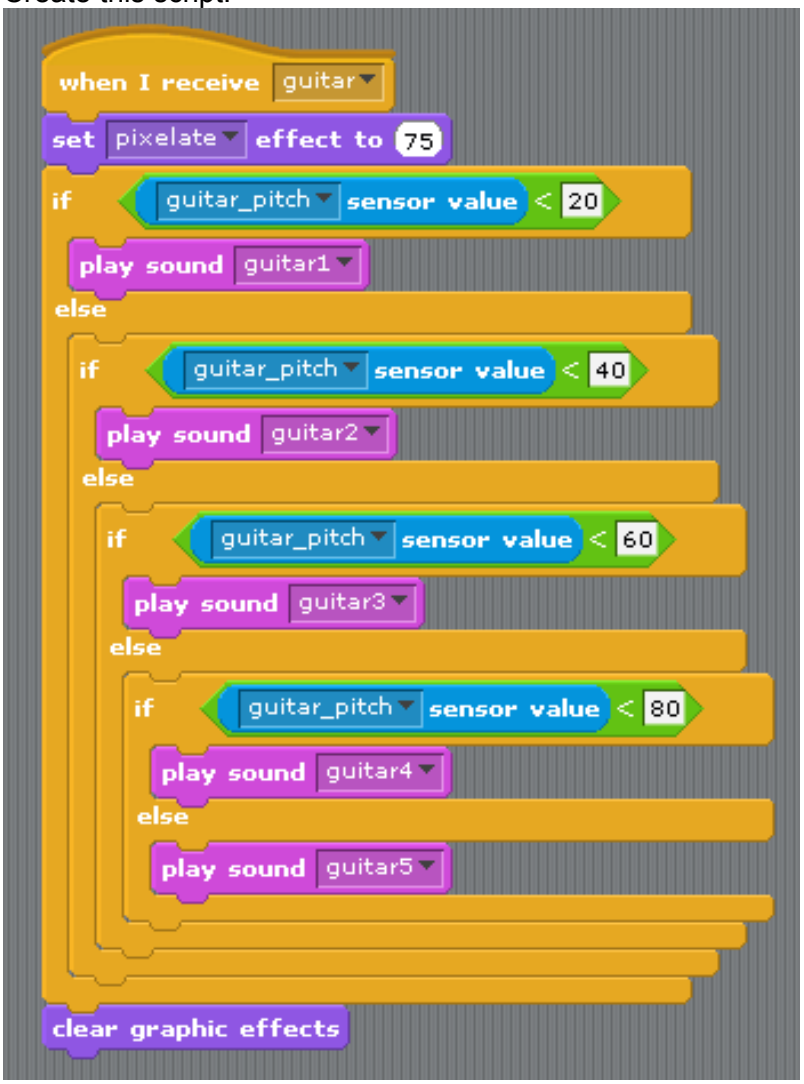
```
INFO:root:Instrument: guitar, Value: 260
INFO:root:sensor-update guitar_pitch 25
INFO:root:broadcast guitar
INFO:root:Instrument: guitar, Value: 75
INFO:root:sensor-update guitar_pitch 7
INFO:root:broadcast guitar
INFO:root:Instrument: guitar, Value: 898
INFO:root:sensor-update guitar_pitch 87
```

If it doesn't work, check that Scratch is running and that you've got the correct names for your Arduino devices. (Use `dmesg` to check after you unplug them and plug them in again.)

# Step 9: Update the Scratch player

1. Create a new sprite called guitar. Use the `miniband/guitar` costume.
2. Import into drum the sounds `miniband/guitar1`, `miniband/guitar2`, `miniband/guitar3`, , `miniband/guitar4`, and `miniband/guitar5`.

3. Create this script:




4. Make sure the Python translator script is running.

<test your project>
Double-click on the script in Scratch. Do you hear a guitar? Now press the physical guitar. Do you hear a sound? Do you hear different sounds when you press it in different places? Do you still see "broadcast guitar" lines?

# Step 10: Rock out!

Your miniband works! Rock!

# Notes for volunteers

## Install the resources you'll need

Open a terminal window on the Raspberry Pi and run these commands:

```
sudo aptitude update
sudo aptitude full-upgrade
sudo aptitude install arduino python-serial
```

That will install the Arduino programming environment and the libraries that allow Python to read data from the Arduinos.

Copy the Miniband resources into the (new) directory `/home/pi/codeclub/miniband`

## Troubleshooting the electronics

When the circuits don't work, check that all the components are pushed in fully and that all the connections are in the same row of breadboard holes. Children are very good at putting things in nearby holes, and it's difficult to spot.

All the resistors used here are *pull down resistors*. In the absence of any other input, it will make the output of the instrument be at zero volts, which will read zero. All the instruments will have them, to ensure that they read zero when not being used.

## Non-obvious Arduino code

Note the use of the `maracas_states` array to record whether the tilt switches were open or closed just now. The script will only send a message if the switches change.

Note the use of `maracas_delays` (and other similar arrays) to stop the Arduino sending too many signals to the Raspberry Pi. Every time the Arduino sends a signal, it sets `maracas_delays` to a large value. Every time through the loop, it reduces that value by one. It will only send a new signal if the `maracas_delays` has been reduced to zero.