ARDUINO

Workshop

HCI Technologies

Danica Mast

Chris Heydra

Version 5.0 28-04-2019

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:

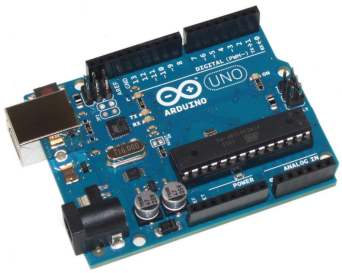
- Push buttons
- Twist knobs
- Light sensor
- Temperature sensor
- Accelerometer
- Infrared sensor
- CO2 sensor
- CO sensor
- Ultrasonic distance sensor
- Alcohol sensor
- Capacitive touch sensor
- Microphone
- Hartrate sensor
- … and lots more

Examples of actuators:

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

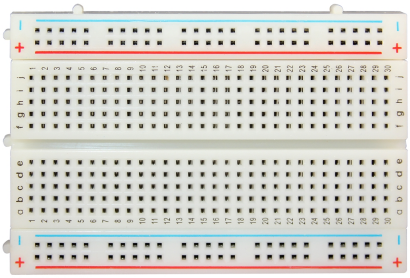
Hardware

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.

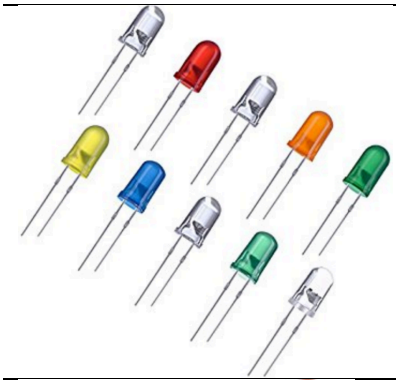


Breadboard

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.

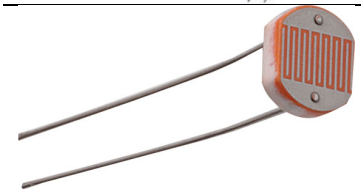
Jumper Wires

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



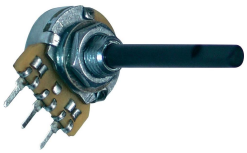
LED

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.



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.

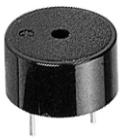


Potentiometer
A potentiometer (often abbreviated to pot-meter or just pot) is a rotary knob that has a variable resistance. The pot we use has a maximum resistance of 10k

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.

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 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.
C. Open the Arduino IDE software.
D. In the menu navigate to Tools > Board and choose *Arduino/Genuino 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/Genuino 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.

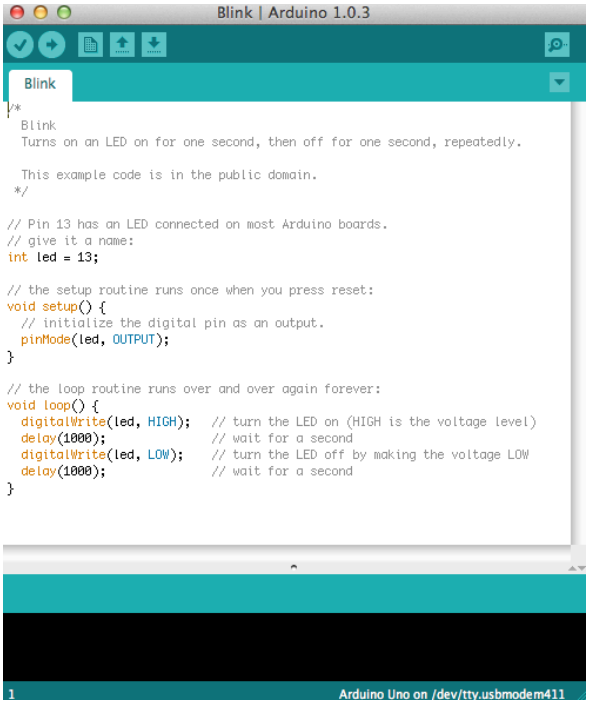
Open
Opens an existing sketch from your sketchbook.

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*.

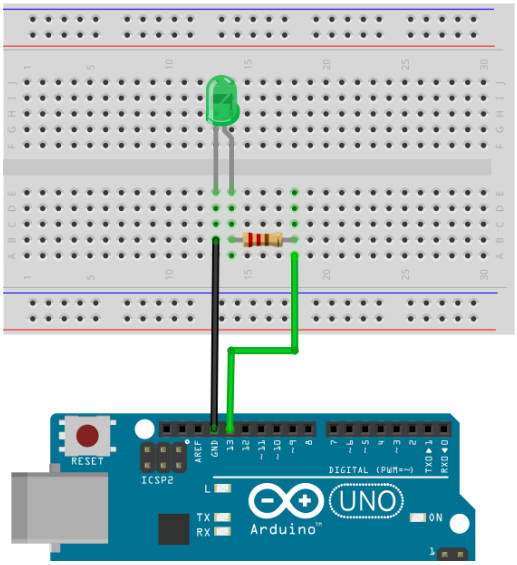
   On the board the LEDs labelled TX and RX should start blinking to indicate that the board is communicating with the computer. After a while the message "done uploading" will appear in the message bar.

   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.



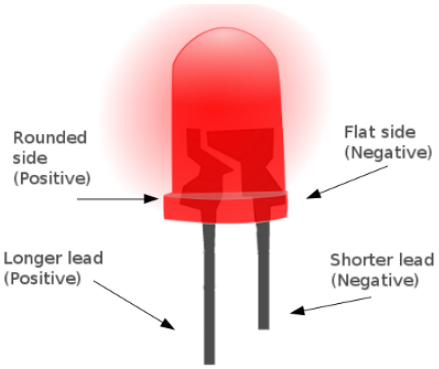
A. Disconnect the Board from your computer. Then recreate the circuit shown in the image on your breadboard.

   You will need:

   - 1x 220Ω (Ohm) resistor. Colour code [red red brown gold] or [red red black black brown]
   - 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[1].

   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 before or after 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.

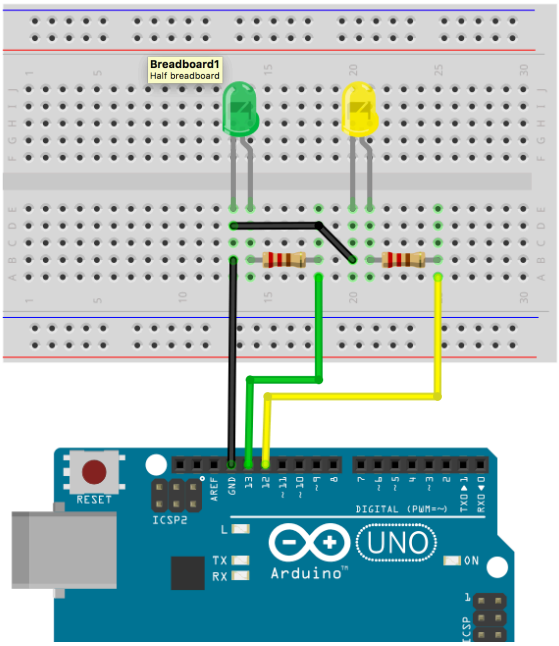
[1] The colours of the jumper wires do not matter for the operation of your circuit. However, there are conventions that make it easier to understand your circuit. Black is used for ground (GND) and red (not used here) for the power supply (5V).

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 220Ω) 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.

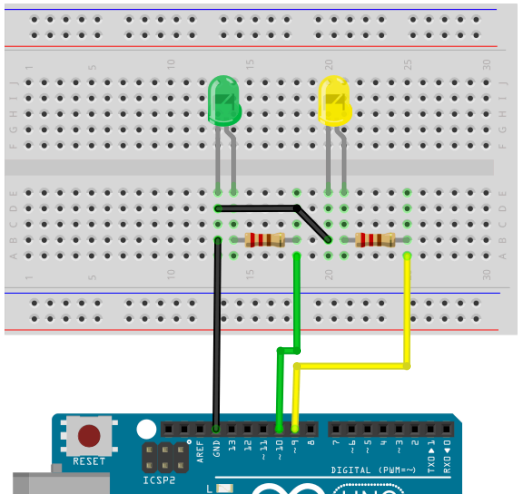
Exercise 4: Fade LED – Analog Output

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[2] (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
}
```

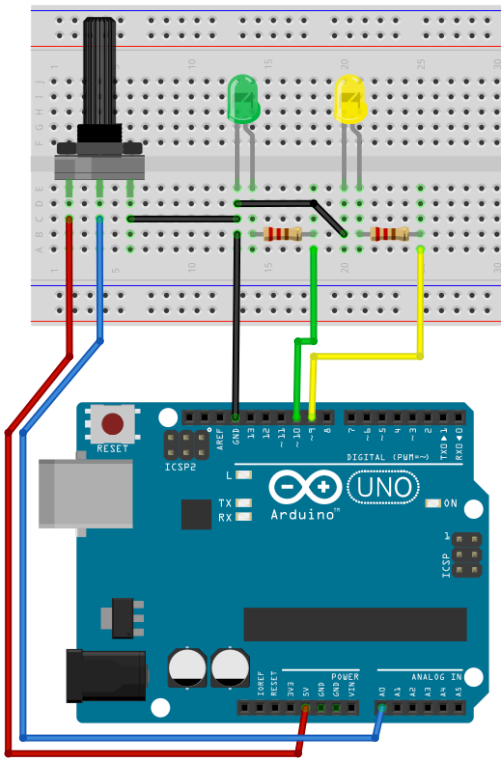
[2] Although the function `analogWrite()` is used, PWM is not truly analog. The pin pulses between HIGH and LOW at a very high frequency. The duration (or width) of the HIGH and LOW pulse determines the output voltage. For more info on PWM check out https://learn.sparkfun.com/tutorials/pulse-width-modulation/all

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 the previous exercise 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. The LEDs can be left in place; these are not used in this exercise, but you will need them later.

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.

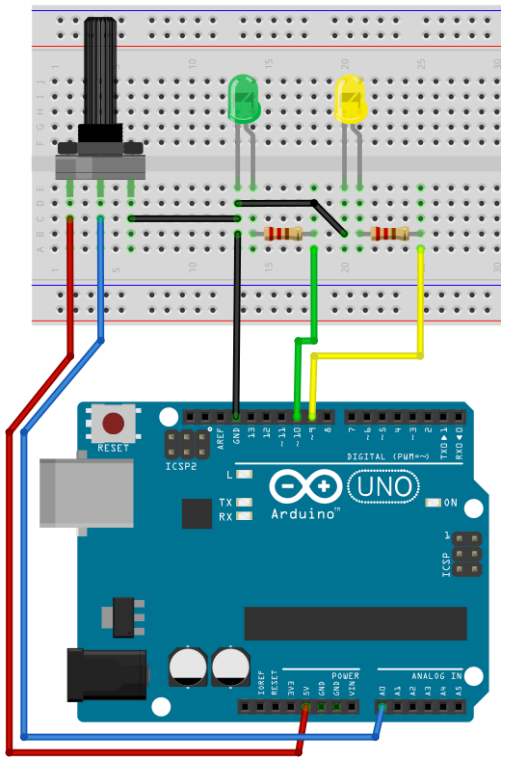
A new screen now opens; the Serial Monitor. It shows everything that is sent to the computer by the Arduino board. And in this case, because of the program we just uploaded, the values of the potentiometer are shown. Make sure the value in the lower right corner is set to 9600 baud. Carefully turn the 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 5 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 5 as a basis for this exercise.

1. The outcome of exercise 5 was that the Arduino read a value between 0 and 1023, which was received from 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;
```

   Make sure the LEDs are connected to these pins.

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

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

   In the `loop()` section add the following lines below where you calculated the `sensorValue`.

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

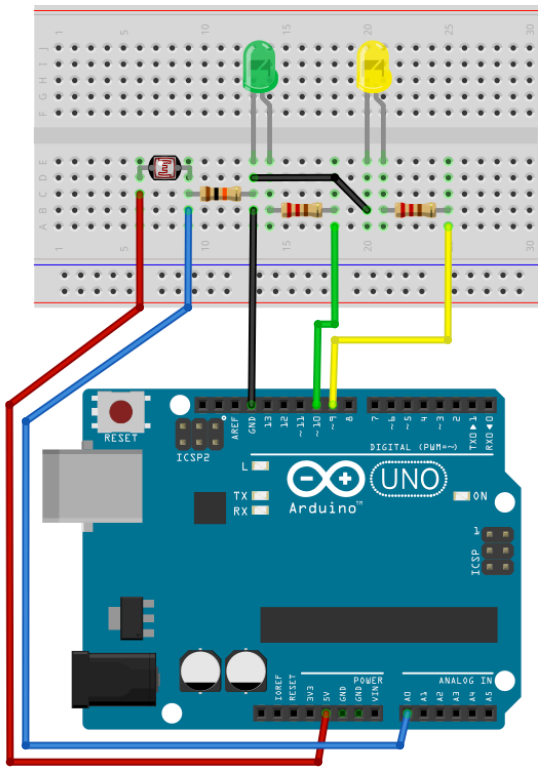
4. Verify and upload your code to the board. If you now twist the potmeter, the LEDs should change their brightness. You might notice that the fading isn't really fluently. To remedy this, shorten the delay. A good value for the delay is 20.

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: Control the LED with light

In the previous exercise 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 sensitive 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 line is now temporarily disabled.

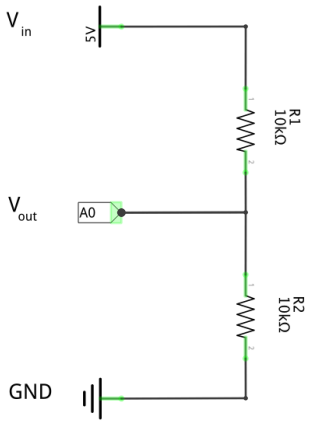
```
// sensorValue = (sensorValue/1023)*255;
```

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) and lowest (obscured) value. This the actual range the sensor has under these lighting conditions.
4. Uncomment the line you commented and change it to rescale the sensor value in the proper way. The object here is to rescale the actual range you just wrote down 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 8: Voltage Divider

The sensors of exercise 5, 6 and 7 are all based on a very common circuit in electronics called a *voltage divider*. The A0 - A5 pins of the Arduino are all capable of measuring a certain voltage between 0 and 5V and converting that to a number between 0 and 1023 we can use in our code.



A voltage divider, as shown in this image, is a circuit that turns a large voltage into a smaller one. Using two resistors (R1, R2) and an input voltage (V_{in}), we can create an output voltage (V_{out}) that is a fraction of the input.
The formula to calculate V_{out} is this:

$$V_{out} = V_{in} * \frac{R2}{R1+R2}$$

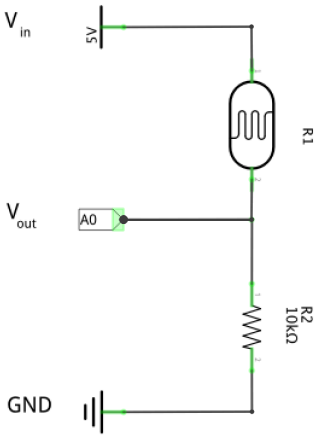
Let's fill in the calculation for the example shown in the image. Both R1 and R2 are 10kΩ (10000Ω) and V_{in} is 5v.

$$V_{out} = 5 * \frac{10000}{10000+10000} = 2.5$$

With two resistors of the same value, the output voltage is 2.5v, exactly half of the input voltage.

Variable resistor

If you look back at exercise 7, you can see a slightly different voltage divider in the circuit. This part of the circuit is schematically shown in this image. R1 is replaced by a variable resistor, namely a Light Dependant Resistor. So when placed in the dark, R1 will be quite high (appr 5MΩ or 5000000Ω) and when placed in bright light R1 will be quite low (appr 200Ω).



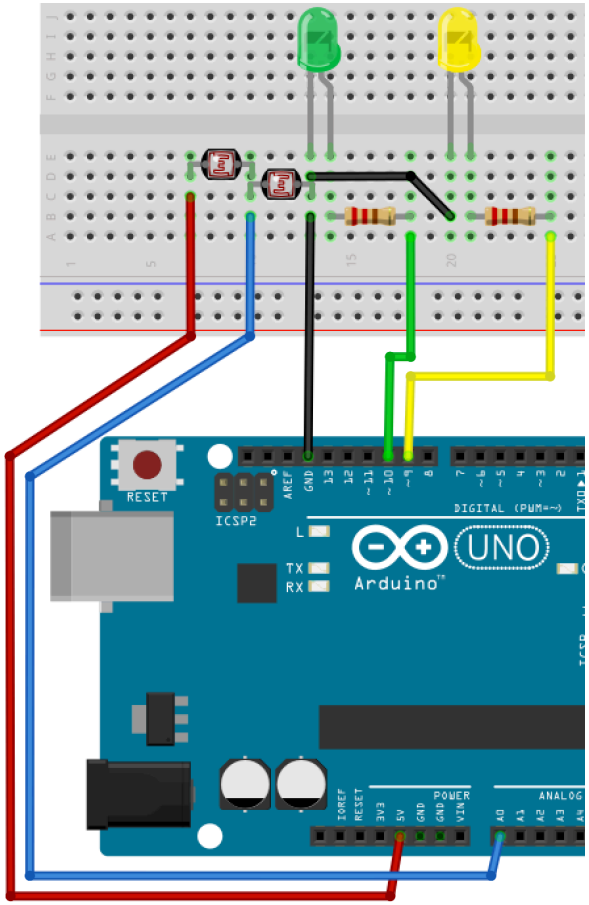
$V_{out} = 5 * \frac{10000}{5000000 + 10000} \approx 0.01$	$V_{out} = 5 * \frac{10000}{200 + 10000} \approx 4.90$
Dark	Light

Two variable resistors

Now for the real exercise. Imagine both R1 and R2 to be variable resistors. What would a circuit like that measure?

1. First of, try to imagine, given what you know about voltage dividers what would happen if both R1 and R2 are LDR's. What would that circuit measure? And in what way?
2. Try to theoretically verify this by using the formula on the previous page. Do this by calculating some test cases: One LDR dark, the other one dark, both dark and both light.
3. Build the circuit shown in the image and test if you were right. For this you can use the code from exercise 5 and the Serial Monitor/Plotter.

For your portfolio, make a video in which you demonstrate how this circuit works. In addition, explain the working of this circuit by referencing the formula and the calculations you made. You may add the explanation to your portfolio as text or as a voice-over in the video.



Exercise 9: 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 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. Upload the Arduino code below and copy the Processing code in a new processing sketch. Change the serial port where indicated in the Processing code and then run it. You can use the circuit of exercise 6 or 7 for this. Make sure the Serial Monitor is closed 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.

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

Arduino Code:

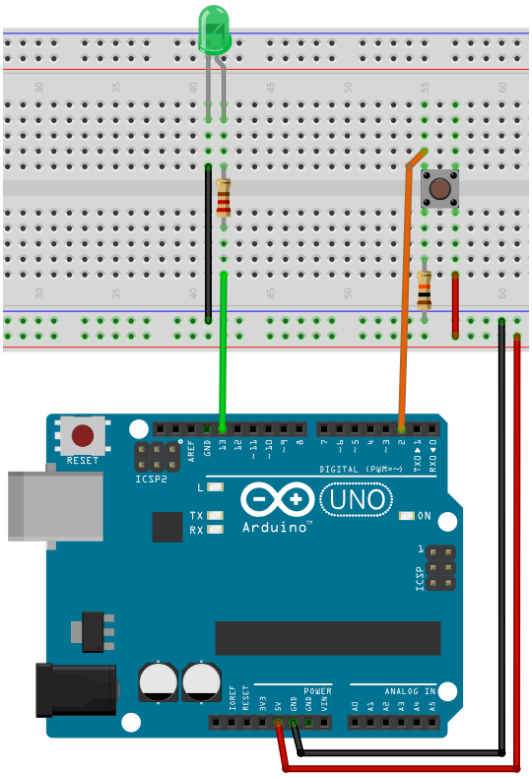
```
float sensorValue = 0;           // variable for sensor value
int sensorPin = A0;              // variable for sensor pin
int greenLedPin = 10;            // variable for green LED pin

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
                                        // 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 even
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 10: Buttons

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

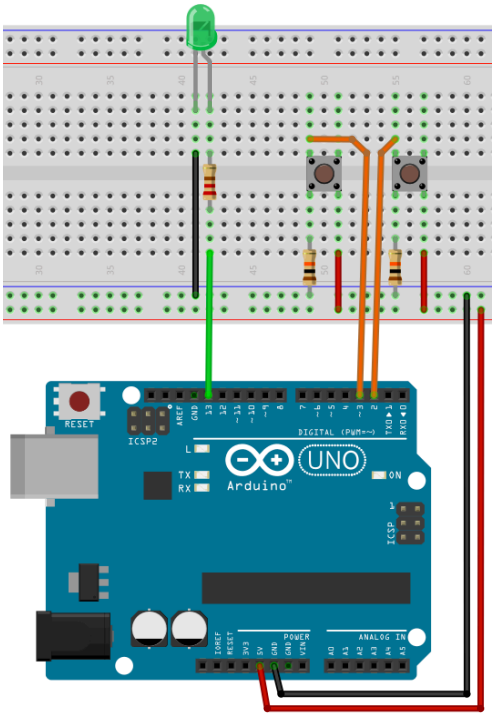
In the image on the site, you notice that there no LED is used. Therefore, use the circuit shown here.

Make sure the LED is oriented the correct way. The two resistors have different values. The resistor closest LED is 220Ω [red red brown gold] or [red red 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. So 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.



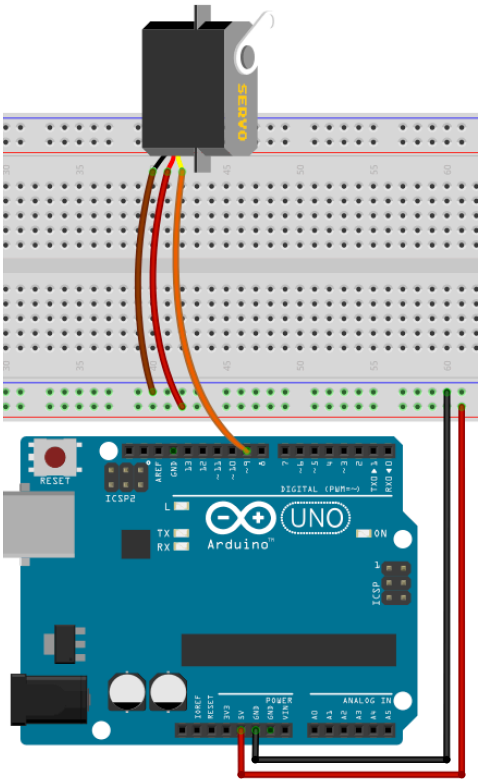
Now you can use the two buttons to interact with. Write 3 programs that work with this circuit.

D. The LED lights up when both buttons are pressed simultaneously.
E. The LED lights up when one button of both buttons are pressed.
F. The LED lights up when one button of the two buttons is pressed, but not both.

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 11: 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 completely. 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.

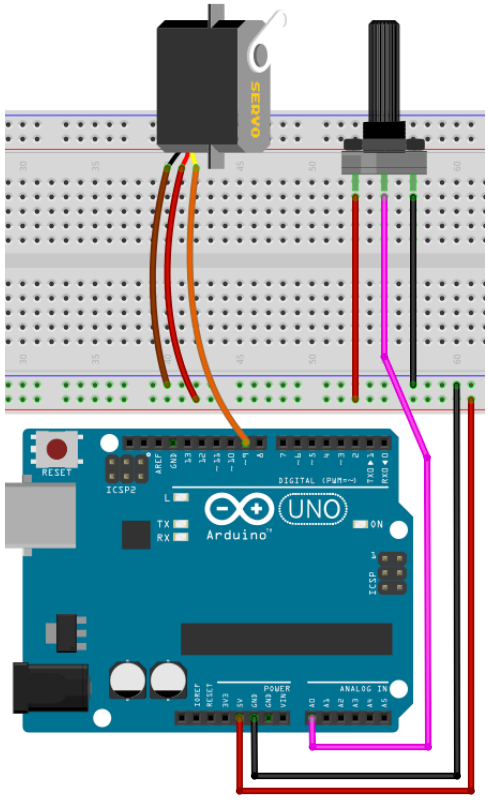
```
#include <Servo.h>
Servo myServo;                // Declare a servo object for us to control
int servoPin = 9;             // Var for the pin the servo connects to
int pos = 0;                  // Var to keep track of the servo's position

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)

Exercise 12: Controlling the servo with inputs

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.



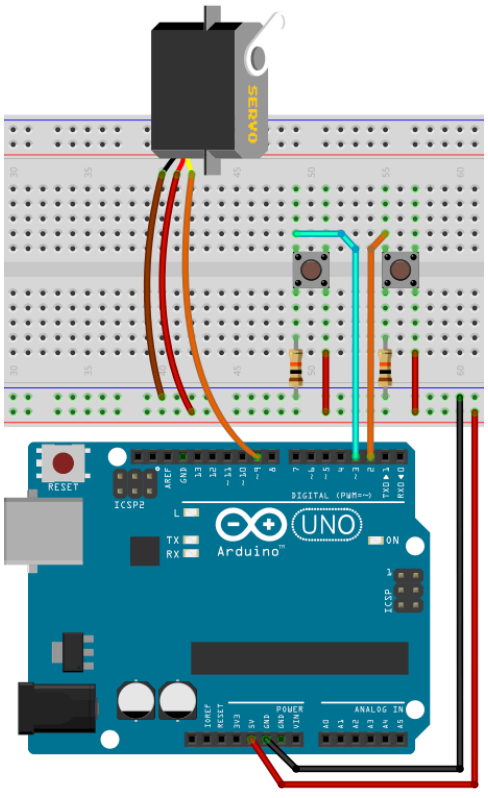
Servo with knob

A. 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.



Servo with buttons

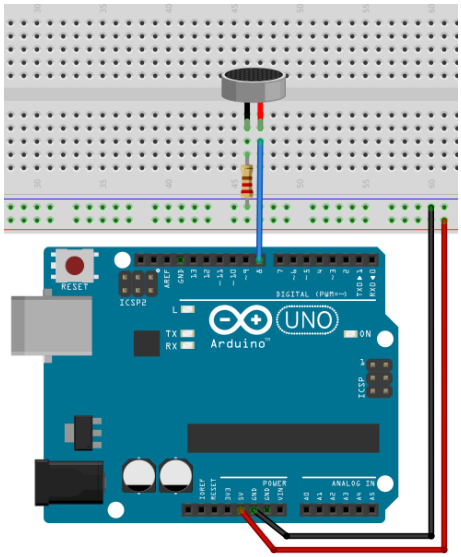
B. Build the circuit shown in the image.

This circuit contains two buttons. These buttons should control the servo. One button should move the servo clockwise and the other counter-clockwise.

A starting point for the code is given on the next page. Add to this code so that the buttons control the servo. Make sure the servo can never move out of the 0 - 160 range. So, the variable **pos** can never be smaller than 0 or greater than 160.

```
#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

int buttonPinL = 3;  // Var for the pin of the left button
int buttonPinR = 2;  // Var for the pin of the right button


void setup() {
  myServo.attach(servoPin);   // Tell the servo to what pin it's connected to
  pinMode(buttonPinL, INPUT); // Set the left button pin to input
  pinMode(buttonPinR, INPUT); // Set the right button pin to input
}


void loop() {

  ........

}
```

Exercise 13: 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 220Ω resistor [red red brown gold] or [red red 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. The code shown below plays a melody. Upload this code and test it with the same circuit.

```
int speakerPin = 8;

int length = 295; // the number of notes
char notes[] = "EE E CE G  g  C  g e  a b ia gEGA FG E CDb C  g e  a b ia gEGA FG E CDb  GNFR E uaC aCD GNFR E 1 11   GNFR
E uaC aCD L  D C   CC C CD EC ag  CC C CDE  CC C CD EC ag  EE E CE G  g  C  g e  a b ia gEGA FG E CDb C  g e  a b ia gEGA
FG E CDb EC g u aF Fa  bAAAGFEC ag  EC g u aF Fa  bF FFEDCe ec  "; // a space represents a rest
float beats[] = { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1.3, 1.3, 1.3, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2,
2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 2, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 4, 4, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1.3, 1.3, 1.3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1.3, 1.3, 1.3, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1.3, 1.3, 1.3, 1.3, 1.3, 1.3, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1,
2, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1.3, 1.3, 1.3, 1, 1, 1, 1, 1, 1, 10 };

int tempo = 95;

void playTone(int ton1, int duration) {
  tone(speakerPin, ton1, duration);
  delay(duration);
}

void playNote(char note, int duration) {
  char names[] = { ' ',  '!', '2', '3', '4', '5', '6', '7', '8', '9', '0', '-', '=', 'c', 'd', 'e', 'f', 'g', 'a', 'b',
'C', 'D', 'E', 'F', 'G', 'A', 'B', 'i', 'N', 'R', 'u',  '1', 'L', 'k'};
  int tones[] =  {   0, 1046, 138, 146, 155, 164, 174, 184, 195, 207, 220, 233, 246, 261, 293, 329, 349, 391, 440, 493,
523, 587, 659, 698, 783, 880, 987, 466, 740, 622, 415, 1046, 622u, 227};

  // play the tone corresponding to the note name
  for (int i = 0; i < 34; i++) {
    if (names[i] == note) {
      playTone(tones[i], duration);
    }
  }
}

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

void loop() {
  for (int i = 0; i < length; i++) {
    if (notes[i] == ' ') {
      delay(beats[i] * tempo); // rest
    } else {
      playNote(notes[i], beats[i] * tempo);
    }
    delay(tempo / 2);
  }
}
```