senseBox:edu

Dokumentation

Open Educational Resources

Table of Contents

senseBox:edu

The senseBox:edu is a toolbox which helps teach programming to students and junior researchers in a playful manner.

The Arduino-based set of electronic components is used to build increasingly complex circuits, which are controlled via a microcontroller.

Along with the learning resources provided on these pages, young students can gather experience in (Arduino) programming and electronic circuits with hands-on training.

About this book

In this book, all the resources regarding senseBox:edu may be found. This includes tutorials, exercises, example projects, as well as links to downloadable material.

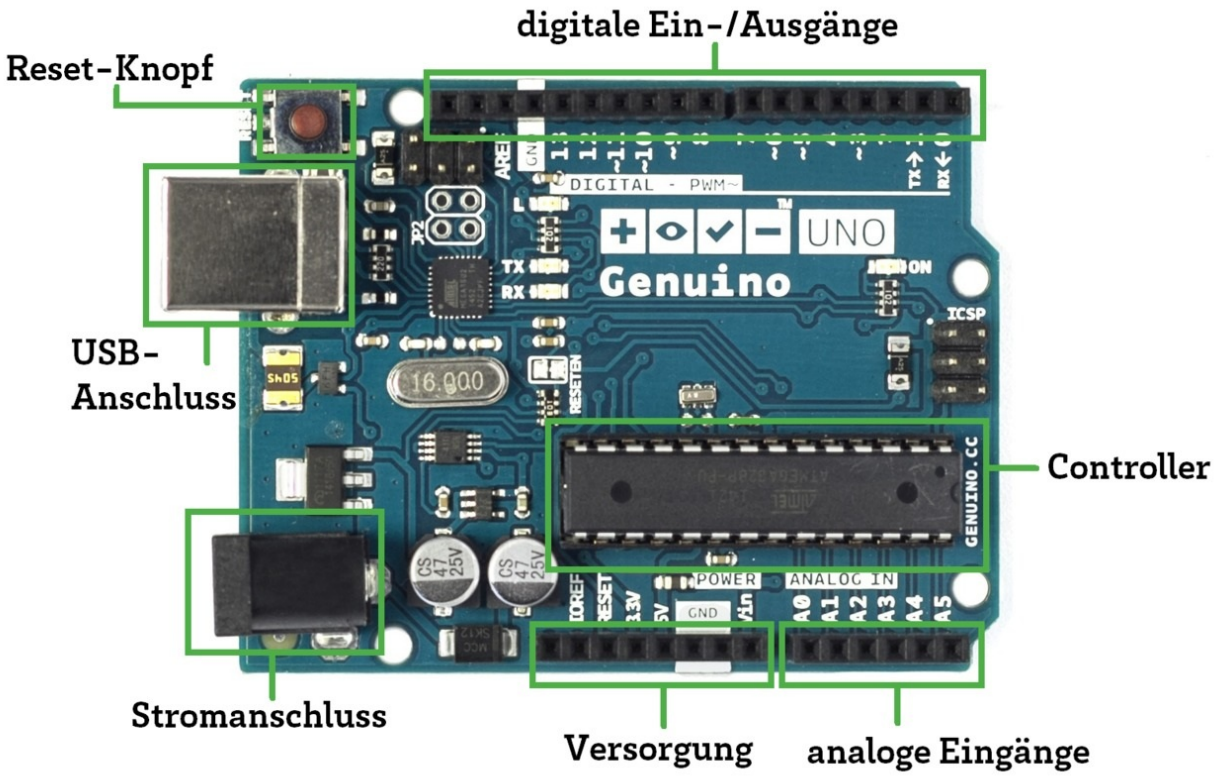
Contributions (improvements to existing content, or entirely new content) are appreciated! We're happy to include documentation you have made for projects using senseBox in this book. To do so, have a look at our contribution guide.

The source code for this book is available on GitHub, where you may also leave feedback about this book.

If you'd like to download this book as a PDF document for printing, check out our download section.

The Genuino Board

The Genuino UNO is a microcontroller board that is especially designed for developing prototype circuits. Besides the Genuino UNO, there are a lot of other microcontroller boards on the market.



Above you can see the Genuino UNO and its main components, which we will introduce in the following sections. The other components are important as well, but we would like to focus on the main parts.

USB connection

To transfer the programs you've written on your computer to the Genuino UNO microcontroller, we use the USB connection. In addition to data transfer, the USB connection can also work as a power supply when there is no other power source.

Power supply

As you may have noticed, this connection provides the microcontroller with power. The Genuino UNO runs on 5V. When using an external power supply, however, 6V - 20V are recommended.

Reset button

The Genuino UNO reboots after pressing the reset button. This means that the sketch will reset and start from the beginning. When you are unsure whether the Genuino UNO is running properly, you can press the reset button.

Digital input and output

To communicate with digital sensors and other components, you can connect to the digital pins. This enables you to perform many actions including reading sensor values or prompting LEDs to blink. Furthermore, it is also possible to output analog signals with the digital pins. The difference between analog and digital signals is explained in the following chapters.

Analog input

Similar to digital pins, analog pins can read out sensors and provide data to the Genuino UNO. The microcontroller does not support analog output. This function is provided by the digital pins.

Power supply pins

The power supply pins are neither digital nor analog pins. They provide power to the connected components. Furthermore, it is possible to provide the Genuino UNO itself with power through several pins.

Controller

The controller is the brain of the Genuino UNO. It executes all processes.

Software installation

To program the Arduino, you need to install the Arduino IDE and drivers. Depending on your OS, there are different ways to install the software.

You can find the latest installation files here.

Installation on Windows

After downloading the Arduino installation file, you just need to run it by double clicking on it. Follow the instructions in the setup menu.

Installation of the drivers on Windows XP/Vista/7

After installing the Arduino software, the main drivers will be installed on your computer. The software can have problems, however, with detecting the right driver for the connected board. You need to do the following steps:

1. Connect the Arduino UNO to your computer with the USB cable
2. The computer tries to detect and install the correct driver. This does not always work properly. We need to install the driver manually.
3. Open system preferences (Start → System preferences)
4. Go to System and Secruity → System → Device Manager
5. Search for Arduino UNO / Genuino UNO or unknown device. Right-click on it and choose the option to install driver software manually.
6. Search driver software on the computer and choose the folder for the arduino software. In this folder, there is the folder Driver. Choose this folder and confirm your selection.
7. Windows will install the correct driver and your Arduino UNO should work properly.

Installation on macOS

After downloading the files (see above) you can install them and copy them to your applications folder. It is not necessary to install drivers on macOS.

Installation on Linux

After downloading the files (see above) you need to extract the `.tar.xz` archive. You can run the installation script `install.sh` with the following command from the teminal:

```
cd <path to extracted folder>
./install.sh
```

In a desktop environment you can start the installation with a double-click on the `install.sh` file.

Afterwards there will appear a shortcut to run the Arduino IDE.

Arduino IDE

The Arduino IDE is the programming interface we use for our microcontroller. The white editor window represents the area for your programming code. In the black area, status and error messages will be displayed. The Arduino IDE is able to recognize misspellings and errors in your code's syntax, however logic errors can not be recognized.

In the upper toolbar of the software, you can find the most important elements to control the software:

- The checkmark is used to check your program's code if there are any mistakes.
- The arrow is used to upload your sketch to the Arduino.
- You can start a new sketch, open an existing one, or save your current sketch
- By using the magnifying glass, you can open the serial monitor. For further information about the serial monitor, check here

DIY Eco Station

In this project we will learn how to build a senseBox environmental station. At the end we will be able to measure various environmental phenomena such as temperature, humidity, light and air pressure, and the data will be published on the openSenseMap!

This project is the most extensive, therefore it was divided into several chapters. In each new chapter, an additional module is inserted until a full - senseBox: home like - Environmental Station is built!

DIY – Experiments with light

If we watch television, turn on the radio, write a message with our smartphone, or heat up food in the microwave, we are using electromagnetic energy. Today, all people are constantly dependent on this energy. Without it, life as we know it in modern cities would be completely different.

Aim of the station

In this station we are using a light sensor to detect the illuminance of visible light in lux.

Materials

- Light Sensor `TSL 45315`

Basics

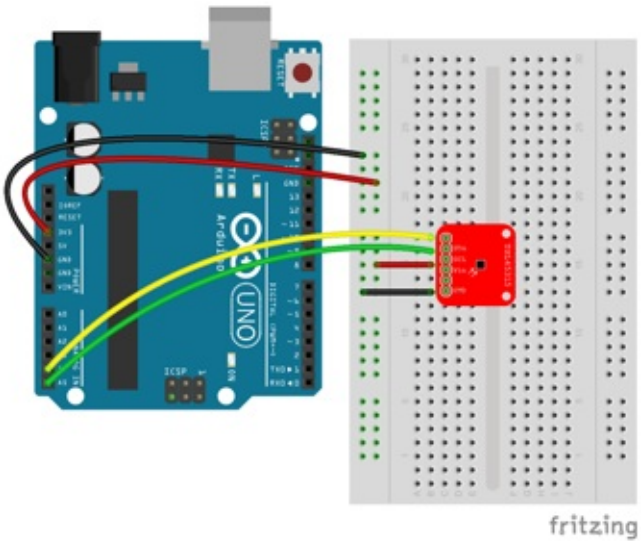
Electromagnetic energy moves in waves through space. These waves range from very long radio waves to very short wavelength gamma rays. The human eye can only perceive a very small part of this spectrum: visible light. Our sun is the source of energy over the entire spectrum. The Earth's atmosphere protects us from exposure to high levels of radiation that could be dangerous for us.

Here, the intensity of visible light is of main interest. To measure the illuminance of the incident light in the visible part of the spectrum, the unit lux is used. It indicates the ratio of the brightness in lumens per square meter. On a bright sunny day, the light is measured at about 100,000 lux, and for a full moon at night the illuminance is measured at only about 1 lux.

For this measurement you use the I2C sensor TSL45315 of AMS-TAOS. In the sensor datasheet you can see that its sensitivity is matched to the visible part of the light spectrum, which is approximately between 400 and 700 nm.

According to the data sheet, the sensitivity of this sensor reaches from 2 to 200000 Lux, with a resolution of 3 lux. Furthermore, the sensor needs a power supply of 3.3V.

Construction



Basics

We will use the library `Wire` . In the beginning we need a few constants that are defined with the directive #. Unlike variables, they occupy a permanent place in the store. In our case, the bus address and the following register addresses of the sensor are to be saved.

ADDRESS	RESISTER NAME	R/W	REGISTER FUNCTION	RESET VALUE
--	COMMAND	W	Specifies register address	0x00
0x00	CONTROL	R/W	Power on/off and single cycle	0x00
0x01	CONFIG	R/W	Powersave Enable / Integration Time	0x00
0x04	DATALOW	R	ALS Data LOW Register	0x00
0x05	DATAHIGH	R	ALS Data HIGH Register	0x00
0x0A	ID	R	Device ID	ID

These registers are used to configure and for communication: ` Arduino

include

define I2C_ADDR (0x29)

define REG_CONTROL 0x00

define REG_CONFIG 0x01

define REG_DATALOW 0x04

define REG_DATAHIGH 0x05

define REG_ID 0x0A

`

In the setup function you will connect the sensor and start the transmission.

`Arduino Wire.begin (); Wire.beginTransmission (I2C_ADDR); Wire.write (0x80 | REG_CONTROL); Wire.write (0x03); // Power on Wire.endTransmission (); `

Next, we will set a fixed exposure time of 400 ms:

`Arduino Wire.beginTransmisson (I2C_ADDR); Wire.write (0x80 | REG_CONFIG); Wire.write (0x00); // 400 ms Wire.endTransmission (); `

To change the shutter speed, you can change the corresponding value of ` 0x00` in 0x01 `or` 0x02 ` to reduce the exposure time to 200 or 100 ms in the configuration register of the sensor. In the Loop function, we give the order to start the measurement routine and let the sensor send the data needed to calculate the illuminance:

`Arduino Wire.beginTransmisson (I2C_ADDR); Wire.write (0x80 | REG_DATALOW); Wire.endTransmission (); Wire.requestFrom (I2C_ADDR, 2); Request // 2 bytes uint16_t low = Wire.read (); uint16_t high = Wire.read (); `

If the sensor still sends data, these should be caught afterwards, to avoid errors in the next passage.

`Arduino while (Wire.available ()) { Wire.read (); } `

Finally, you will use the data to calculate illuminance in lux. In the data sheet, there is the matching formula:

```
`Arduino uint32_t lux; lux = (high << 8) | (Low << 0); lux lux = * 1; // Multiplier for 400ms `
```

To adjust this formula to an exposure time of 200 or 100 ms, you can increase the multiplier to 2 or 4.

Exercise 1

Adapt this lesson's code to create a custom function that can print the sensor data to the Serial Monitor.

Exercise 2

Change the exposure time of the sensor and then compare the results of the measurements.

Tip: Don't forget to adjust both the lux value and exposure time in the configuration register accordingly.

Traffic Counter

Goal

Building a functioning traffic counter. For this purpose we use an ultrasonic distance sensor. The recorded values will be shown in the Serial Monitor.

Materials

- Arduino Uno with Breadboard
- Ultrasonic distance sensor

Additional materials

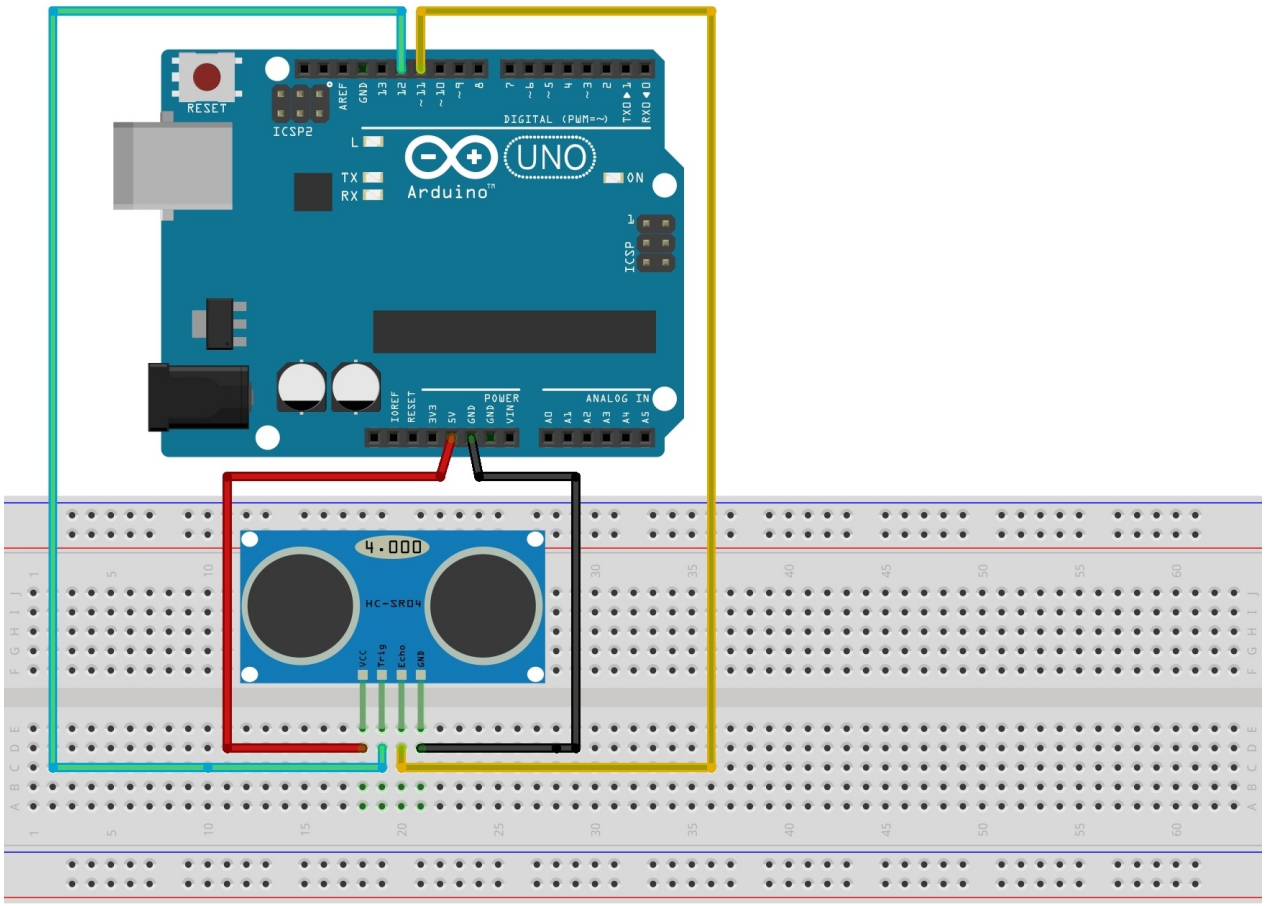
- No additional materials.

Basics

The ultrasonic distance sensor uses sound to calculate the distance of objects. The sensor sends out a pulse and measures the time until it receives the echo of the pulse. The distance is calculated by using the speed of sound and the meassured time.

Construction

The ultrasonic sensor will be connected to four different ports on the Arduino. For the power supply connect the VCC pin with the 5V port on the Arduino. To close the circuit, the GND pin has to be connected to the GND port of the Arduino. Finally the echo and the trigger pin of the sensor have to be connected to two different digital ports of the Arduino. (e.g. 12 and 11)



Programming

Define the pins that are connected to the sensor as usual. In addition to that we need to define two variables to safe the measured time and the calculated distance.

`` `Arduino int trig = 12; // Trig pin of the sensor is connected to Pin 12 int echo = 11; // Echo-pin of the sensor is connected to Pin 11 unsigned int time = 0; unsigned int distance = 0; ` ``

In `setup ()` we must now start the Serial Monitor . We then have to define which pin will be input and which one output. The trigger pin of the sensor must be defined as output and the echo pin as input.

`` ` `` Arduino Serial.begin (9600); pinMode (trig, OUTPUT); pinMode (echo, INPUT);

`` ` ``

In `loop ()` we execute `` `Arduino digitalWrite (trig, HIGH); delay microseconds (10); digitalWrite (trig, LOW); `` `a 10 microseconds long ultrasonic pulse. The subsequent command` time = pulseIn (echo, HIGH); `saves the time to catch the echo into the variable` Time `` ` ``. Finally the distance to the car must be calculated. To do this we use the variable 'time'. The result we than display on the Serial Monitor.

`` `Arduino distance = time / 58; Serial.println (distance); ` ``

Task 1

Try to build a traffic counter.

Notice:

- Try to evaluate a specific distance range, to prevent disturbances by movements in the background.

- To avoid double counting of a stationary vehicle we should program a condition which stops counting until the lane is free again.

Traffic light

Goal

We will simulate a traffic light which will be startable from a button.

Materials

From senseBox: edu

- Arduino Uno with Breadboard
- Red LED
- Yellow LED
- Green LED
- 3x 470Ω resistor
- button
- 10kΩ resistor

Setup Description

Hardware configuration

! [Ampel-button-wiring diagram]
(https://raw.githubusercontent.com/sensebox/resources/master/images/edu/ampel_button_schaltplan.png)

Software Sketch

` Arduino int red = 13; int yellow = 12; int green = 11;

int button = 8;

void setup () { pinMode (red, OUTPUT); pinMode (yellow, OUTPUT); pinMode (green, OUTPUT);

pinMode (button, INPUT);

// set of traffic lights first to RED digitalWrite (red, HIGH); digitalWrite (yellow, LOW); digitalWrite (green, LOW); }

void loop () {

// Check if button is pressed if (digitalRead (button) == HIGH) {

```
  delay (5000);

  // RED to GREEN
  digitalWrite (red, HIGH);
  digitalWrite (yellow, HIGH);
  digitalWrite (green, LOW);

  delay (1000);

  digitalWrite (red, LOW);
  digitalWrite (yellow, LOW);
  digitalWrite (green, HIGH);

  delay (5000);


  // GREEN to RED
  digitalWrite (red, LOW);
  digitalWrite (yellow, HIGH);
  digitalWrite (green, LOW);

  delay (1000);


  digitalWrite (red, HIGH);
  digitalWrite (yellow, LOW);
  digitalWrite (green, LOW);
```

}}

`` ` ``

- At the beginning of the `loop ()` -function we check if the start button is pressed.
- `DigitalRead (button)` reads the current state of the button.

The first sound - using a beeper

Until now, our senseBox has been silent. In this project, we will change this!

Aim of the Project

For the first step, we simply want to get a sound from the beeper. Next we want to change the volume of the beeper. Finally, the beeper should play a simple melody. While the first two steps can be quickly achieved, the third step is a little trickier.

Materials

- Beeper
- Potentiometer

Basics

A Beeper, or a Piezo, is a component that converts electrical signals into sound. The volume can reach up to 80dB. The beeper has two pins where it can attach to the plug-in board. The operating voltage of the beeper is between 1V and 12V, and it consumes up to 19mA. Similar to the LED, the beeper's electricity is only able to flow in one direction. The shorter pin must be connected to the grounding (GND) source and the longer pin must be connected to the voltage source.

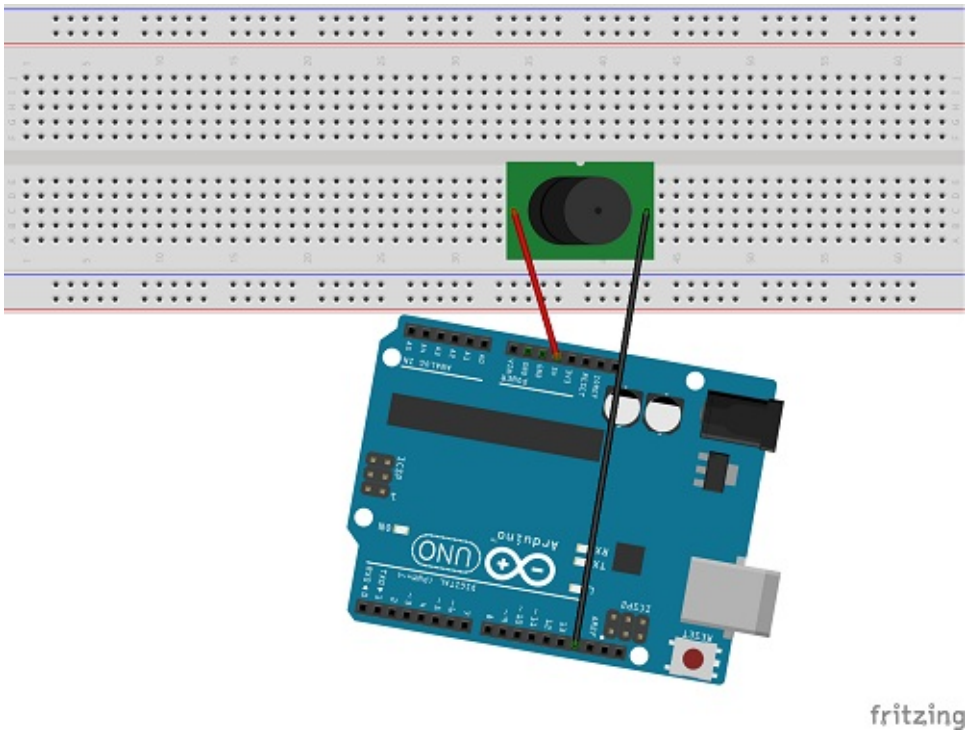
The Potentiometer is an electrical component whose resistance value can be continuously adjusted. Its settings can be adjusted by moving a grinder over the resistor body. Usually a potentiometer has three pins: two for the resistance and a third for the tap. Our potentiometer has a maximum resistance of 10k ohms.

Warning Small potentiometers are designed only for a small current flow. For the electrical components in senseBox, this potentiometer is sufficient. If you connect components with greater power consumption (for example, a servo motor) however, you need a larger potentiometer.

Construction

Step 1:

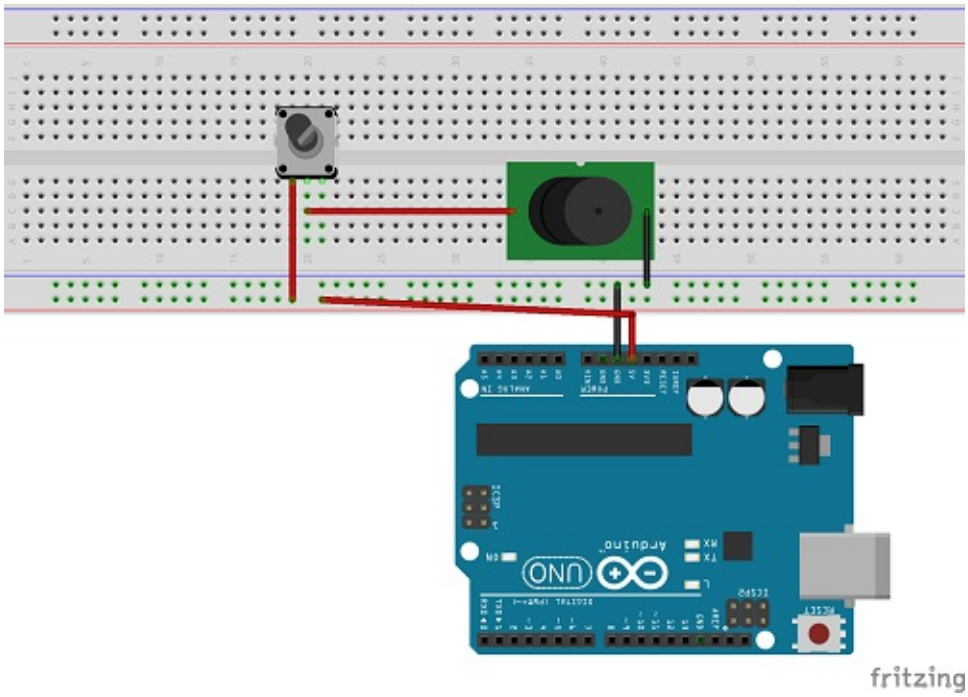
If you build the circuit as shown in the graph and connect the Arduino to the power supply, the beeper should produce a loud sound. By doing this we have already completed our first step.



Step 2:

Now we'd like to integrate another component into our circuit that will enable us to change the volume. We will use a potentiometer to control the beeper's volume, similar to the volume control knob that is seen on older radios.

Next, connect the `5V` output of the Arduino and the long pin of the beeper to the potentiometer. Now you are ready to change the volume using the potentiometer!



Step 3:

A single continuous sound is not really exciting - our beeper is capable of more! To produce different tones, we can use special outputs from the Arduino that are able to output pulse-width modulation. For more information on pulse-width modulation (PWM), look here.

These pins are marked with the sign ~: The included pins are 4, 5, 6, 9, 10 and 11. A beeper produces a specific sound for each pulse width. Every tone on the scale (scale: c, d, e, f, g, a, h, c) receives a pulse width. We will use the construct `# define` as follows:

```
#define h 4064 // 246 Hz
#define c 3830 // 261 Hz
#define d 3400 // 294 Hz
#define e 3038 // 329 Hz
#define f 2864 // 349 Hz
#define g 2550 // 392 Hz
#define a 2272 // 440 Hz
#define h 2028 // 493 Hz
#define C 1912 // 523 Hz
#define E 1518 // 659 Hz
#define F 1432 // 698 Hz
#define R 0 // define a note as substitute for a pause
```

Now we need some variables to control the playback behavior of the Arduino. Later you can try different values and check how this affects the melody:

```
// Set overall tempo
long tempo = 26000;
// Set the pause length between notes
int pause = 1000;
// Loop variable to increase residual length
int rest_count = 50;
```

Now we need some global variables that are used by the playback functions, and we will define the `setup` :

```
// Initialize core variable
int tone = 0;
int beat = 0;
long duration = 0;
int speakerOut = 9;

void setup () {
  pinMode (speakerOut, OUTPUT);
}
```

Now we can write our melody using an array. Another array `beats` is defined, representing how long the corresponding note should be played in `melody` :

```
int melody [] = {G, E, R, R, R, e, f, g, e, e, C}; // Example melody
int beats [] = {8, 8, 8, 8, 8, 8, 8, 8, 16, 16, 32};
```

Later you can insert your own melody here.

We will next write a help method, which plays one tone of our melody. For this purpose, it will check the first `if` statement and evaluate whether it is a tone or a pause. If it is a tone, the tone will be played in a loop for a certain `duration` measured in milliseconds:

```
void playtone () {
  long elapsed_time = 0;
  if (tone> 0) {// if this is not a radical beat
    while (elapsed_time <duration) {

      digitalWrite (speakerOut, HIGH);
      delayMicroseconds (tone / 2);

      // DOWN
      digitalWrite (speakerOut, LOW);
      delayMicroseconds (tone / 2);

      // Keep track of how long we pulsed
      elapsed_time + = (tone);
    }
  }
  else {// Rest beat;
    for (int j = 0; j <rest_count; j ++) {
      delayMicroseconds (duration / 2);
    }
  }
}
```

After our sound has been played, it's time to play an entire melody with another helper method. We will now implement a method for playing the whole melody. For this is a `for` loop is written to go through the `melody` array, and to retrieve the helper function `playtone ()` for each entry, which we have defined above. In addition to each tone, a short pause, or delay, is inserted.

```
int MAX_COUNT = sizeof (melody) / 2; // Number of tones

void play melody () {

  for (int i = 0; i <MAX_COUNT; i ++) {
    tone = melody [i];
    Beat = beats [i];

    duration = * beat tempo; // Set up timing

    playtone ();

    delay microseconds (pause);
  }
}
```

Now we must include the main loop, which controls the flow of the program:

```
void loop () {
  play melody ();
}
```

Idea: If you would like to include higher or lower notes in your tune, you can define them similar to what we have done in the example above. [here] (http://www.phy.mtu.edu/~suits/notefreqs.html) you can check how much Hertz a tone has.

Warning All variables defined in the program must have a unique name !

Community Projects

In this section community-contributed projects are documented.

You may also have a look on german project tutorials in the german section of this book.

If you want to contribute a project documentation yourself, have a look on our contribution guide.

Project Title	Author	Date	additional material required
Mobile Sensor Logger	Michelle Gybel & Johannes Schöning (Hasselt University)	07.07.2016	no
Heatmap	Michelle Gybel & Johannes Schöning (Hasselt University)	08.08.2016	no

DIGITAL CITIZENS, CONNECTED COMMUNITIES:

THE ROLE OF SPATIAL COMPUTING AND VOLUNTEERED GEOGRAPHIC INFORMATION

Tutorial: Mobile Sensor Logger

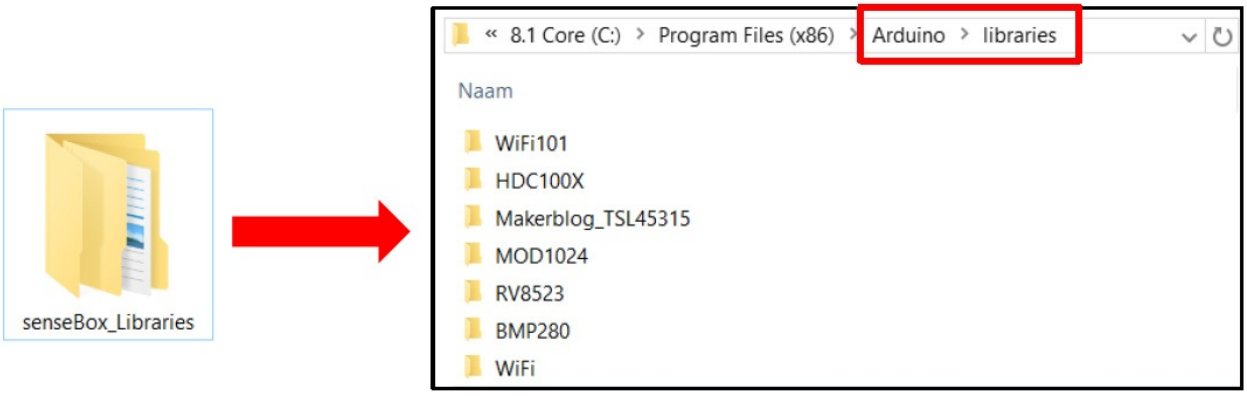
Authors: Michelle Gybel and Johannes Schöning of Hasselt University

Edited by Felix Erdmann using pandoc

This tutorial is a step-by-step guide which will teach you how to build mobile sensor logging device using the senseBox Edu kit.

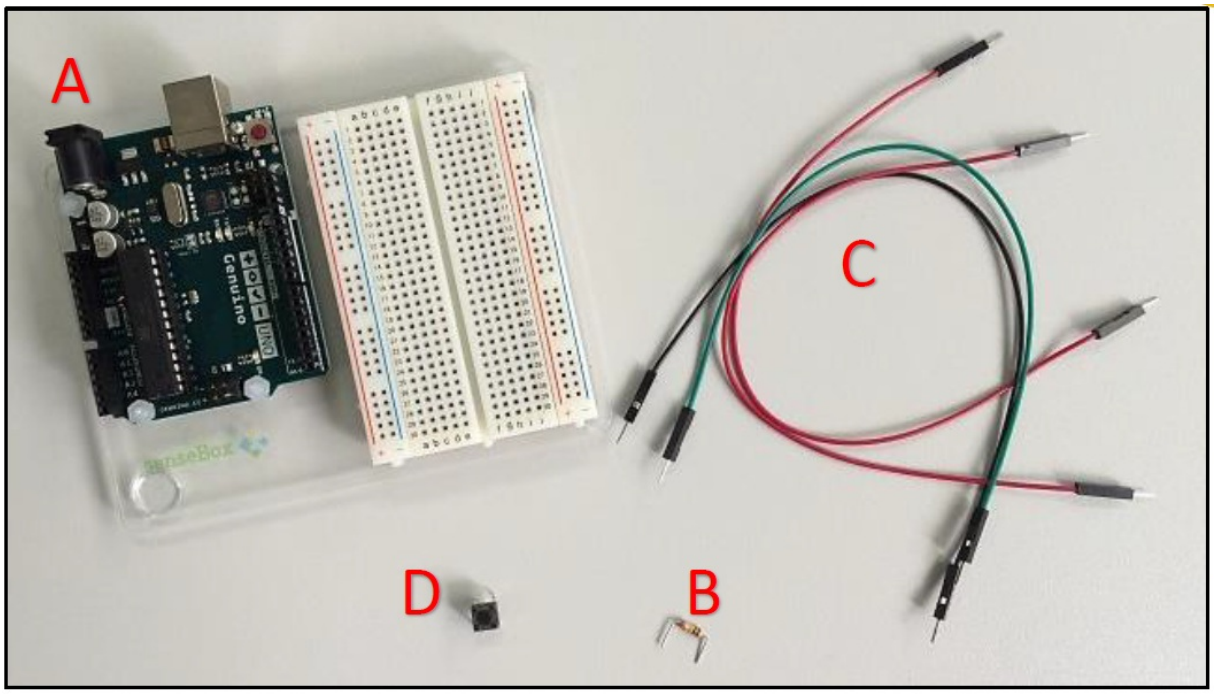
Preparation

1. Install Arduino
    - Download from here
    - (Extra:) Tutorials and Reference Guides can be found here
2. Install senseBox Plugins
    - Download from here
    - Extract the folder and copy the content to the `libraries/` folder in Arduino's installation files.

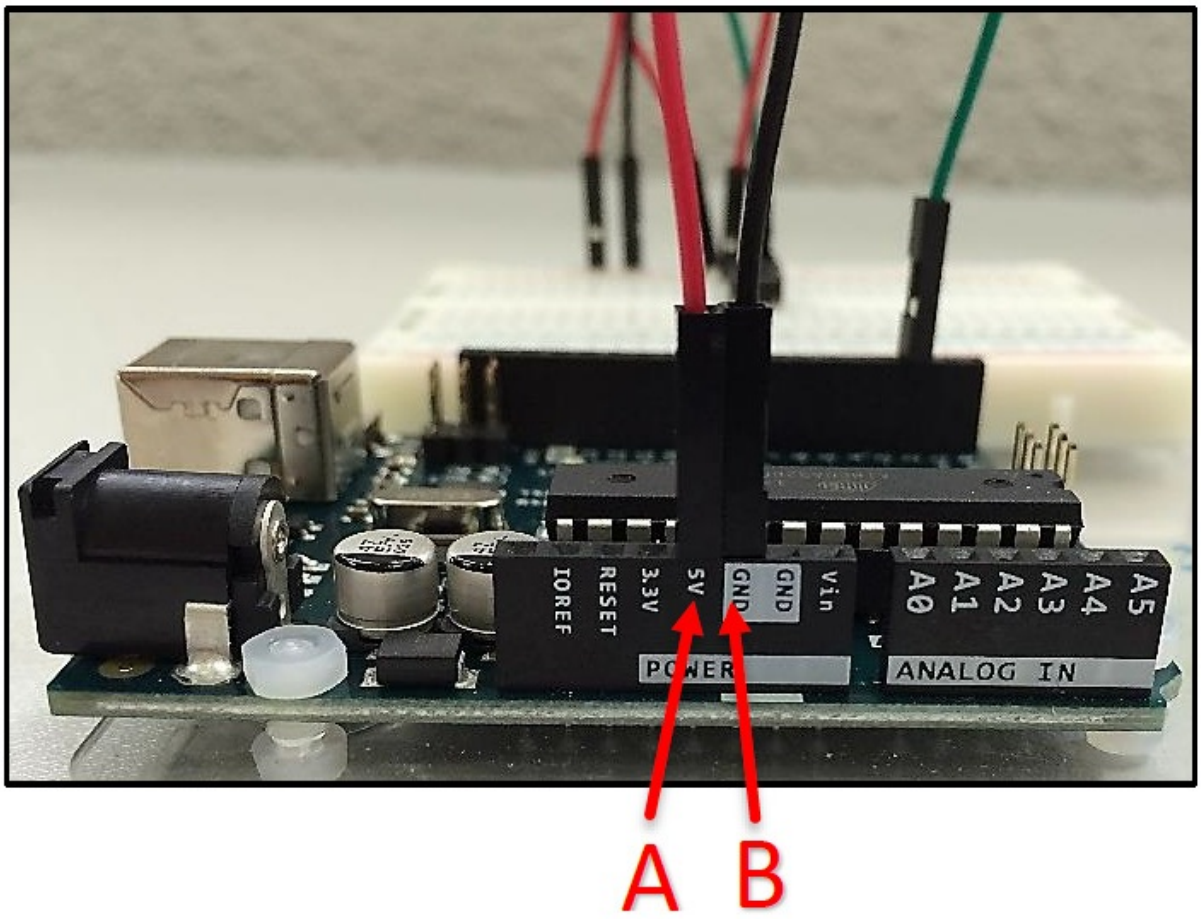


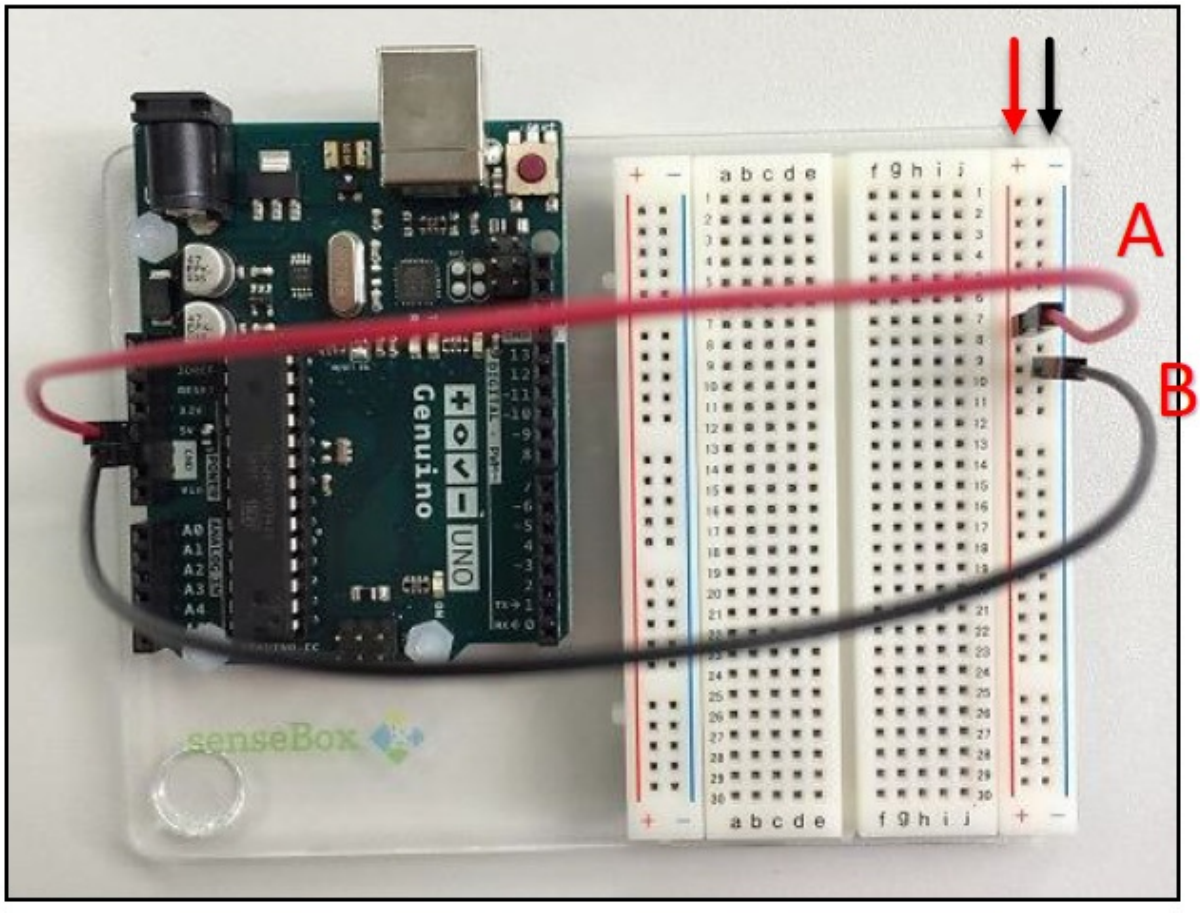
Start with the basics... How to Push a button!

1. What do you need?
    - Genuino Board with breadboard (A)
    - 10K ohm resistor (B)
    - Wires (C)
    - Pushbutton (switch) (D)



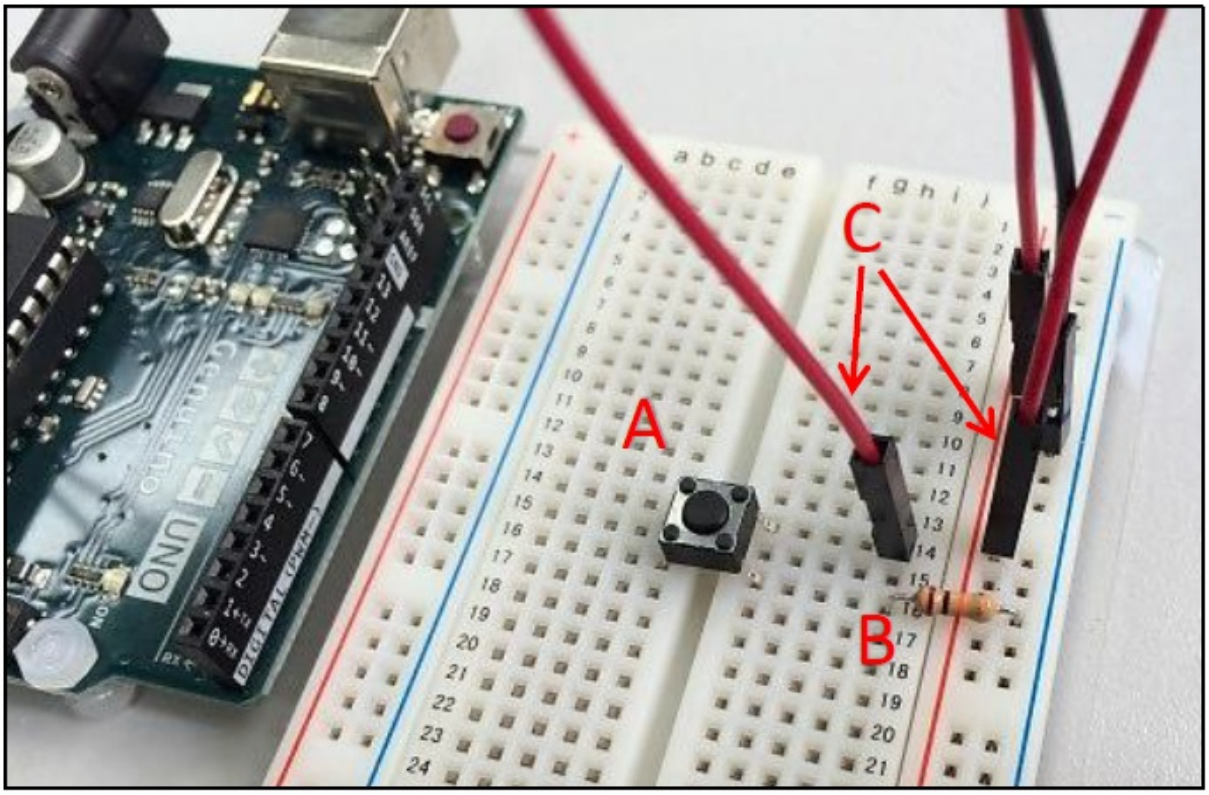
2. Provide voltage and grounding to the breadboard
    - Connect a red wire to the positive vertical row of the breadboard and to the 5 volt supply on the Genuino Board (A).
    - Connect a black wire to the negative vertical row of the breadboard and to the ground (GND) on the Genuino Board (B).



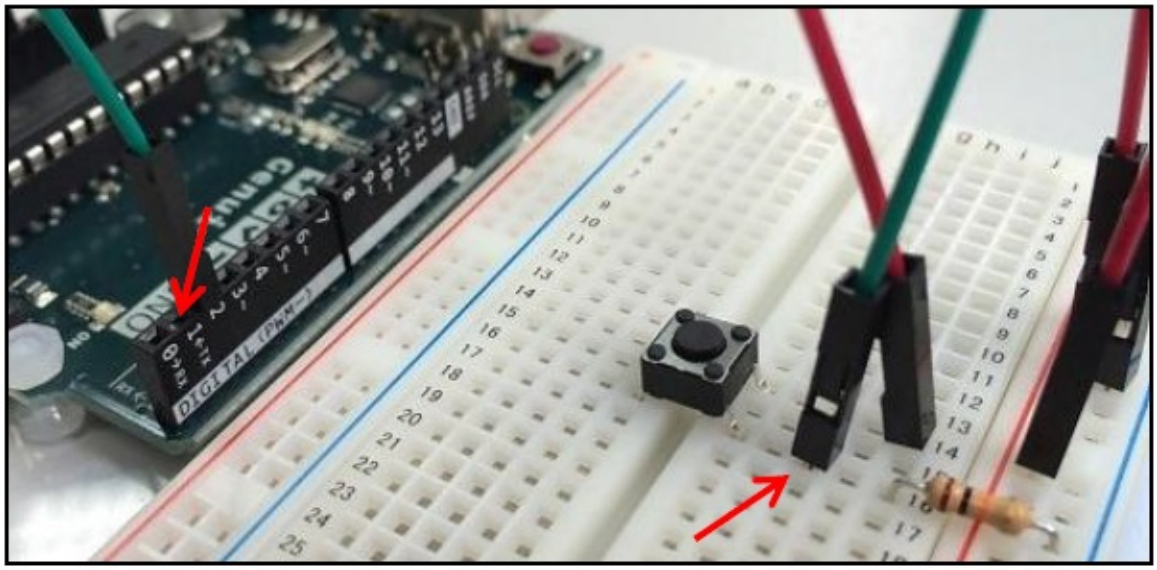


3. Add the pushbutton and provide it with power and grounding

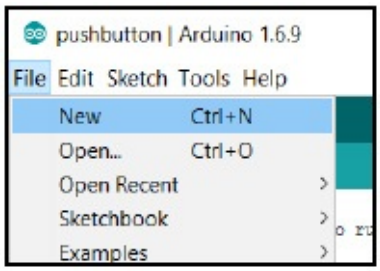
- Attach the pushbutton to the breadboard as presented below (A).
- Create a connection between one of the legs of the pushbutton and the ground with the 10K ohm resistor (B).
- Connect a red wire to the positive vertical row of the breadboard (C).



- Provide with a green wire a connection between digital pin 2 on the Genuino Board and the leg of pushbutton connected to the ground.

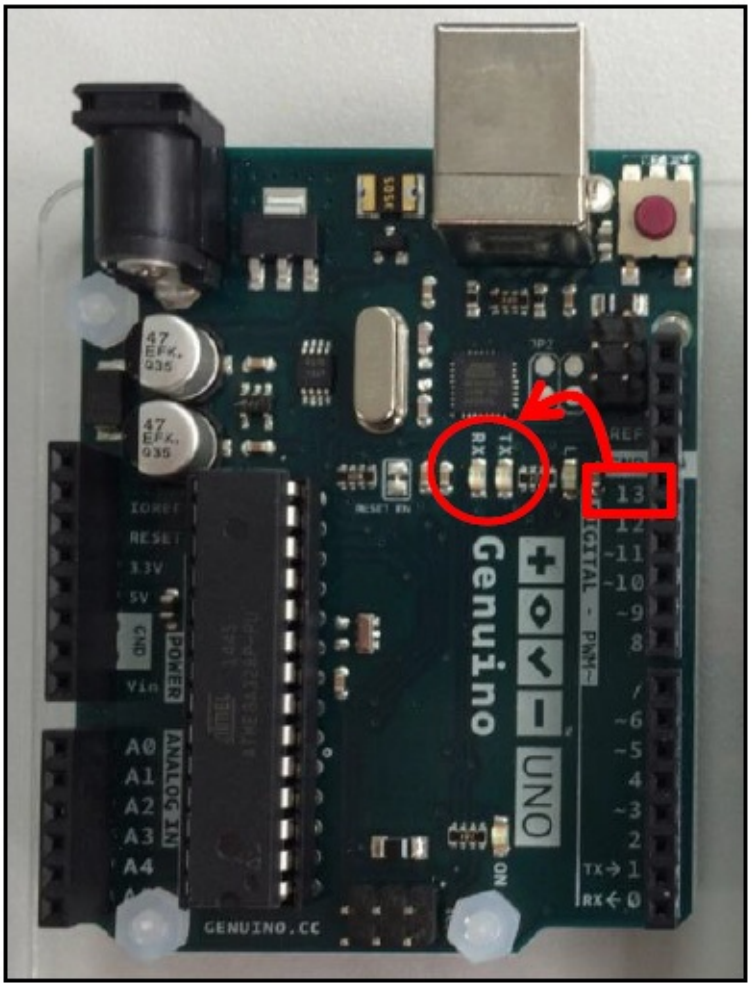


1. Turn on the light, ähm button... by writing some code
   - Start the Arduino IDE.
   - Programs written in the IDE are called sketches. Go to File New and name the file Pushbutton.ino or similar to create your first sketch.



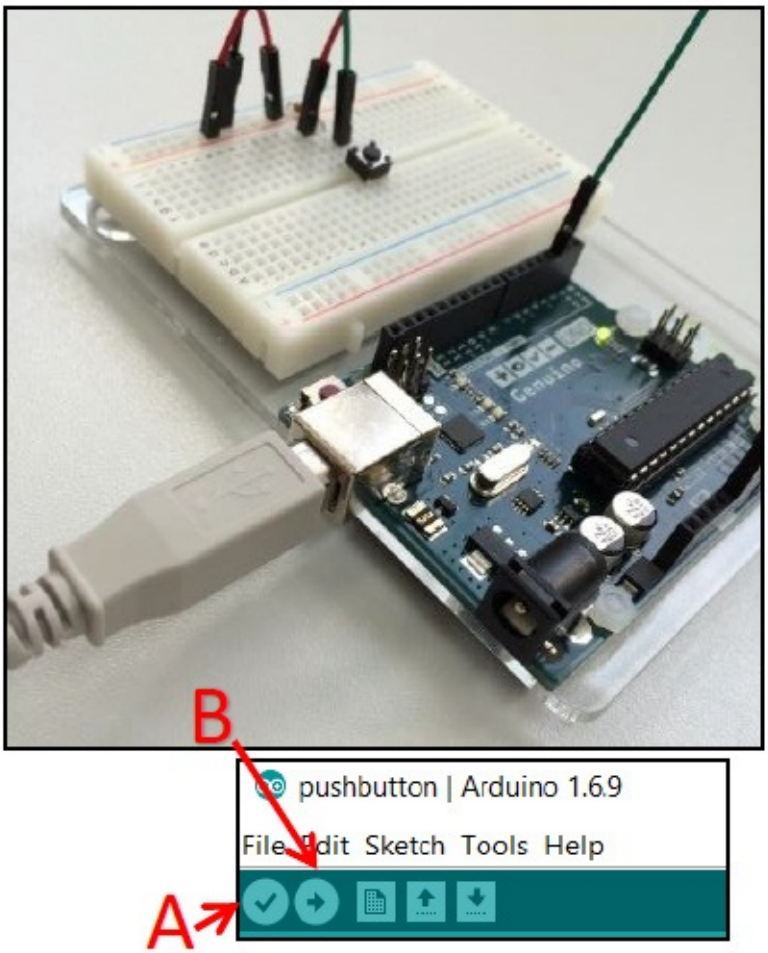
- Each sketch must contain a setup() and a loop() function:

- As a start you can may place the following code in your sketch. It initializes the which pins are connected to the LED light on the Genuino Board and the push button. Next, it provides the functionality to turn the LED light on when the button is pressed in. This LED is built-in in the Genuino board and driven by pin 13.




```
const int buttonPin = 2; // the number of the pin to which the pushbutton is connected
const int ledPin = 13; // the number of the pin connected to the LED
int buttonState = 0; // variable for reading the status of the pushbutton
void setup() {
  pinMode(buttonPin, INPUT); // initialize the pushbutton pin as an input
  pinMode(ledPin, OUTPUT); // initialize the LED pin as an output
}
void loop() {
  buttonState = digitalRead(buttonPin); // get the value of the pushbutton
  //check if the pushbutton is pressed in
  if (buttonState == HIGH){
  digitalWrite(ledPin, HIGH); // if the button is pushed, turn LED on
  } else {
  digitalWrite(ledPin, LOW); // if the button is released, turn LED off
  }
}
```

1. Compile and upload the sketch to the board
    - Connect the board to your computer with a USB cable.



    - Compile the code by clicking on the Verify button (A).
    - Click on the Upload button (B).

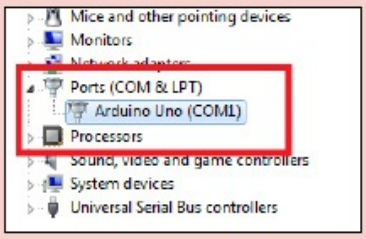
Help! I'm on Windows and I got an error!

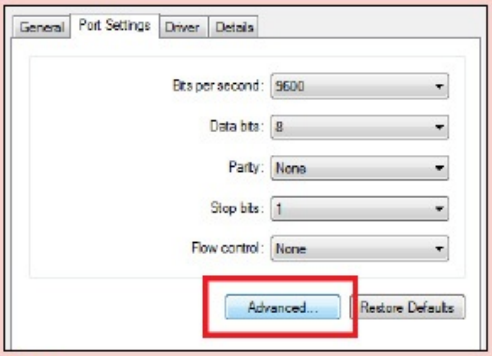
Windows users might get the following error while trying to upload their code:

AVRDUDE: SER_OPEN():SYSTEM CAN'T OPEN DEVICE "\.\COM1"...

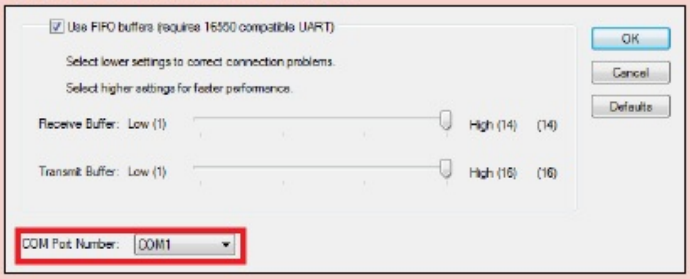
This can be solved by setting the Genuino Uno port to COM1:

- Open the **Device Manager.**
- Go to **Ports → Arduino Uno / Genuino Uno .**
- Open the **Advanced properties.**

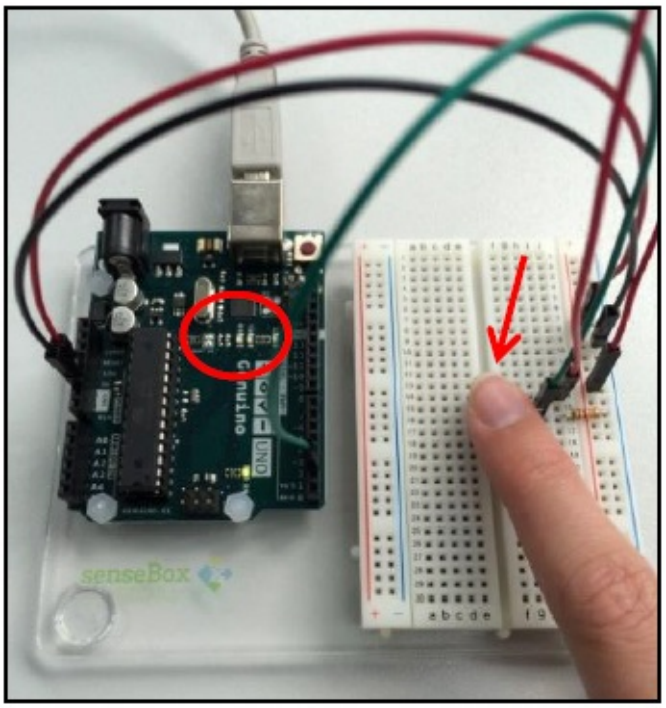




- Set the **COM Port Number** to **COM1.**

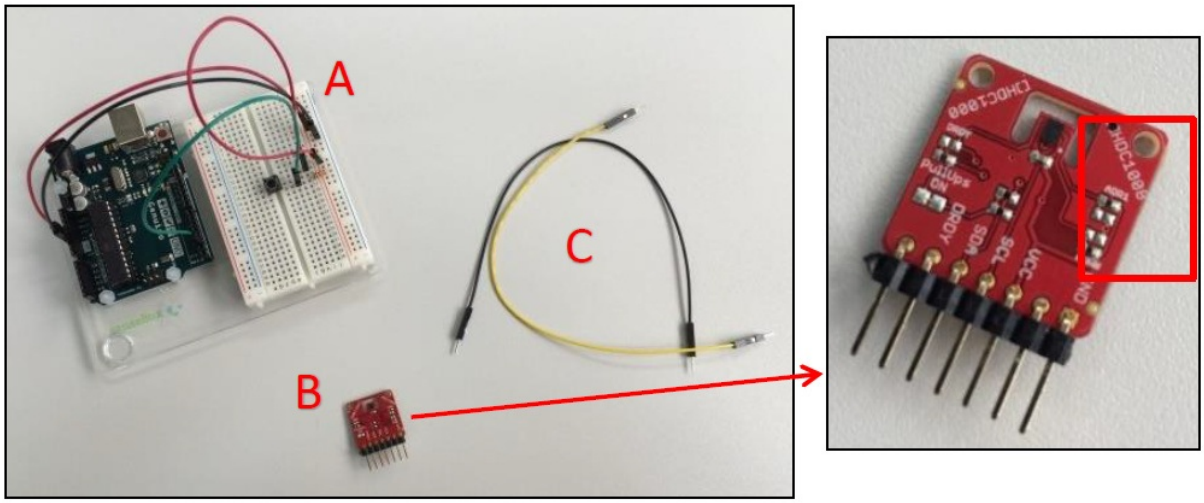


2. Test your program
    - Push the button on the board.
    - A LED light on the Genuino Board should light up.

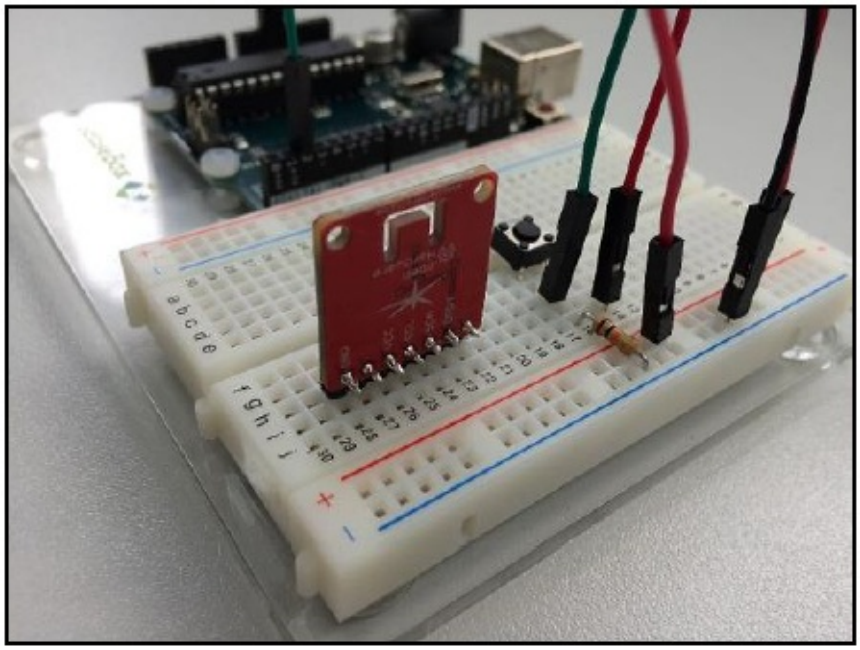


Let's build the Sensor Data logger

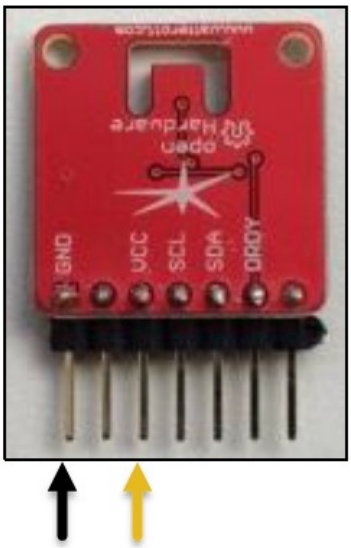
1. What do you need?
    - Your "Push-a-button" construction (A)
    - HDC1008 Temperature and Humidity Sensor (B)
    - Extra wires (C)

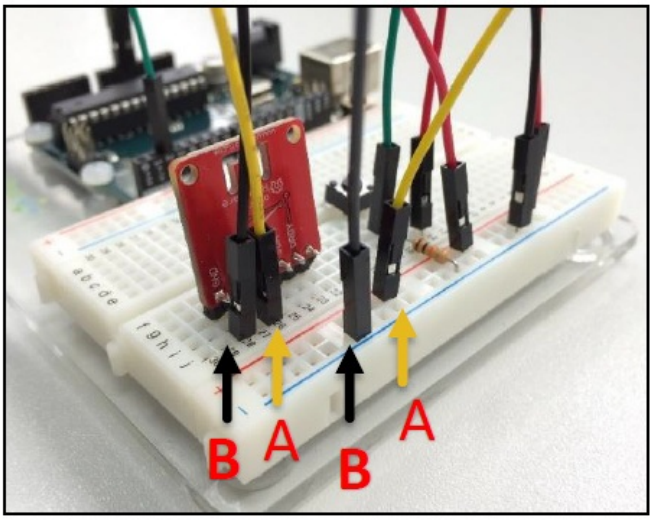


2. Connect the sensor and provide it with power and grounding
    - Connect the sensor to the breadboard.

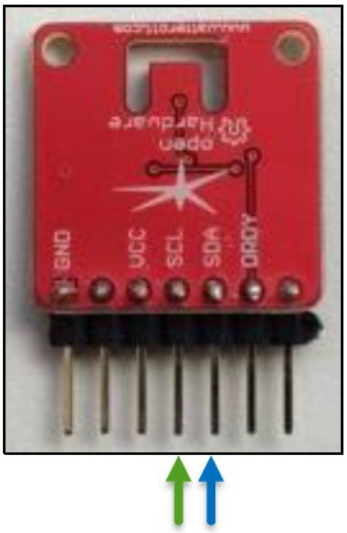


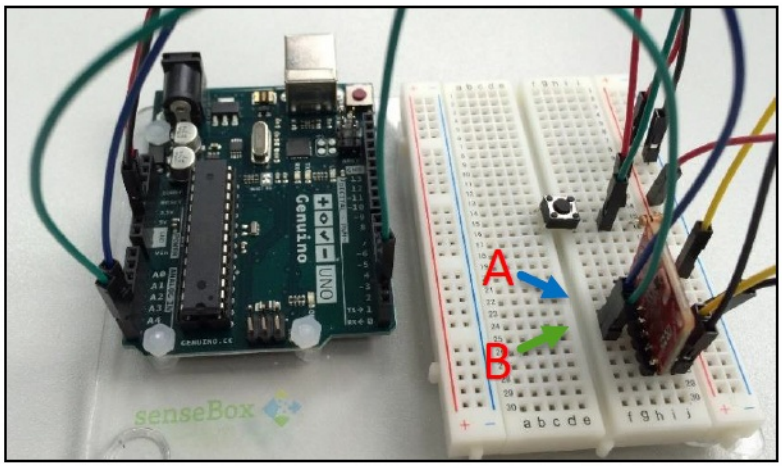
    - The labels on the sensor mark tell which pins need to be connected to the power and ground. If you are unsure, please consult the data sheet for your sensor.
    - Connect a yellow wire to the positive vertical row and to the VCC pin of the sensor (A).
    - Connect a black wire to the negative vertical row and to the GND pin of the sensor (B).

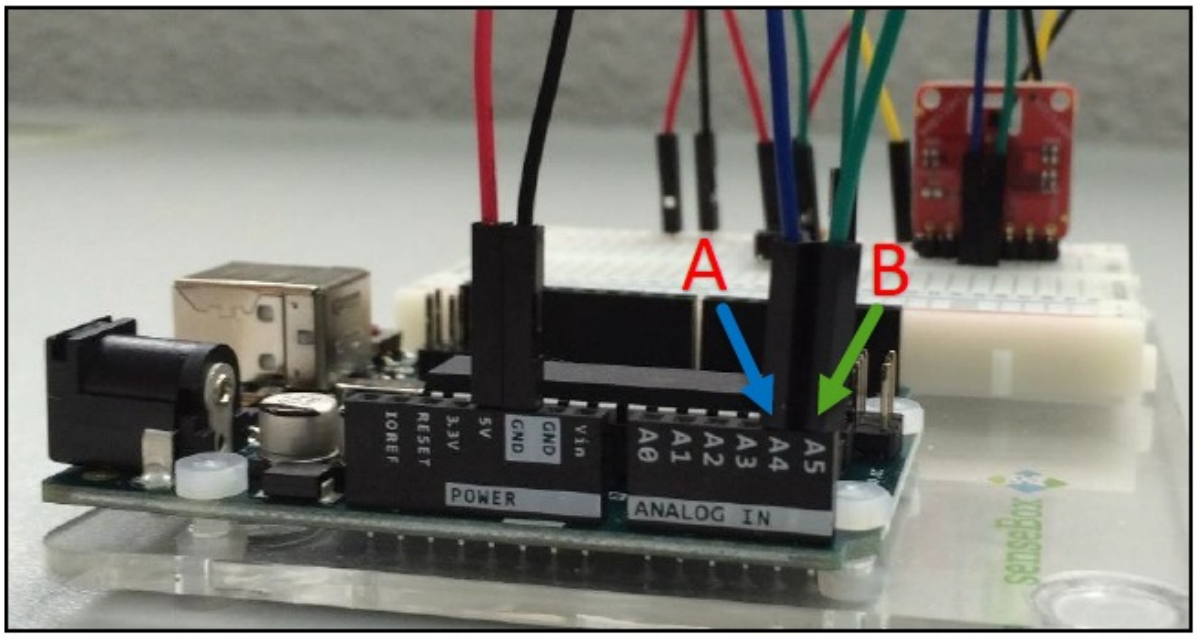




3. Create a connection between the sensor and the microcontroller to transmit measurements
    - Provide with a blue wire a connection between the analog pin AV4 (you may also connect to another free analog pin; then check if you address the sensor in your code correctly) on the Genuino Board and the SDA pin of the sensor.
    - Provide with a green wire a connection between the analog pin AV5 (you may also connect to another free analog pin; then check if you address the sensor in your code correctly) on the Genuino Board and the SCL pin of the sensor.







Write the program for the Weather Station

1. Create a new sketch
    - Start the Arduino IDE.
    - Go to File New and name the file PushbuttonTemperature.ino to create your own sketch.
2. Include the required libraries
    - Add the following code to your program to add two libraries:
        - Wire.h provides functions to communicate with the data line (SDA) pin and clock line (SCL) pin.
        - HDC100X.h makes communication and actions with the SD card possible.

        ```
        #include <Wire.h>;
        ```

        ```
        #include <HDC100X.h>;
        ```
3. Define global variables
    - Create a connection with the sensor and pushbutton.
    - Define variables for the button state and id.

```
HDC100X HDC1(0x43); // create a connection to the sensor on address 0x43

int buttonPin = 2; // the number of the pin to which the pushbutton
// is connected, change this if you
// connected the button to a different port

int buttonState = 0; // variable for reading the status of the pushbutton

int lastButtonState = 0; // previous state of the pushbutton

int id = 0; // count for how many times the button has been pressed
// and set this as ID
```

1. Write the setup() function
   - Start the sensor.
   - Initialize the pushbutton as an input.

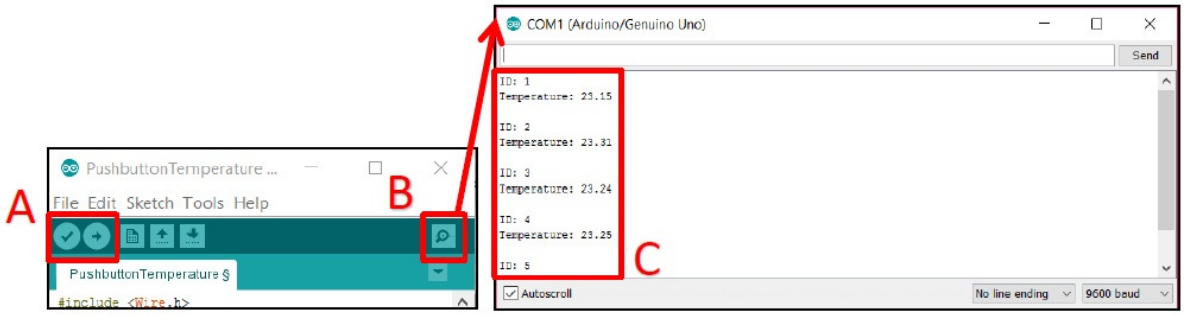
```
void setup() {

Serial.begin(9600); // provide communication with the
// computer with data rate 9600 bits per second

HDC1.begin(HDC100X\_TEMP\_HUMI, HDC100X\_14BIT, HDC100X\_14BIT,
DISABLE); // start the sensor

pinMode(buttonPin,INPUT); // initialize the pushbutton pin
// as an input

}
```

2. Write the loop() function
   - Read the value from the push button.
   - If the button is pressed, measure the temperature and print it to the console.

```
void loop() {

  buttonState = digitalRead(buttonPin); // read value from push button

  // if button is pressed in

  if (buttonState != lastButtonState && buttonState == HIGH) {

  id++; // update id

  Serial.print("ID: "); // print the id to the console

  Serial.println(id);

  Serial.print("Temperature: "); // get the temperature and
  // print it to the console

  Serial.println(HDC1.getTemp());

  Serial.print("\\n");

  }

  lastButtonState = buttonState; // set current state as last button state

}
```

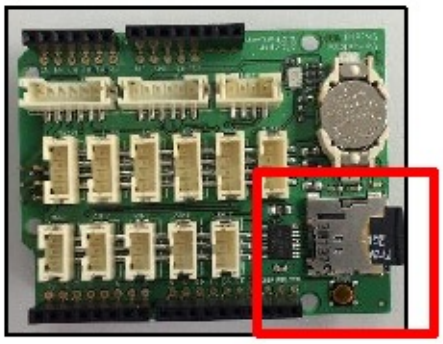
3. Run your program

- Compile your program and upload it to the board (A).
- Click on the Serial Monitor button. The COM1 dialog should pop up (B).
- Push the button on your board. A new result should appear on your screen (C).

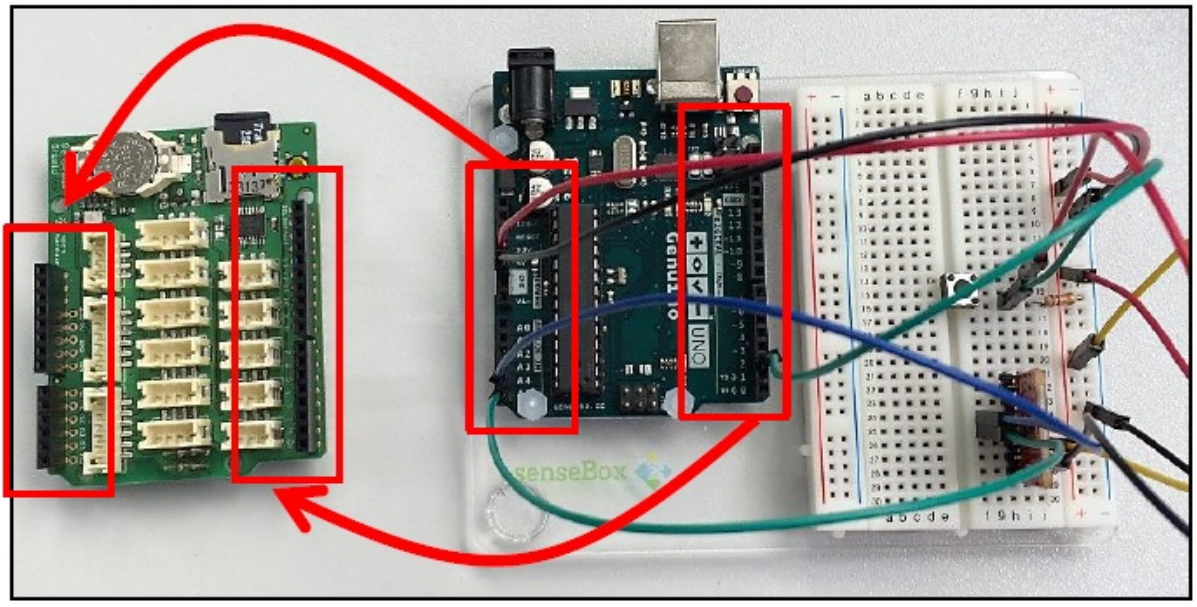


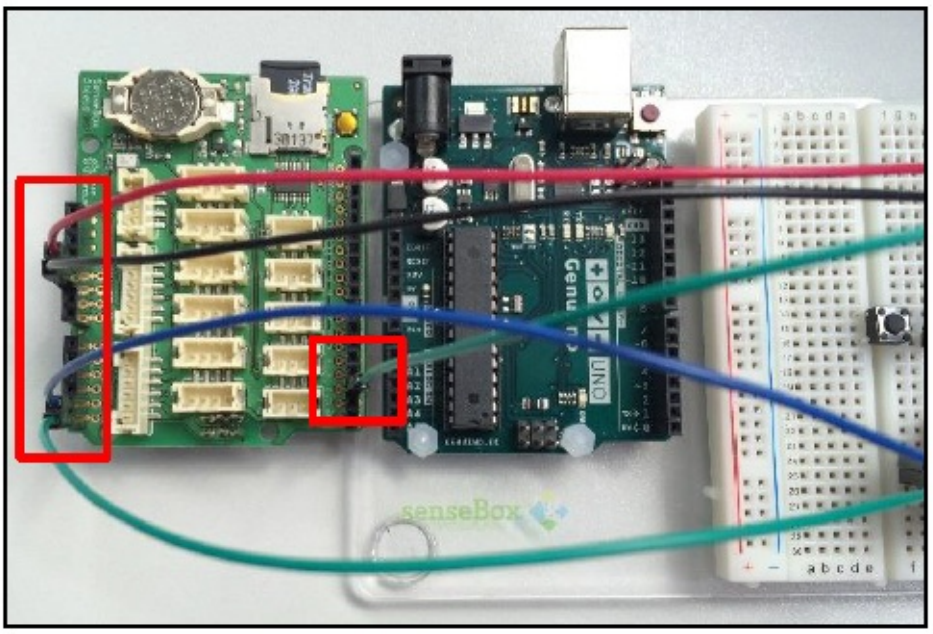
Write the results to a file on a SD card

1. What do you need?
   - Your Arduino construction from the previous steps.
   - A senseBox Shield with SD card slot.

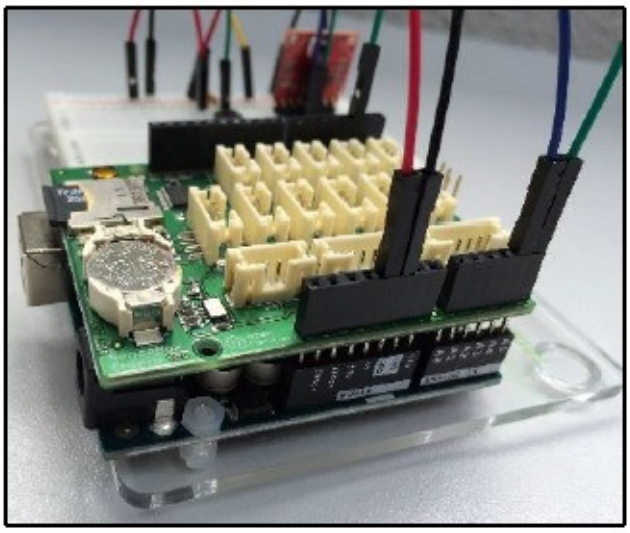


1. Connect the senseBox Shield to the Genuino Board
   - Since the senseBox Shield will be placed on top of the Genuino Board, the wires need to be connected to the senseBox Shield.





2. Place the senseBox Shield on the Genuino Board.



3. Adjust your program
    - Open the file PushbuttonTemperature.ino in the Arduino IDE.
    - Add a library for SD card functionalities:

```
#include <SD.h>
```

- Define a global variable which contains the identifier of the pin connected to the SD card.

```
int chipPin = 4; // pin connected to the SD card
```

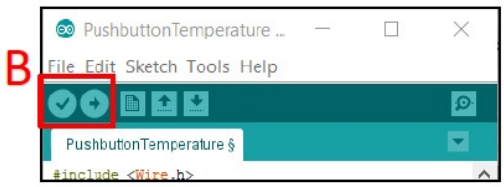
    - Initialize the SD card in the setup() function.

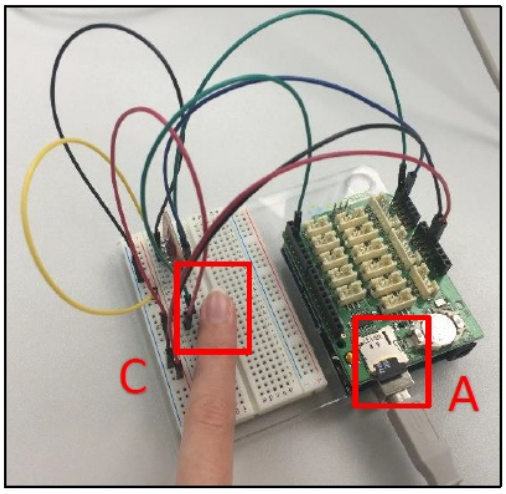
```
SD.begin(chipPin); // initialize the SD card.
```

    - Adjust the loop() function.

```
void loop() {

  buttonState = digitalRead(buttonPin); // read value from push button

  // if button is pressed in

  if (buttonState != lastButtonState && buttonState == HIGH) {

    id++; // update id

    File file = SD.open("temp.csv", FILE\_WRITE); // open a file named temp.csv

    file.print(id); // print the id to the file

    file.print(";");

    file.print(HDC1.getTemp()); // get the temperature and print it to the file

    file.print("\\n");

    file.close(); // close the file

  }

  lastButtonState = buttonState; // set current state as last button state

}
```

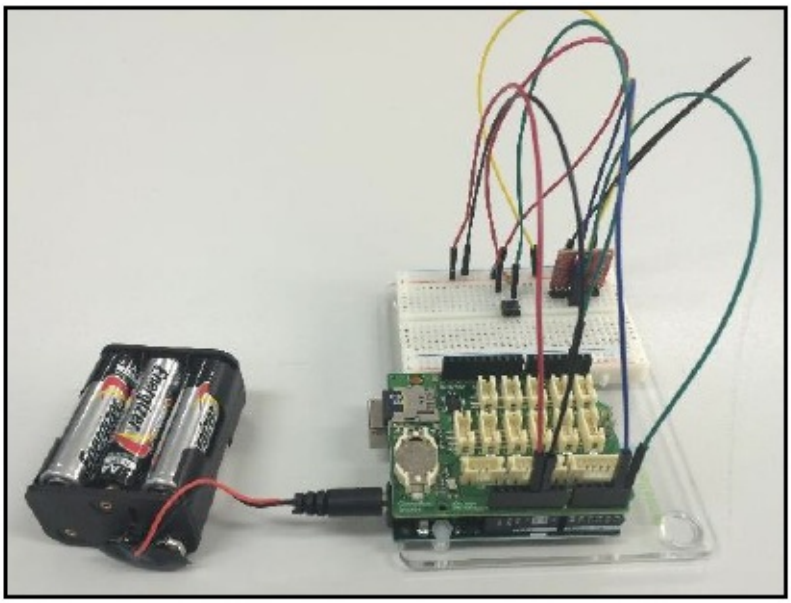
- Run your program
  - Place the SD card in the slot on the senseBox Board (A).
  - Compile your program and upload it to the board (B).
  - Push the button on the breadboard multiple times (C).





- View the results
  - Place the SD card in the card reader slot of your computer.
  - Open the file TEMP.CSV.
  - You can now see the resulting temperature measurements.

Make it mobile



Time to go on a field study and measure the temperatures throughout the building and surroundings. You can make your Arduino weather station more mobile by attaching a battery to the Genuino Board. Good luck during your expedition!

DIGITAL CITIZENS, CONNECTED COMMUNITIES:

THE ROLE OF SPATIAL COMPUTING AND VOLUNTEERED GEOGRAPHIC INFORMATION

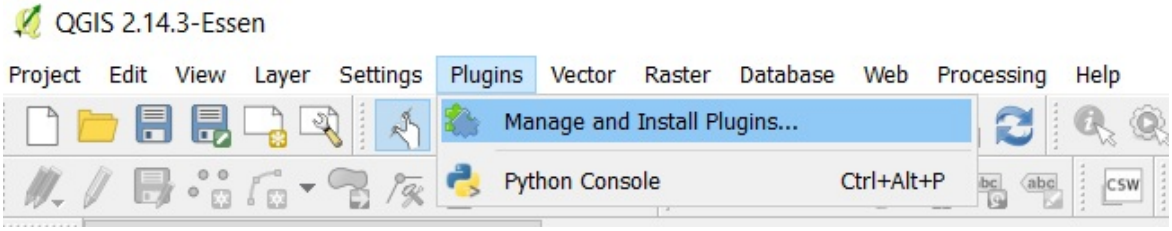
Tutorial: Create a heat map in QGIS

Edited by Felix Erdmann using pandoc

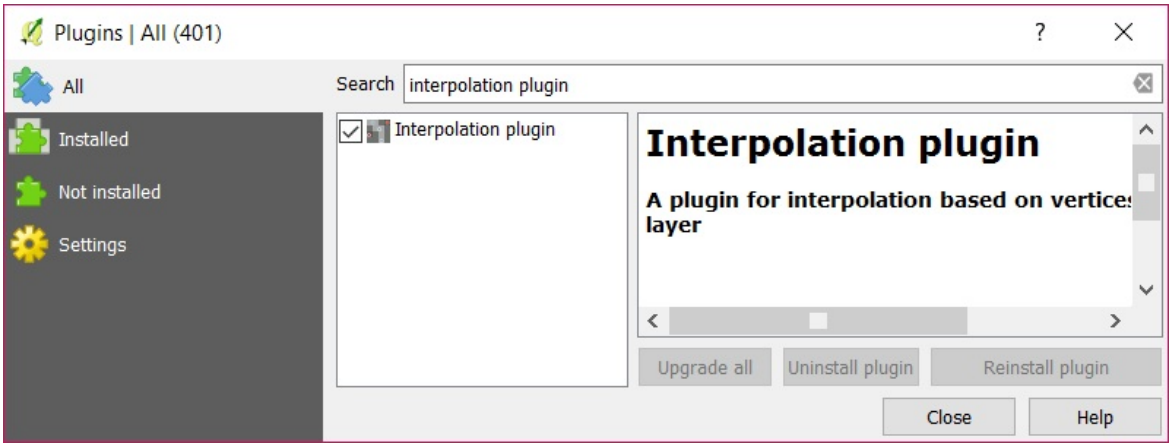
The field measurements you have acquired with your DIY weather station can be used to generate a thematic map. Since the measurements consist of temperature data a heat map is the most visualization type for that. In this tutorial you will learn how to visualize your data as a heat map in QGIS.

Preparation

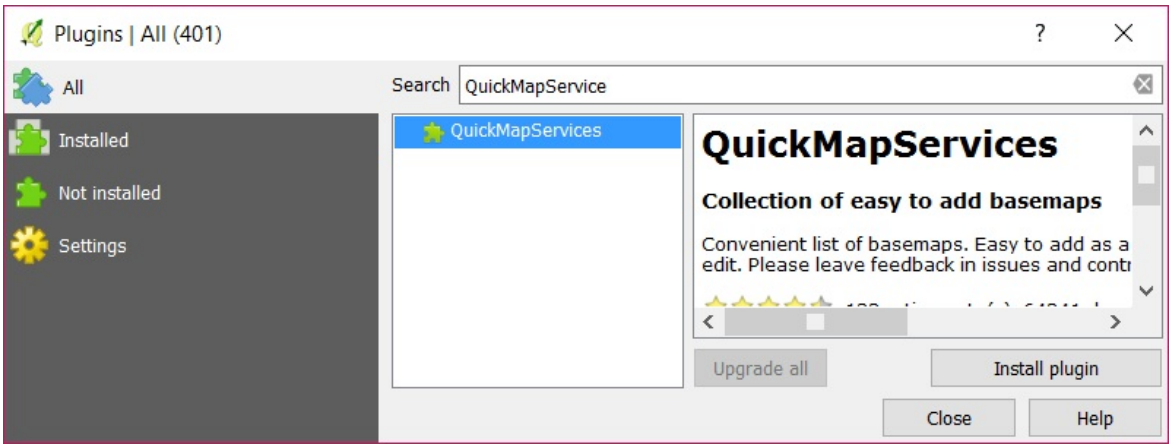
1. Install QGIS
    - http://www.qgis.org/nl/site/index.html
    - (Extra:) Documentation and a QGIS Training Manual can be found here:\ http://www.qgis.org/en/docs/index.html
2. Install Plugins
    - Open QGIS.
    - Go to Plugins Manage and Install Plugins...



    - Search for Interpolation Plugin (A). Select Install Plugin (B) if this option is still available.

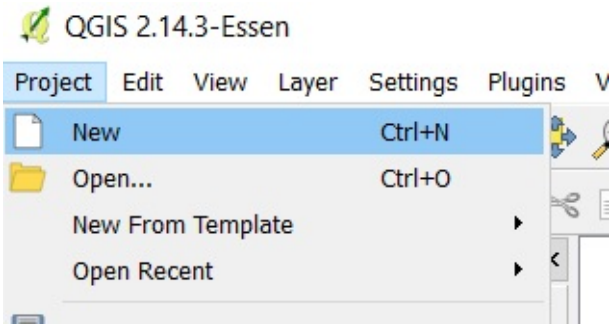


- Search for QuickMapService (A) and select Install Plugin (B).

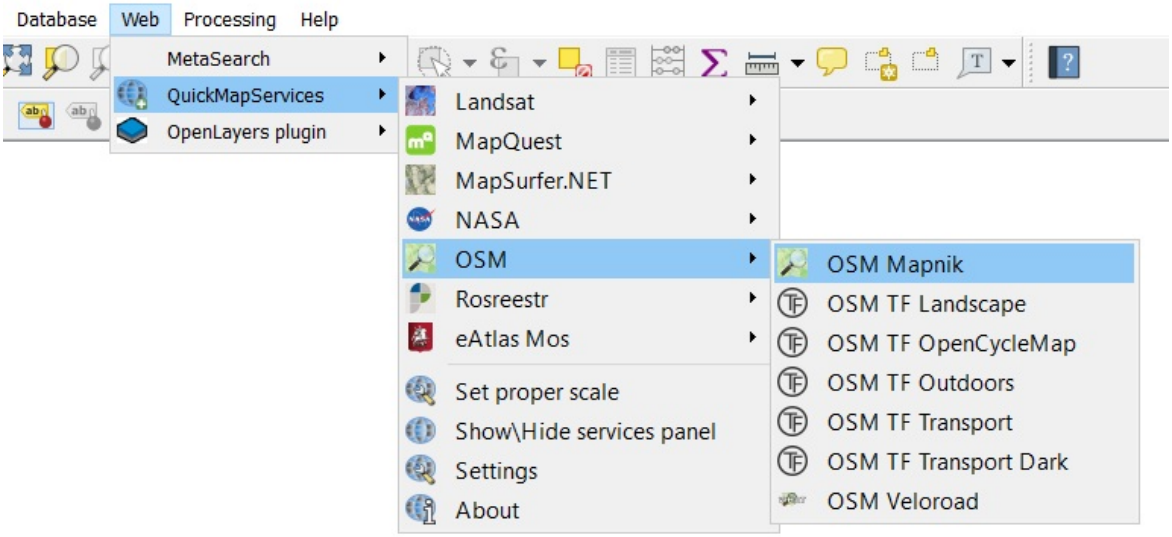


Import the data

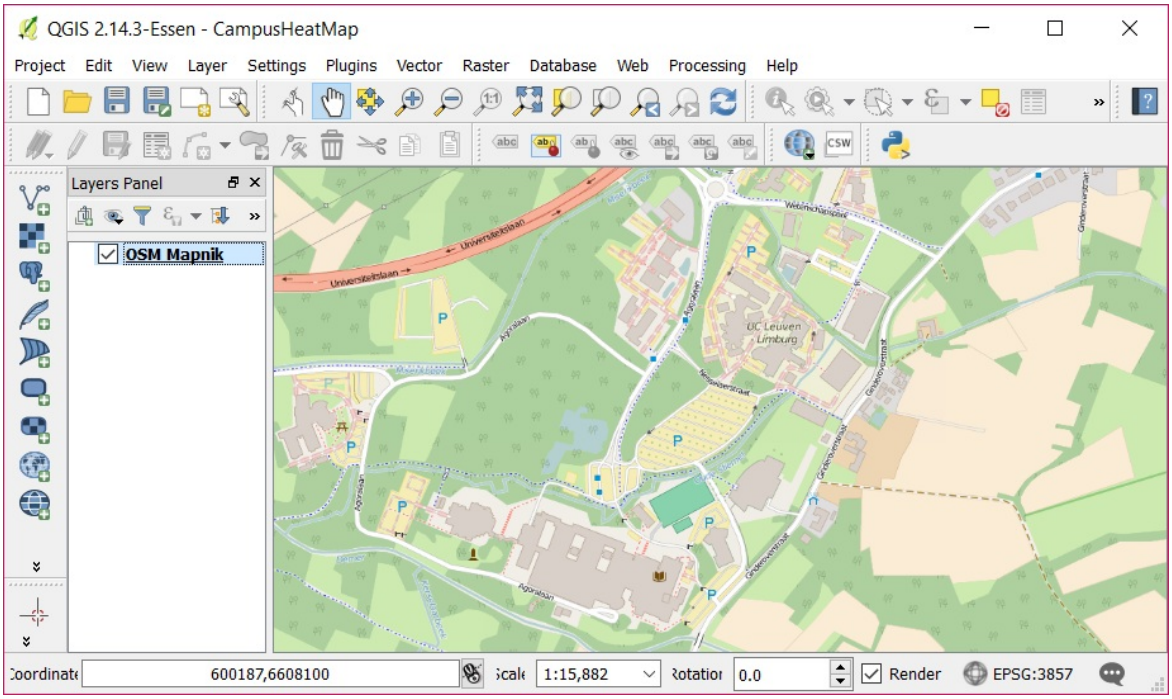
1. Create a new project
   - Go to Project New.



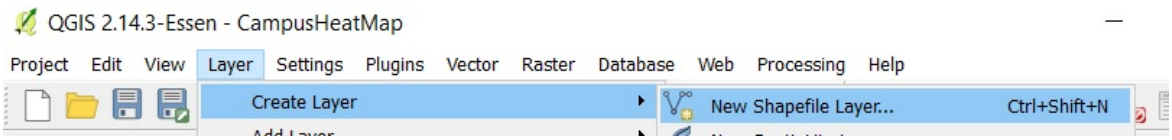
   - Save your project as CampusHeatMap.qgs.
2. Get OpenStreetMap data
   - Go to Web QuickMapServices OSM OSM Mapnik.

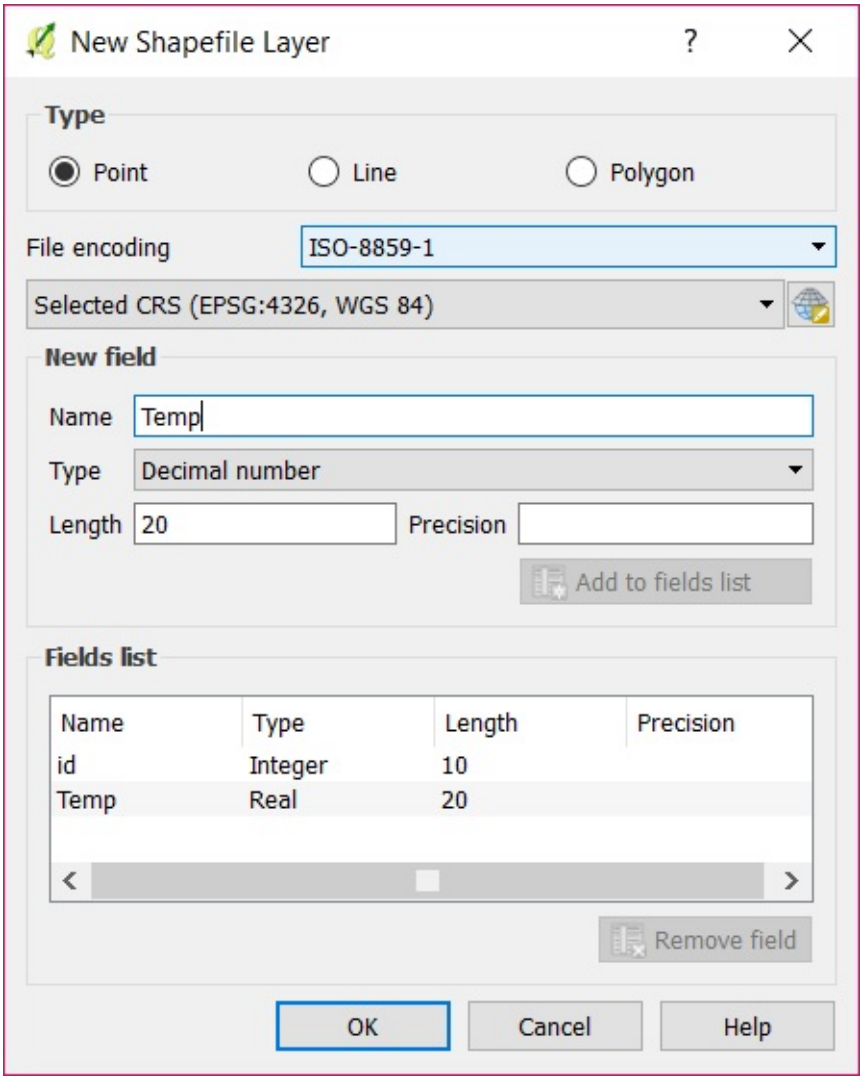


   - Open the layer OSM Mapnik and navigate to Campus Diepenbeek by using the Pan tool.



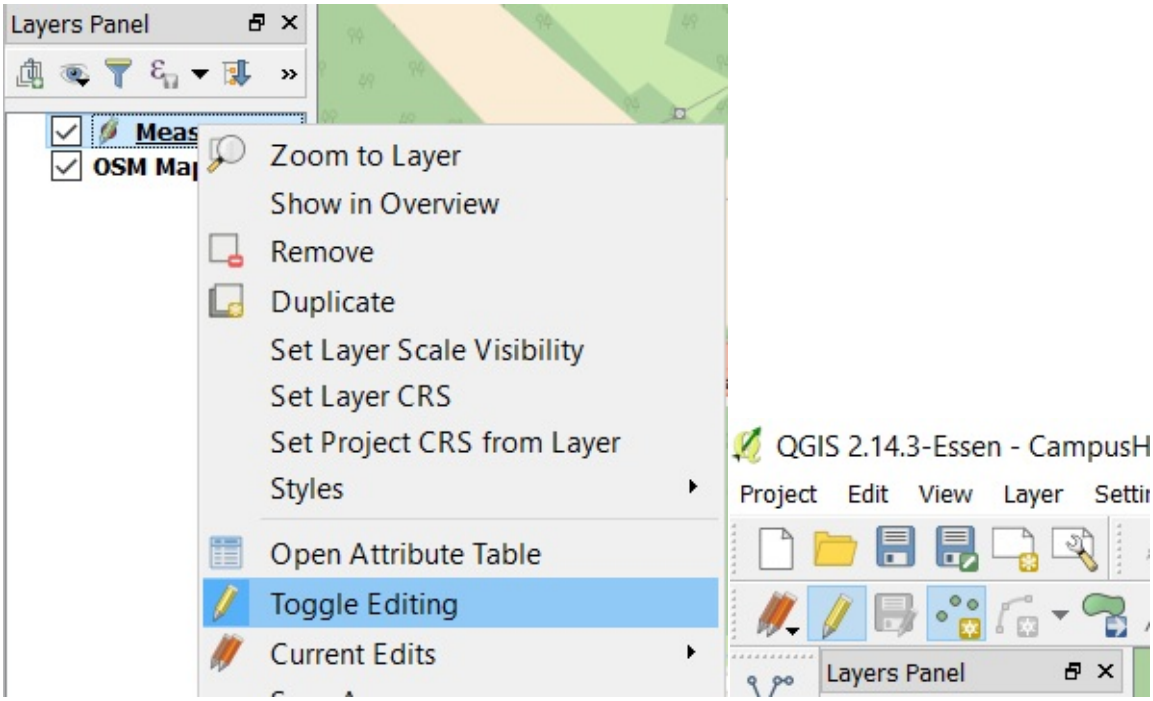
3. Add a layer for the temperature measurements
    - Go to Layer Create Layer New Shapefile Layer...



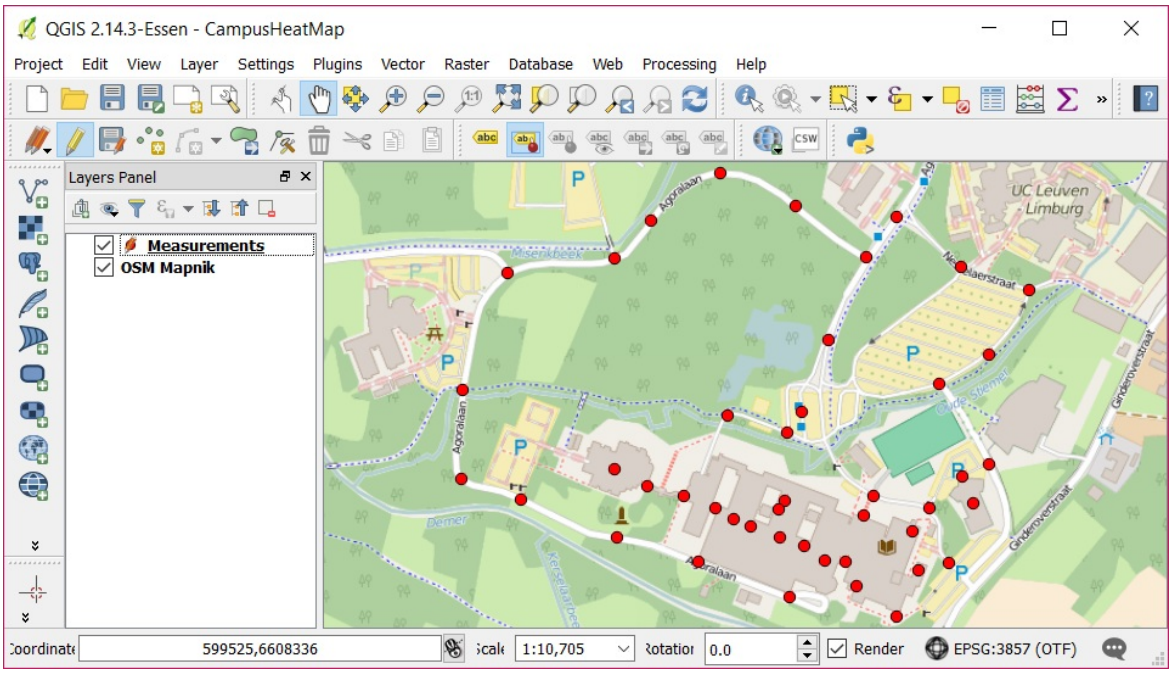


- Set the type of the new shapefile layer to Point (A).
- Add a new field Temp which type is Decimal number. Don't forget to click on the button Add to fields list (B).
- Click on OK (C).
- Name the layer Measurements.

4. Add points to the Measurements layer
   - Right click on Measurements and select Toggle Editing.
   - Click on the Add Feature button. Points will now be added when you click on the map.
   - Place points to mark all locations where you have measured the temperature. You can add the temperature values directly, but other methods will be explained later.



5. Changing the properties of the layers within the project
    - The checkboxes in front of the layers indicate whether a layer is visible or not.
    - By changing the order of the layers you decide which layers are in the front and which will function as backgrounds.
    - Make sure both layers are visible and Measurements is the top layer.

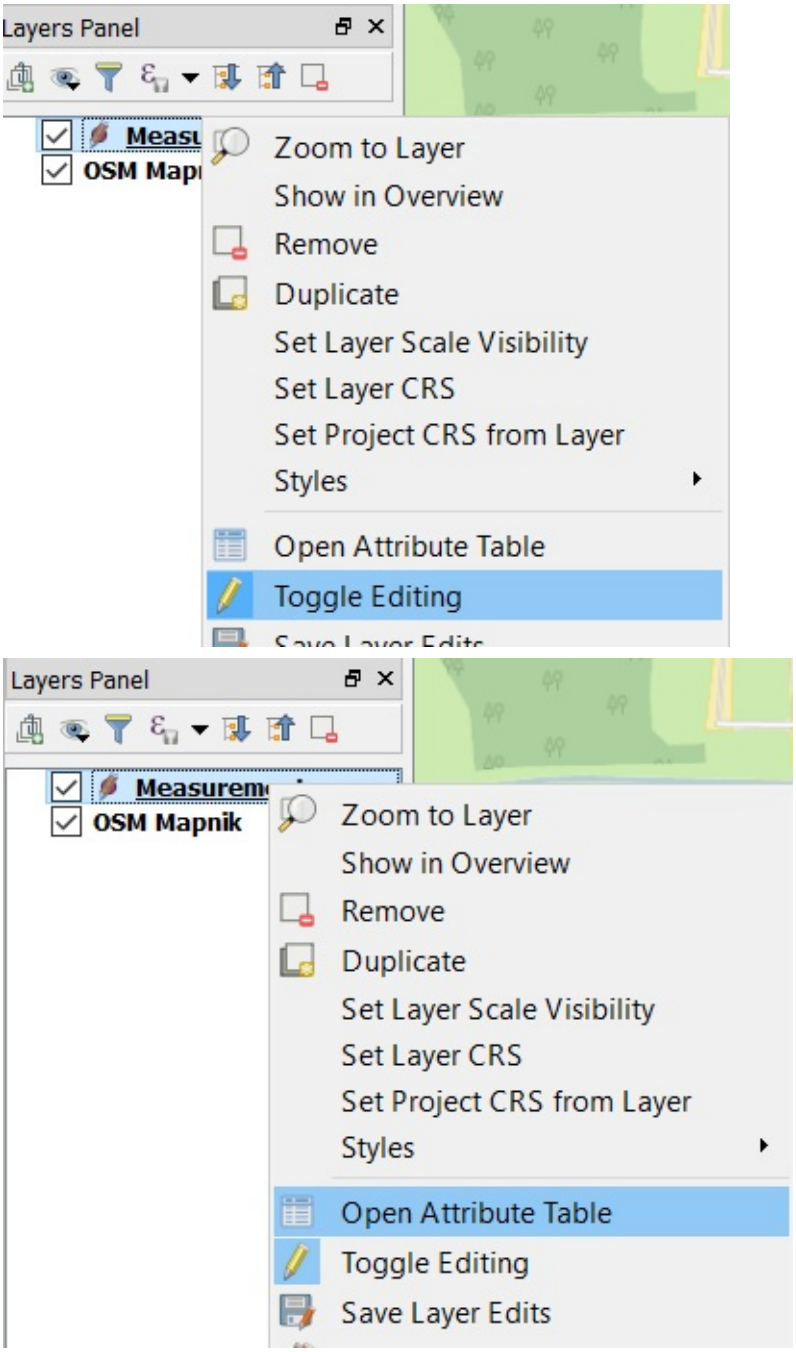


6. Update Measurements with your own data

    This can be done in various ways:

    Option 1: Insert the data manually

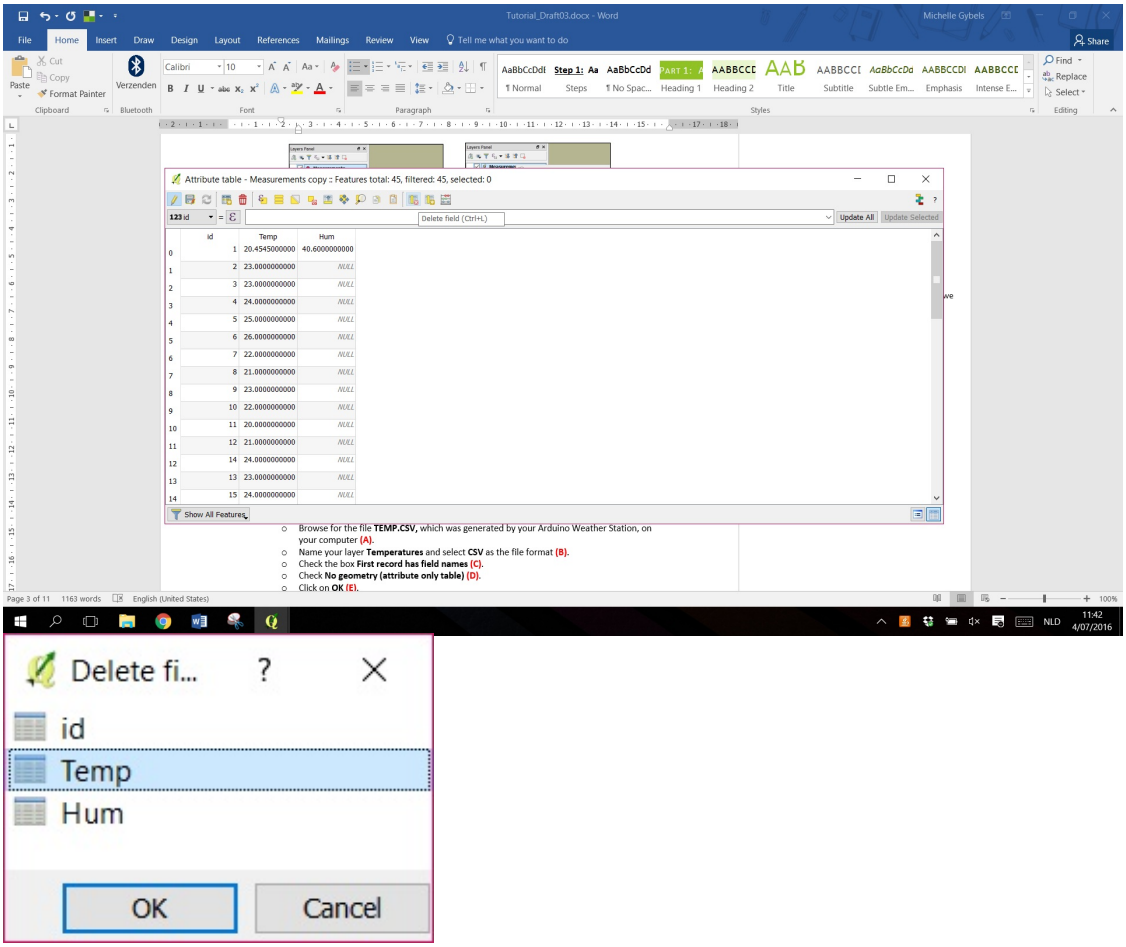
    - Right click on Measurements and select Toggle Editing (A).
    - Right click on Measurements again and select Open Attribute Table (B).



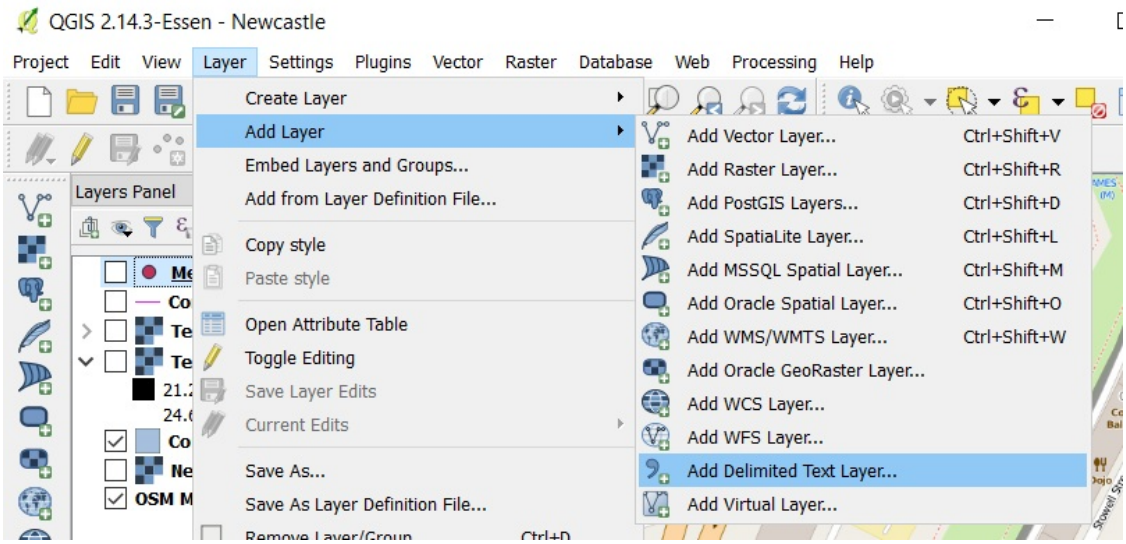
- Select each cell of the Temp field in the attribute table and fill in your obtained values.
- When you are done editing, select Toggle Editing again.

Option 2: Import the data from a CSV file

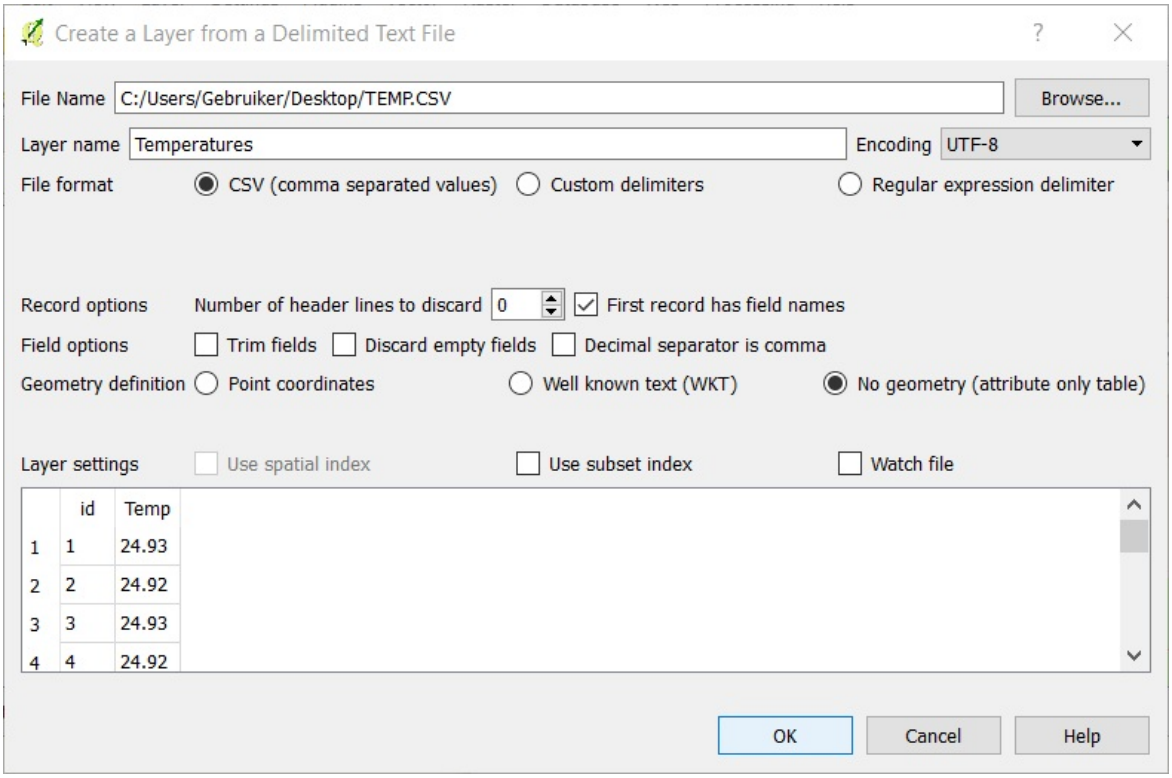
- Delete the Temp field from the attribute table.
  - Right click on Measurements and click on Toggle Editing.
  - Right click on Measurements and open the attribute table.
  - Click on the Delete field button (A).
  - Select Temp and click on OK (B).



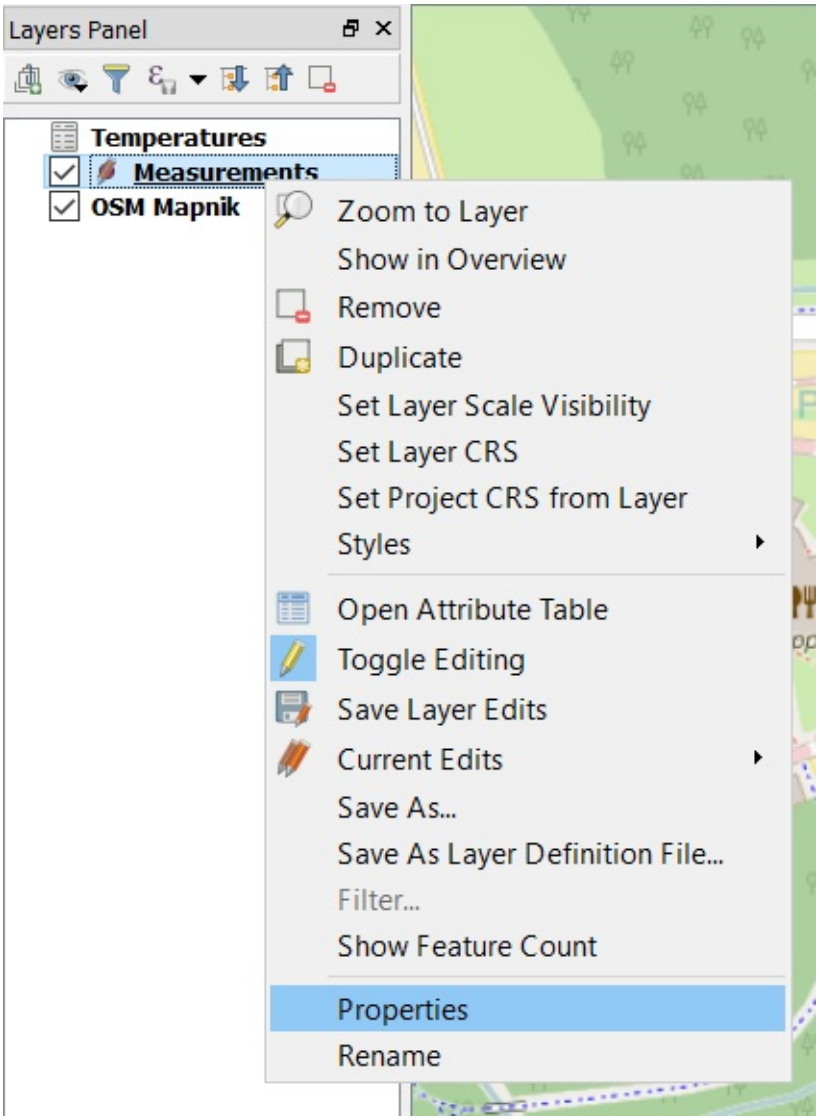
- Import your CSV file as a layer.
  - Go to Layer Add Layer Add Delimeted Text Layer...

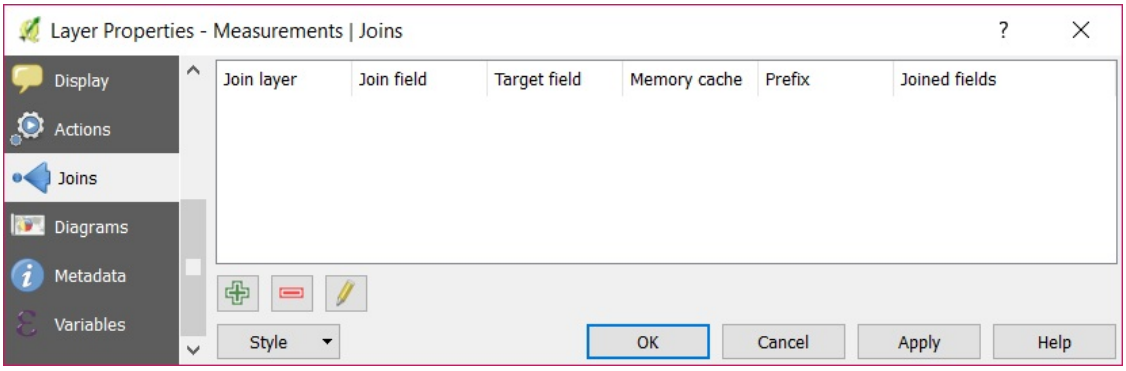


- Browse for the file TEMP.CSV, which was generated by your Arduino Weather Station, on your computer (A).
- Name your layer Temperatures and select CSV as the file format (B).
- Check the box First record has field names (C).
- Check No geometry (attribute only table) (D).
- Click on OK (E).

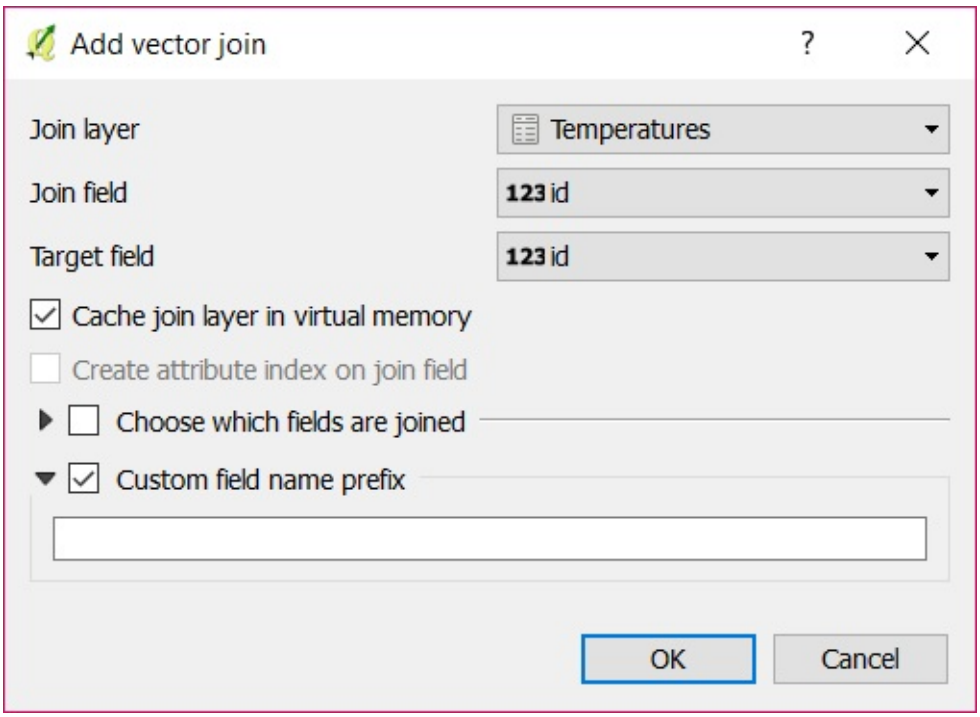


- Join the Temperatures layer with the Measurements layer.
  - Right click on Measurements and select Properties (A).
  - Open the tab Joins and click on Add (B).



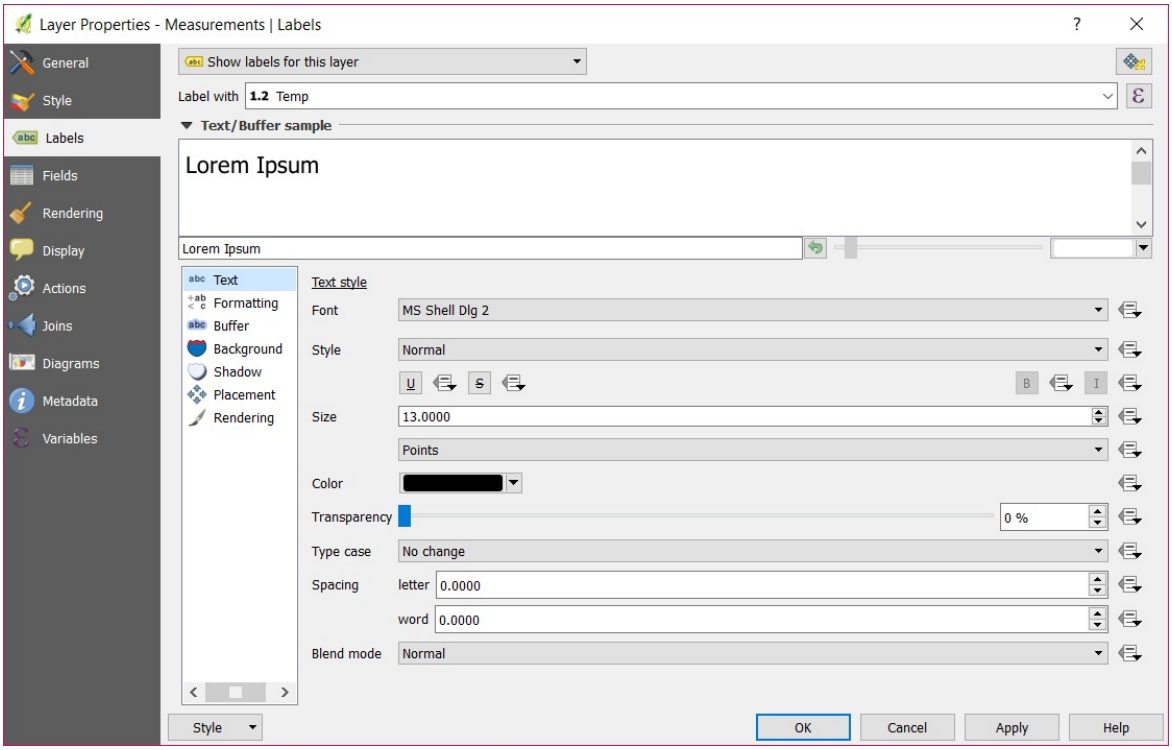


- Set Temperatures as the Join layer. And id as the Join field and Target field (A).
- Make sure the Custom field name prefix field is empty (B).
- Click on OK (C).



- Click on OK in the Layer Properties window.

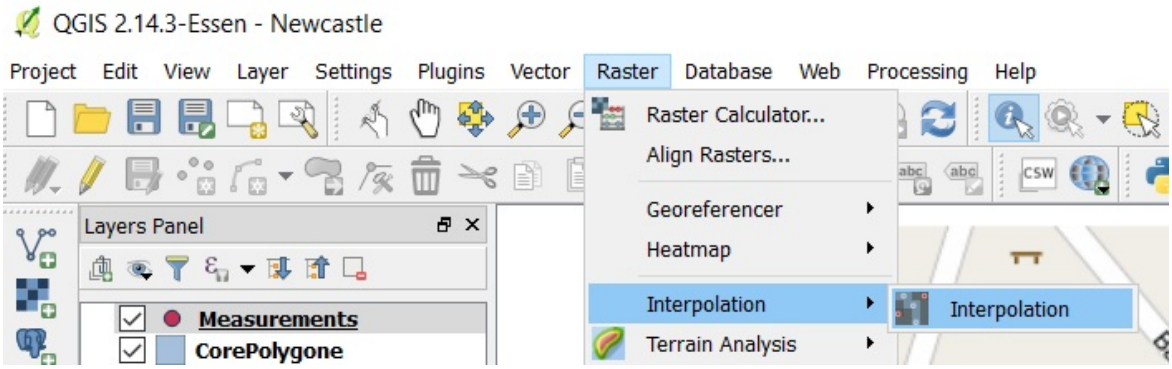
7. Change the visualization of the layer Measurements
   - Right click on Measurements and select Properties.
   - Click on Labels and select Temp as variable to label the data points with (A).
   - Optionally, you can change the marker within the Style menu (B).



Interpolating Point Data

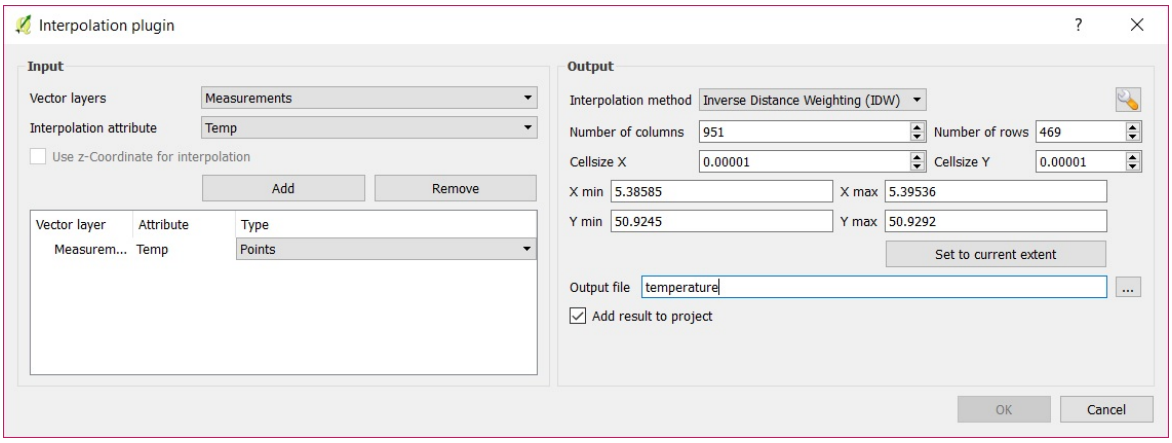
1. Open the Interpolation Plugin

- Go to Raster Interpolation Interpolation.



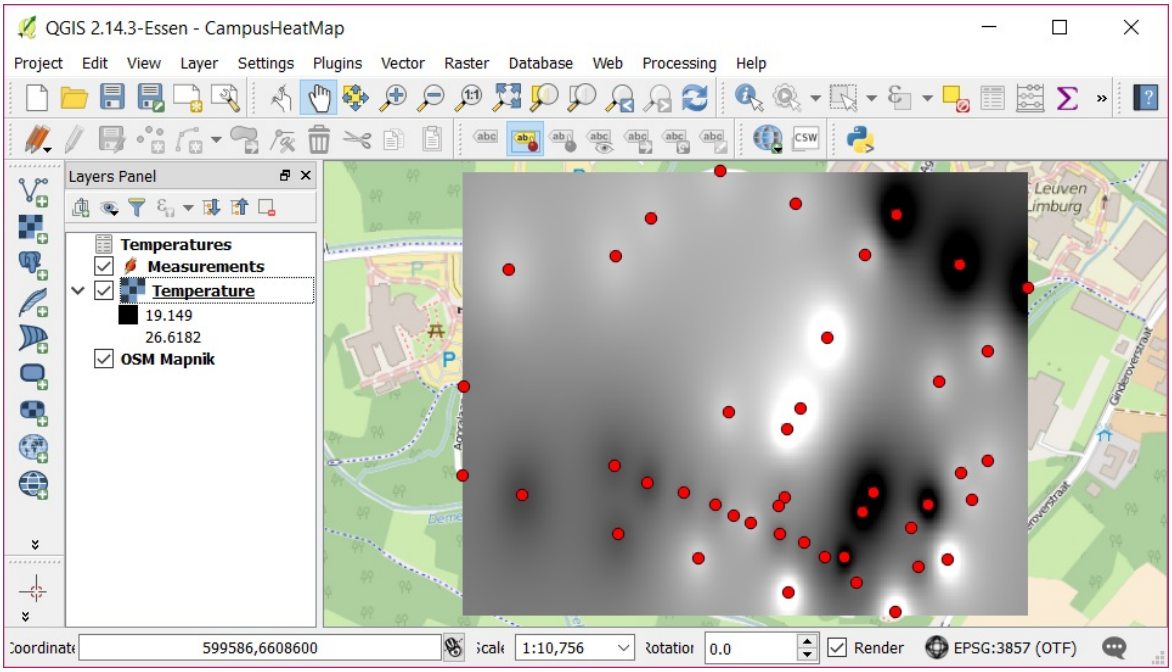
2. Fill in the parameters within the Interpolation Plugin dialog

- Set Measurements as the input vector layer and Temp as the interpolation attribute (A).
- Click on Add (B).
- Within the output section, select Inverse Distance Weighting (IDW) as interpolation method (C).
- Fill in the other values as presented in the image below (D).
- Navigate to a folder to save the generated output file and name it temperature (E).
- Click on Ok (F).



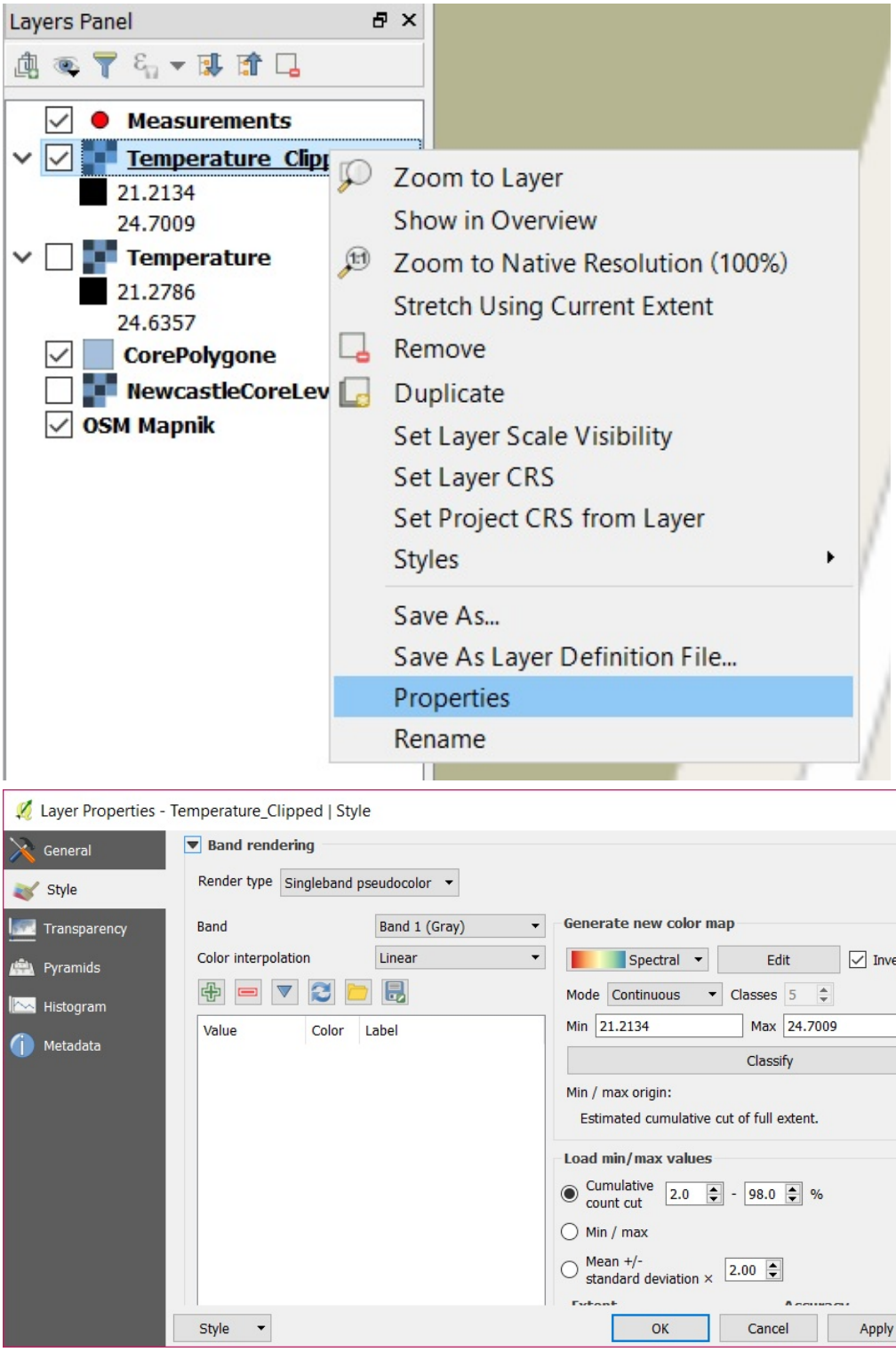
3. Verify your result

- The generated output file is added as a layer within the project.
- To zoom in on the layer, right click on Temperature and select Zoom to Layer.
- Move the Measurements layer on top of Temperature.

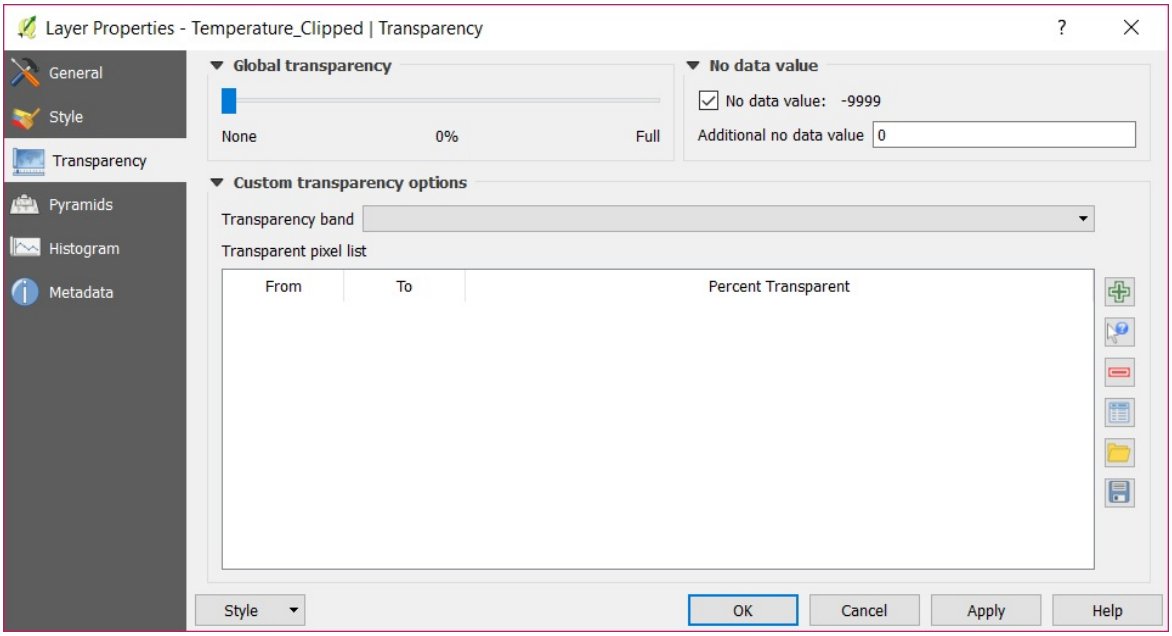


Change the style of the (clipped) layer

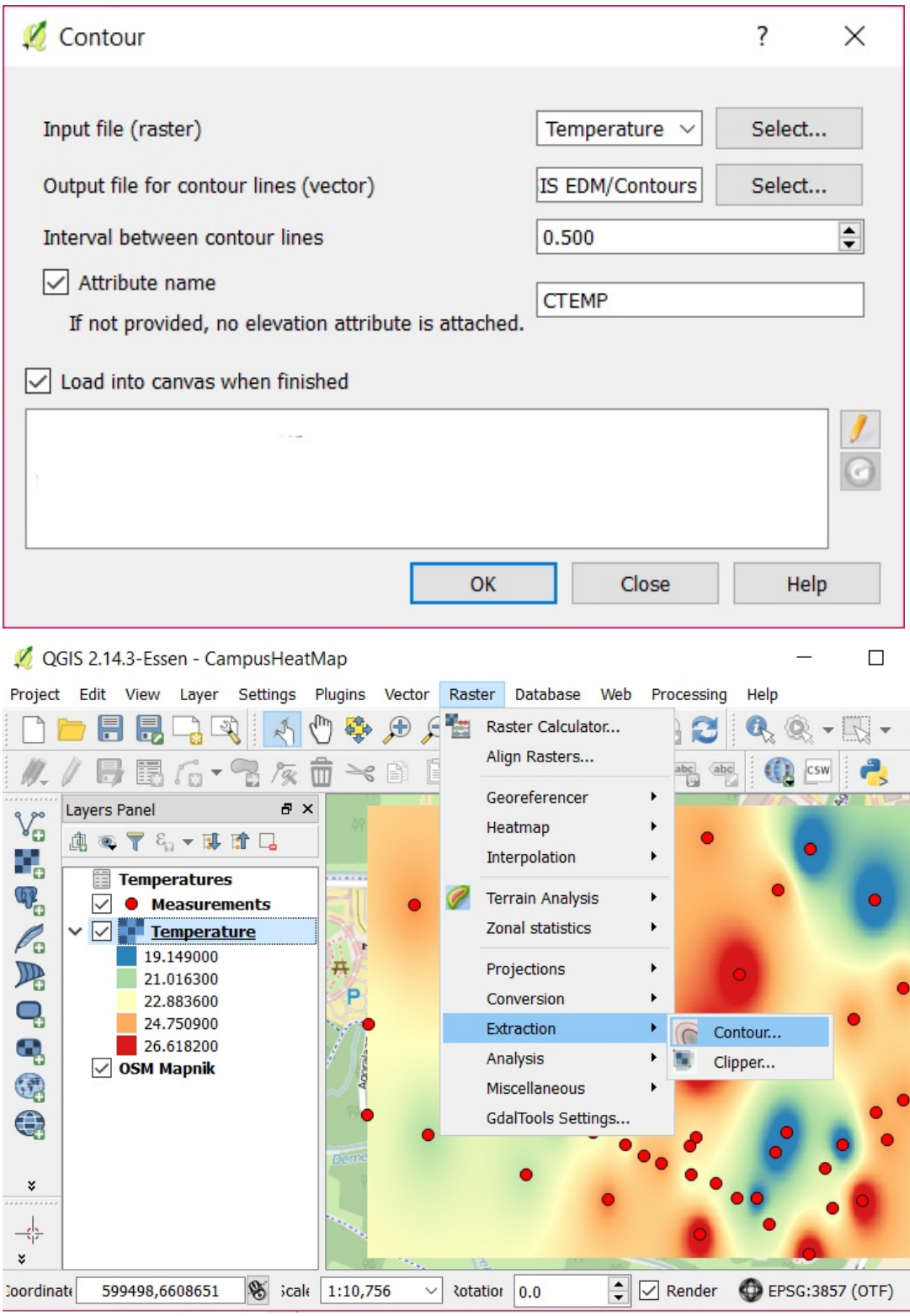
1. Adjust the color map
    - Right click the Temperature layer and select Properties (A).
    - Open the Style tab and set the render type to Singleband pseudocolor (B).
    - Select Spectral color map (C).
    - Check the Invert box, so that blue will be assigned to low temperatures and red to high temperatures (C).
    - Click on Classify (D).



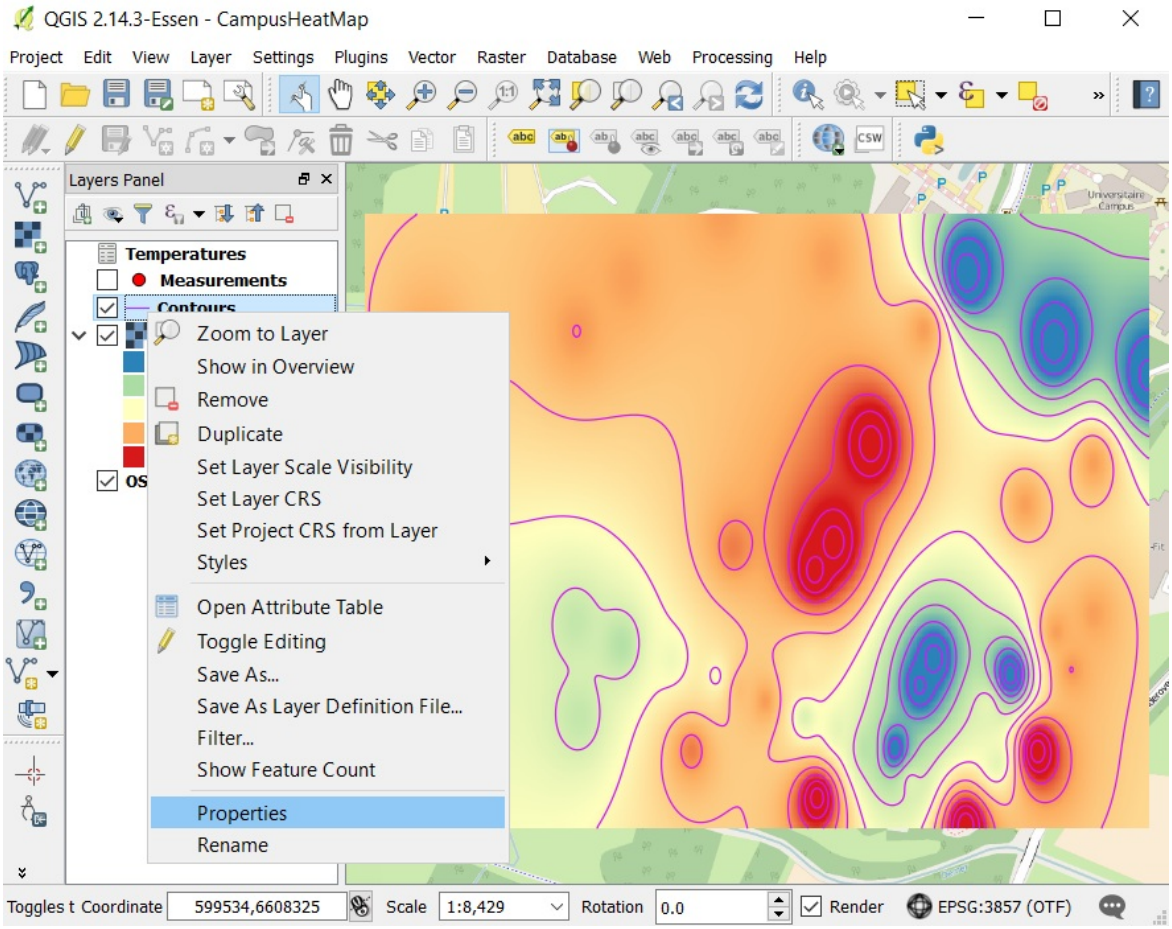
2. Remove the black pixels from the output
    - Open the Transparency tab.
    - Set 0 as the additional no data value.
    - Click on Ok.
    - This results in a temperature relief map for the campus.



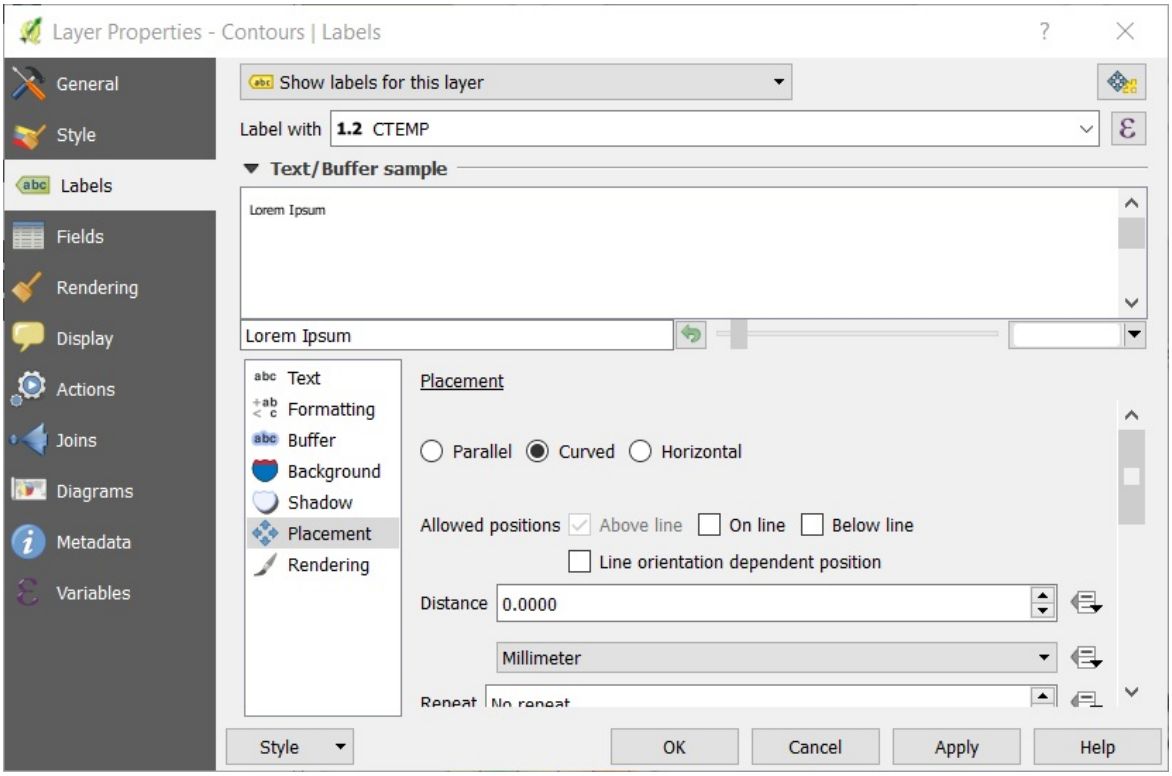
3. [Optional] Generate contours lines
    - Go to Raster Extraction Contour (A).
    - Set the Temperature_Clipped layer as the input file and name the output file Contours (B).
    - Set the interval between contour lines as 0.5 or 1(C).
    - Check the Attribute name box and set this to CTEMP (D).
    - Click on OK (E).



4. Add labels to the contour lines
    - Right click on the layer Contours and select Properties.

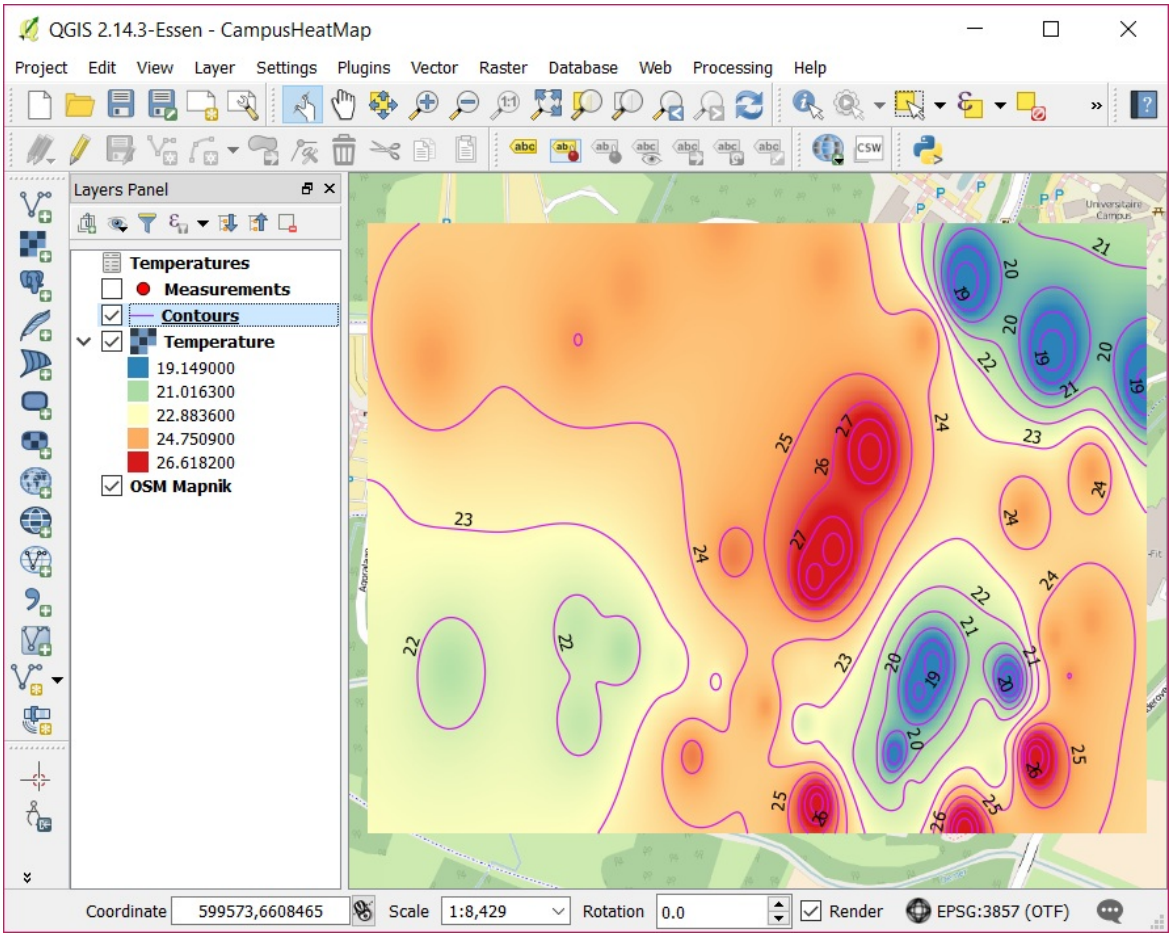


- Open the Labels tab (A).
- Select Show labels for this layer in the dropdown box (B).
- Set CTEMP as the value for Label with (C).
- Select Curved as Placement type (D).
- Click on OK (E).



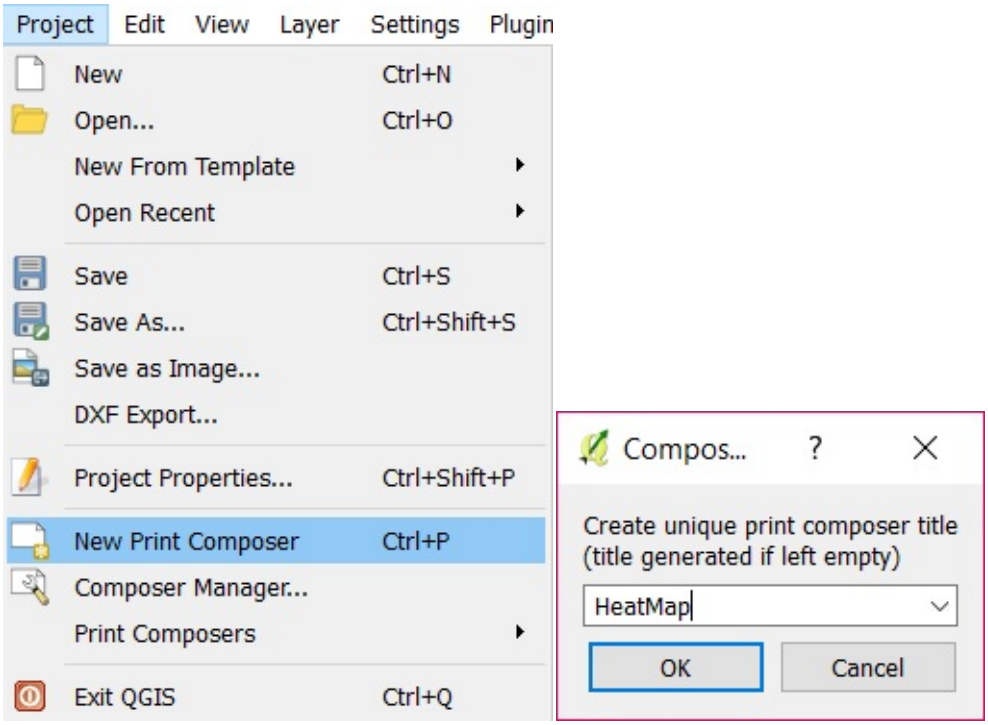
5. Verify the result

   - The result should look like this:

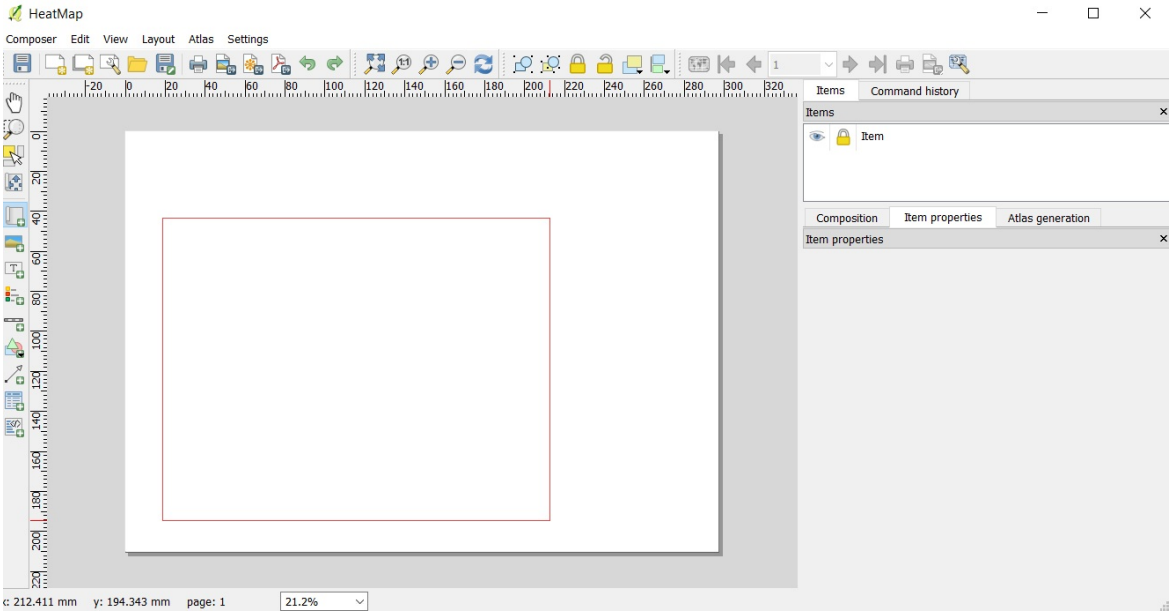


Exporting the result

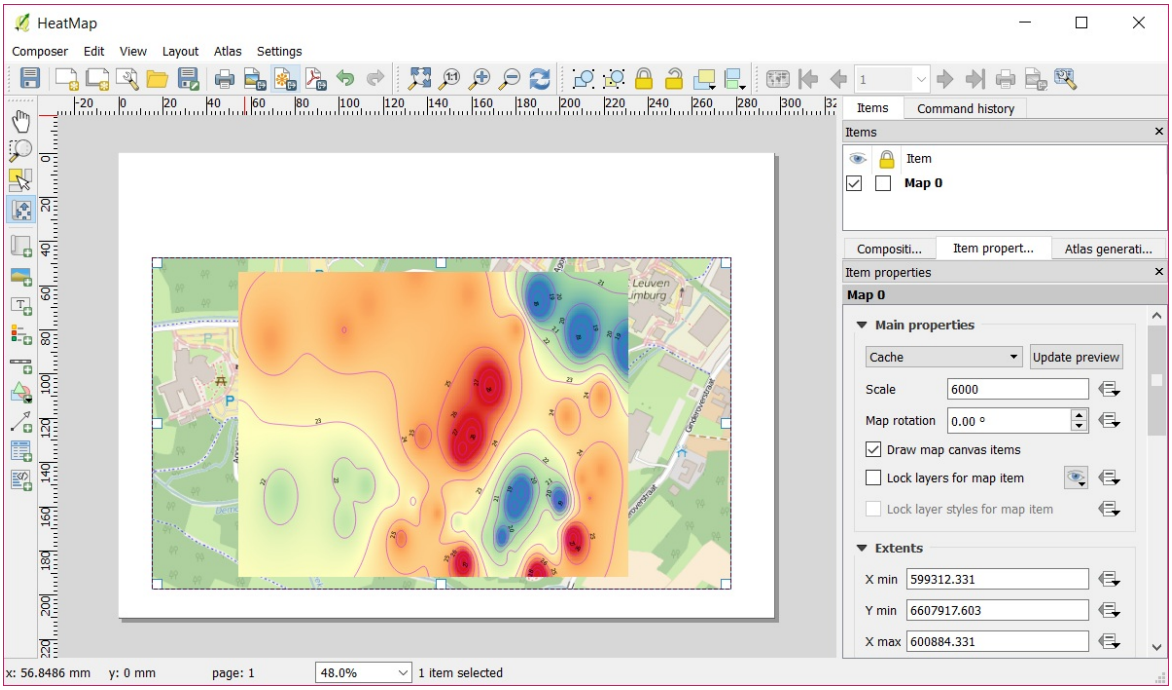
1. Create a new print composer
    - Go to Project New Print Composer (A).
    - Set HeatMap as the print composer title (B).

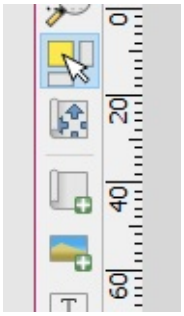


2. Add the created heat map
    - Click on the Add new map button (A).
    - Draw a rectangle on the presented page (B).

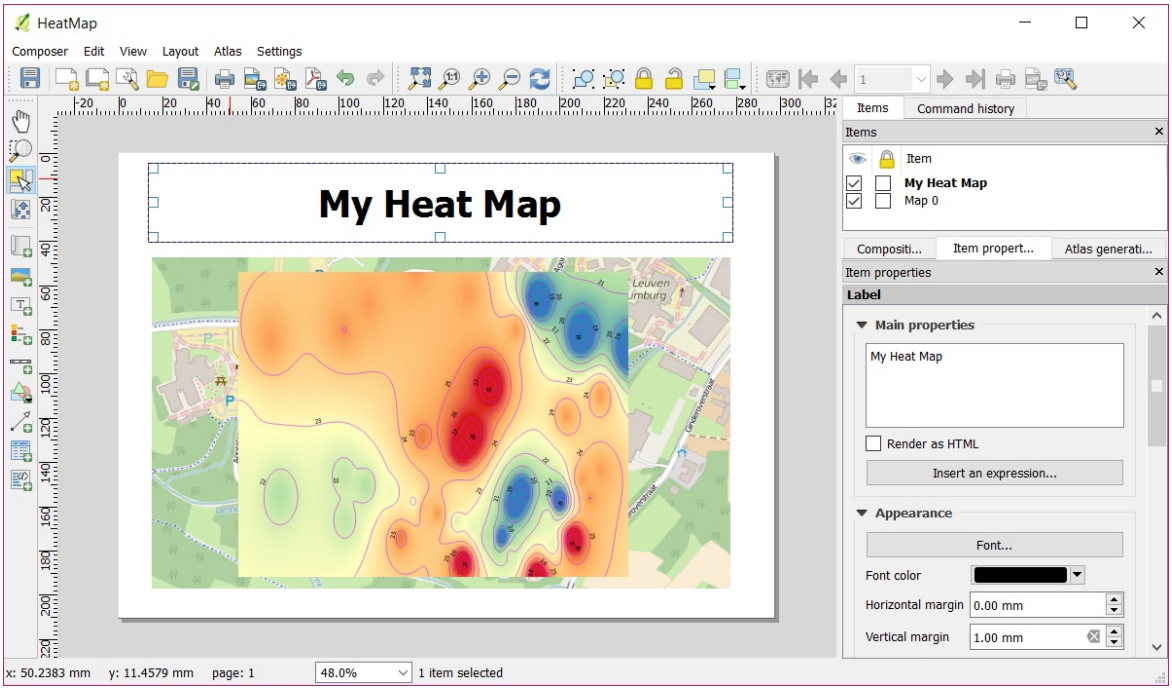


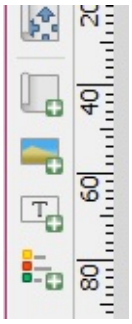
    - Click on the Move item content button (C).
    - Click on the map and drag it until the heat map is centered within the rectangle.
    - Set Scale to 6000 (D).



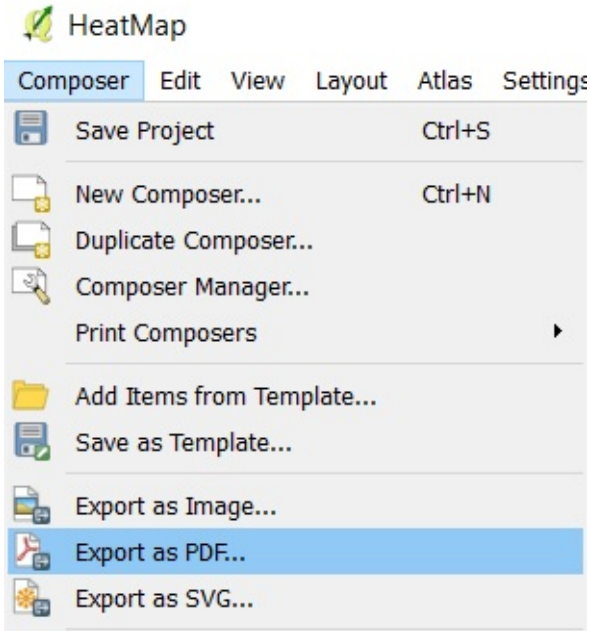


3. Add a title to the page
    - Click on the Add new label button (A).
    - Draw a rectangle on the page (B).
    - Set a title for the heat map (C).
    - Choose a font for the title (D).





4. Save your creation



- Go to Composer Export as PDF.
- Name your file MyHeatMap.pdf.

Contributing Projects

If you developed a senseBox project on your own and did document it, you may add it to this book!

Such a documentation should provide an overview over the project as well as a step-by-step tutorial on how to build / reproduce the project. Photos and code may be included, too!

For this book's management we use github.com and the tool GitBook, all content is written in markdown.

Besides new project tutorials we also appreciate any improvements or corrections to the existing content!

Writing the documentation

The documentation should be written in markdown, so the content can be directly included in the book. If you are not familiar with markdown, have a look at an explanation and syntax-explanation here.

To streamline the writing of markdown we recommend dedicated editors, such as the web-editor stackedit.

Content

Should it be applicable to your project, you may use our project template as a starting point.

Make shure you provide all information required to reproduce your project with a senseBox:edu kit!

File structure

Place your documentation file in the directory `edu/en/community_projects/` .

If you want to include additional resources, place them in a subfolder with the same name:

```
mobile-weatherstation.md
mobile-weatherstation/overview.jpg
mobile-weatherstation/mobile-weatherstation.ino
```

Filenames may not include any spaces!

Content license

Your contribution will be added under the same CC BY-SA 4.0 license as our content. This means, that the content may be freely adopted by others, as long as the authors name is provided with the content.

Uploading the documentation

To provide us your documentation, you can insert it in the book yourself by submitting a pull request on GitHub. The source code of this book is hosted on GitHub. Fork this repository, add your content there, and create a new pull request.

In case you are unfamiliar with the GitHub process, have a look at a GitHub contribution guide.

Alternatively to working directly on GitHub, there is also a Gitbook.com editor, wich might simplify the process. However we didn't try that one.

If none of that works, just send us your contribution written in markdown via mail.

> Having issues? Mail our support!

Thank you for your contribution!

Downloads

In this area various downloads and helpful resources regarding the senseBox are listed.

Libraries

A package of all the required libraries for the various sensors we use is provided here. Note that we use a customized `Ethernet.h` library, as the stock-library is not compatible with our ethernet shield.

Datasheets

Most manufacturers provide datasheets for their sensors and other components, which provide specifications and further insights into the inner workings of the devices:

Sensor	Description	Manufacturer	Download
BMP280	air-pressure sensor	Bosch	Datasheet
HC-SR04	supersonic distance-sensor	KT-Electronic	Datasheet
HDC1008	temperature- & humidity-sensor	Texas Instruments	Datasheet
TSL4531	digital lightintensity sensor	TAOS Texas Advanced Optoelectronics Solutions	Datasheet
VEML6070	UV-light sensor	VISHAY	Datasheet
GP2Y0A21YK0F	IR distance sensor	Sharp	Datasheet

Documentation as PDF

This book is also available as printable PDF!