Welcome!

Thank you for purchasing our *AZ-Delivery DS18B20 Temperature Sensor*. On the following pages, we will introduce you to how to use and set-up this handy device.

Have fun!

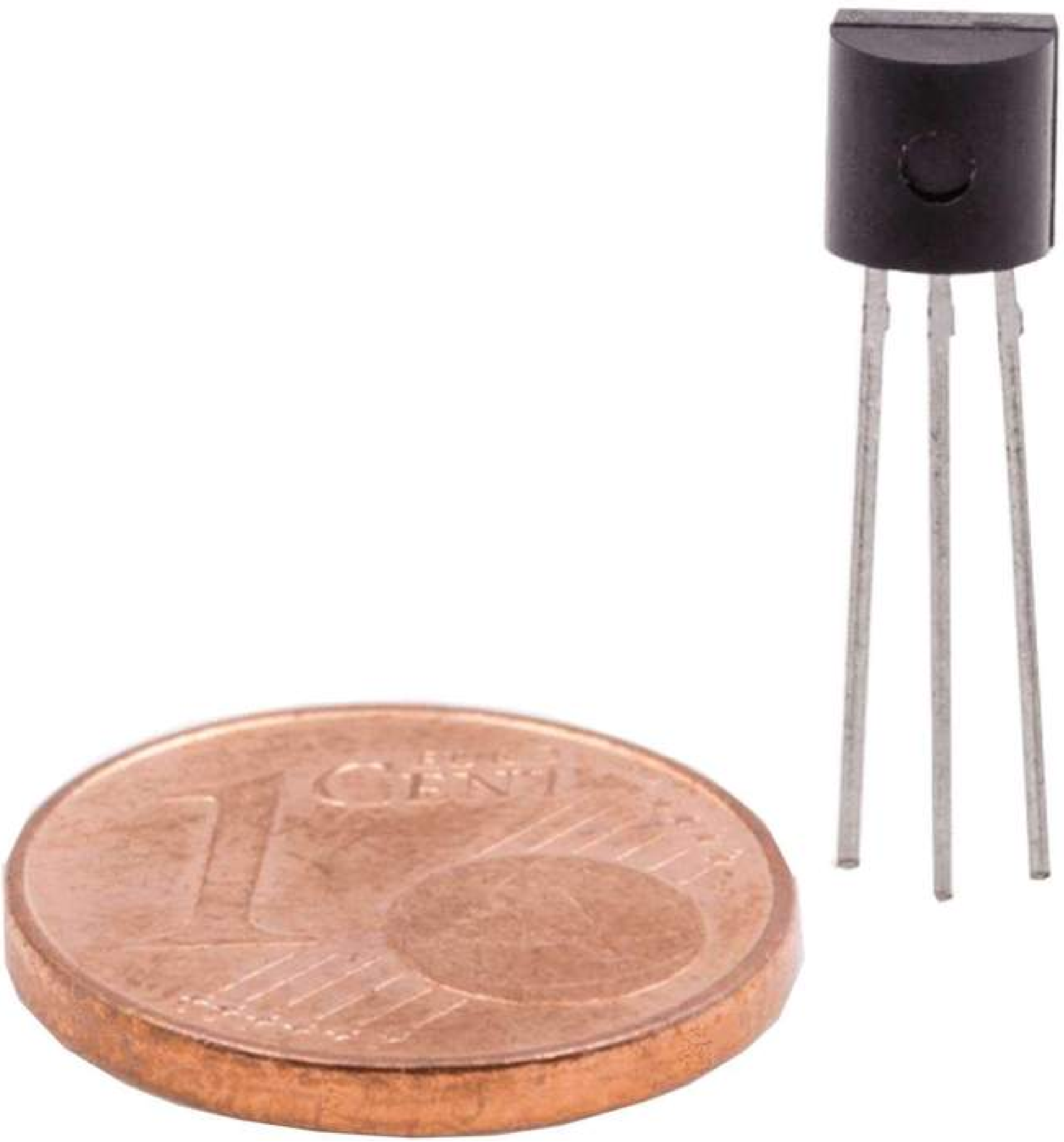


Table of Contents

Introduction

The DS18B20 is a digital temperature sensor that provides *9* to *12* bits digital temperature measurements and has an alarm function with nonvolatile user programmable upper and lower trigger points. The sensor communicates over a bus that requires only one data pin, power supply pin and ground pin for communication with a microcontroller.

Each sensor has a *64* bit serial address, which allows multiple sensors to function on the same bus. Thus, it is simple to use one microcontroller to control many sensors distributed over a large area.

The sensor does not require standby power, which means that when temperature data is not read, the sensor does not use power at all.

Measurement temperature range is from *-55°C* to *+125°C* (*67°F* to *257°F*), with an accuracy of ±*0.5°C* (*9* bit); ±*0.25°C* (*10* bit); ±*0.125°C* (*11* bit); and ±*0.0625°C* (*12* bit) resolution.

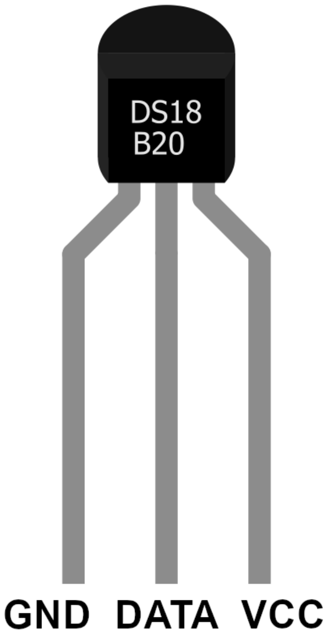
NOTE: If you experience any communication issues, try adding a *4.7k*Ω pull-up resistor on the *DATA* pin.

NOTE: The relevant technical data on the interface does not mention the maximum number of sensors that can be linked on the same interface, but in practical application, this number is not as high, and you should pay attention.

NOTE: There is a cable length limitation that should be taken into consideration when using long distance communications. You should pay attention to cable distributed capacitance and resistance.

NOTE: The DS18B20 and ordinary transistors look similar, so be careful not to regard it as a transistor to avoid damage!

The pinout



***VCC* pin** - supplies power for the sensor. Although supply voltage can range between *3.3V* and *5.5V*; a *5V* supply is recommended. In case of a *5V* power supply. A cable that connect sensor and microcontroller can be up to *20* meters long. However, with *3.3V* supply voltage, cable length should not be longer than one meter, otherwise, the line voltage drop will lead to errors in measurement.

***DATA* pin** - is the data pin, and it is used for communication between the sensor and the microcontroller (can be connected on One-Wire interface).

***GND* pin** **-** is ground pin and should be connected to the common ground, or 0V (on Atmega328P Board or Raspberry Pi).

How to set-up Arduino IDE

If you did not install Arduino IDE already, follow the link:
https://www.arduino.cc/en/Main/Software
and download installation file for your operating system .

For *Windows* users, double click on downloaded *.exe* file and follow instructions in installation window.

For *Linux* users, download a file with the extension *.tar.xz*, which you need to extract. When you extract it, go to the extracted directory, and open terminal in that directory. You need to run two *.sh* scripts, fiirst called *arduino-linux-setup.sh* and second called *install.sh*.

To run the first script in terminal, run the following command:

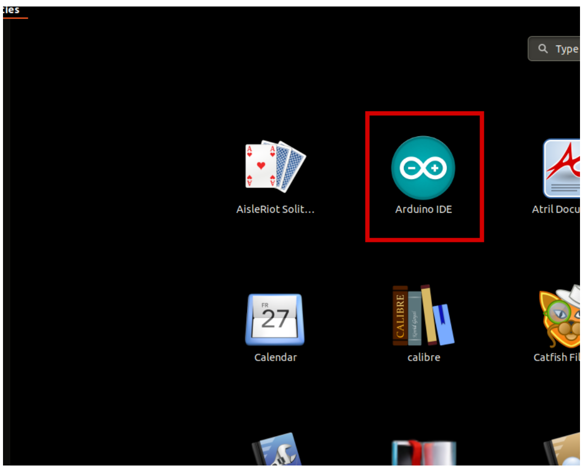
sh arduino-linux-setup.sh user_name

user_name - is the name of a super user in the Linux operating system. You will be prompted to provide password for the super user. Wait for a few minutes for script to complete everything.

After installation of the first script, you have to run the second script called *install.sh* script. In the terminal, run the following command:

sh install.sh

After the installation of these scripts, go to the *All Apps*, where you can find the *Arduino IDE* installed.

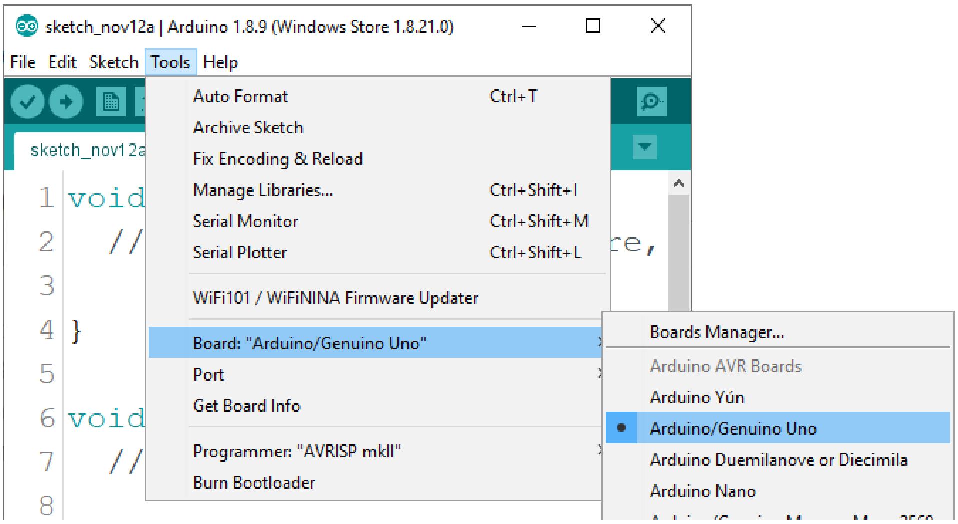


Almost all operating systems come with a text editor preinstalled (for example `Windows` comes with the `Notepad`, `Linux Ubuntu` comes with the `Gedit`, `Linux Raspbian` comes with the `Leafpad` etc.). All of these text editors are perfectly fine for the purpose of the eBook.

Next thing is to check if your PC can detect your microcontroller board. Open freshly installed Arduino IDE, and go to:

Tools > Board > {your board name here}

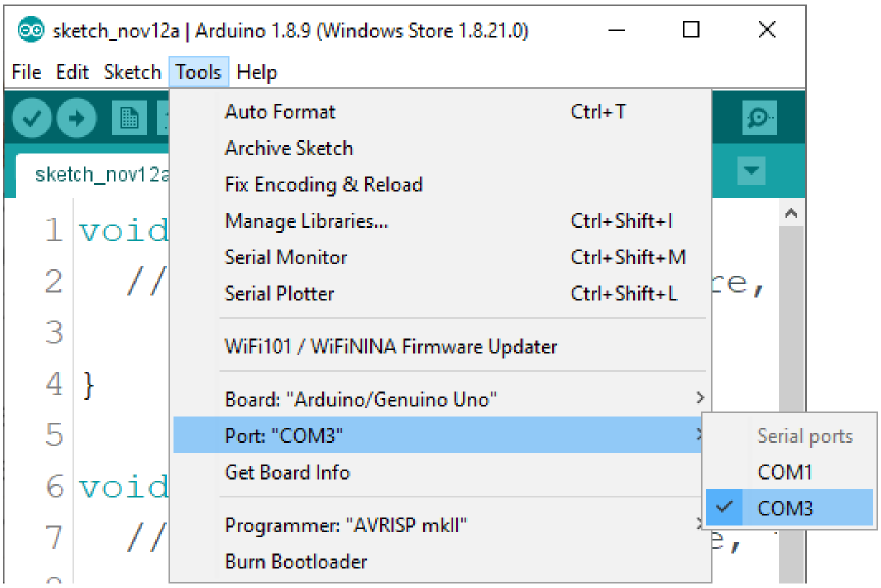
{your board name here} should be the `Arduino/Genuino Uno`, as you can see on the following image:



You need to select the port on which the microcontroller board is connected. Go to: *Tools > Port > {port name goes here}*

and if you connected the microcontroller board on the usb port you should see a port name.

If you are using Arduino IDE on Windows, port names are as follows:



For *Linux* users, port name is */dev/ttyUSBx* for example, where *x* represents integer number between *0* and *9*, for instance.

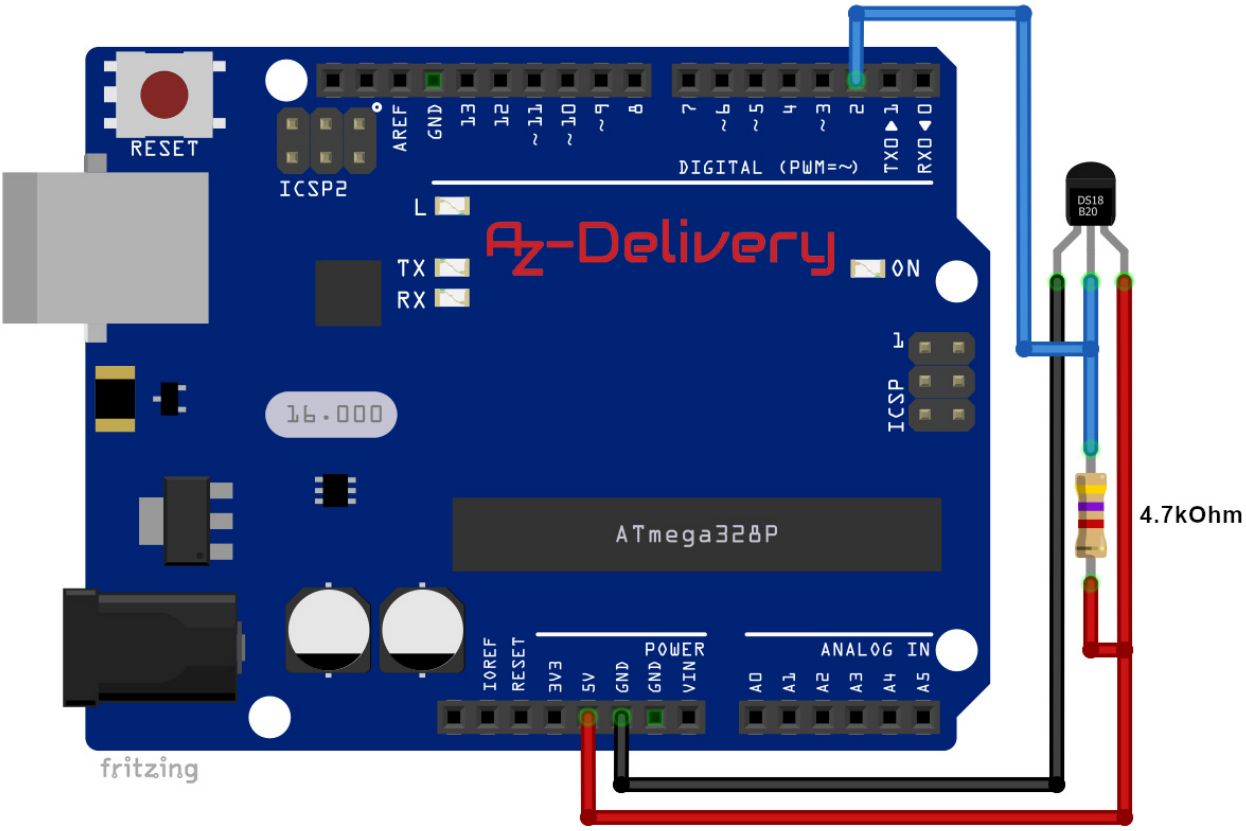
How to set-up the Raspberry Pi and the Python

For the Raspberry Pi, you will first have to install operating system on it, then to set-up everything, so that you can use it in the *Headless* mode. The *Headless* mode enables you to remotely connect to the Raspberry Pi, without the need for *PC* screen Monitor, mouse or keyboard. Only things that you need for this mode are the Raspberry Pi, power supply and internet connection. All of this is explained in detail in the free eBook *Raspberry Pi Quick Startup Guide*, which can be found on our site:
https://www.az-delivery.de/products/raspberry-pi-kostenfreies-e-book?ls=en

The *Raspbian* operating system comes with the *Python* preinstalled.

Connecting the sensor with Atmega328P Board

Connect the DS18B20 sensor with the microcontroller board as shown on the following connection diagram:

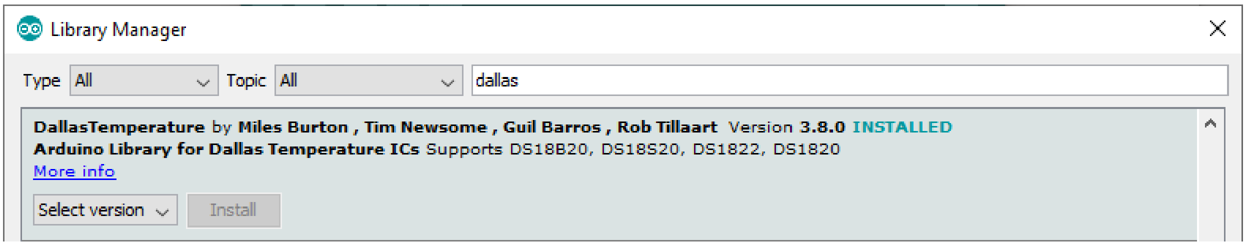


DS18B20 pin	**>**	**Microcontroller pin**	
DATA	>	D2	**Blue wire**
VCC	>	5V	**Red wire**
GND	>	GND	**Black wire**

NOTE: Pull up *4.7kΩ* resistor is connected between *OUT* pin and *VCC* pin.

Library for Arduino IDE

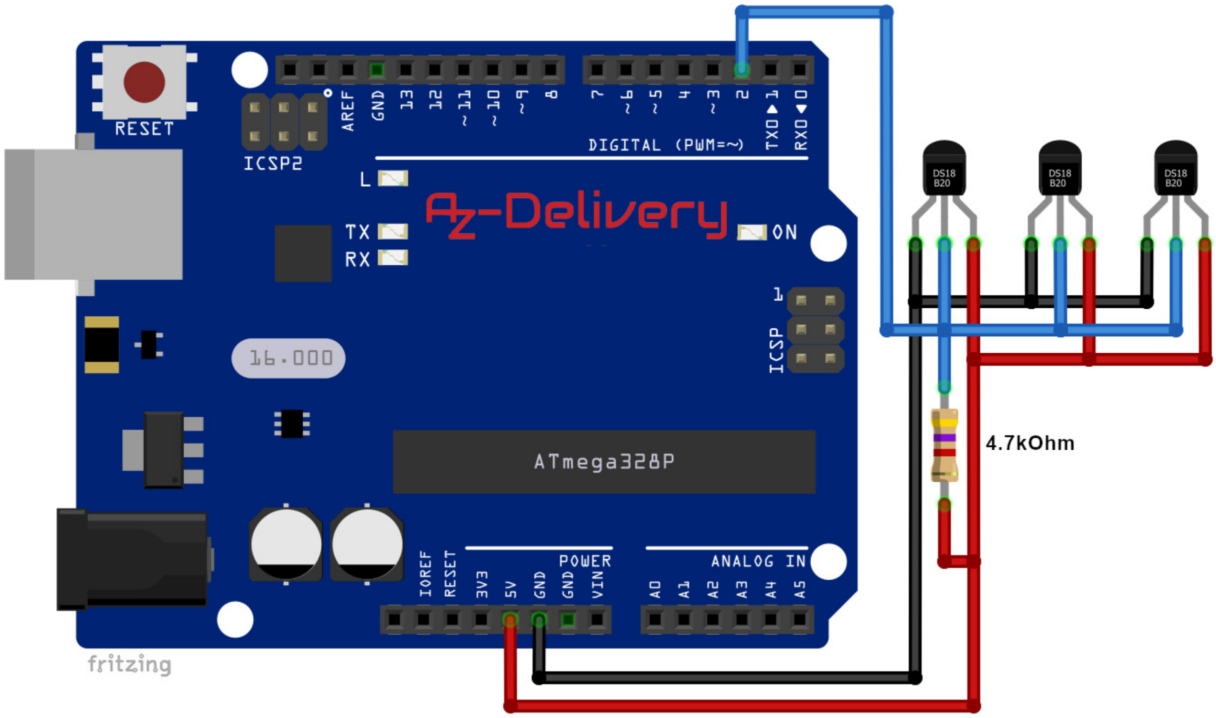
To use the DS18B20 sensor with Atmega328P Board, first we have to download a library for it. Go to: *Tools > Manage Libraries*
When a new window opens, type `Dallas` in the search box and download the library *`DallasTemperature`* by *`Miles Burton, Tim Newsome, Guil Barros and Rob Tillaart`*, as shown on the following image:



Now, go to: *File > Examples > DallasTemperature > ...* and you will find many sketch examples. We used and modified a sketch called *`Multiple`* in order to read temperature data from three different DS18B20 sensors.
File > Examples > DallasTemperature > Multiple

If you use only one DS18B20 sensor, a sketch called *`Simple`* is good enough:
File > Examples > DallasTemperature > Simple

Connect three DS18B20 sensors with the Atmega328P board as shown on the following connection diagram:



Sketch example

The following is the sketch example for three DS18B20 sensors on the same One-Wire interface:

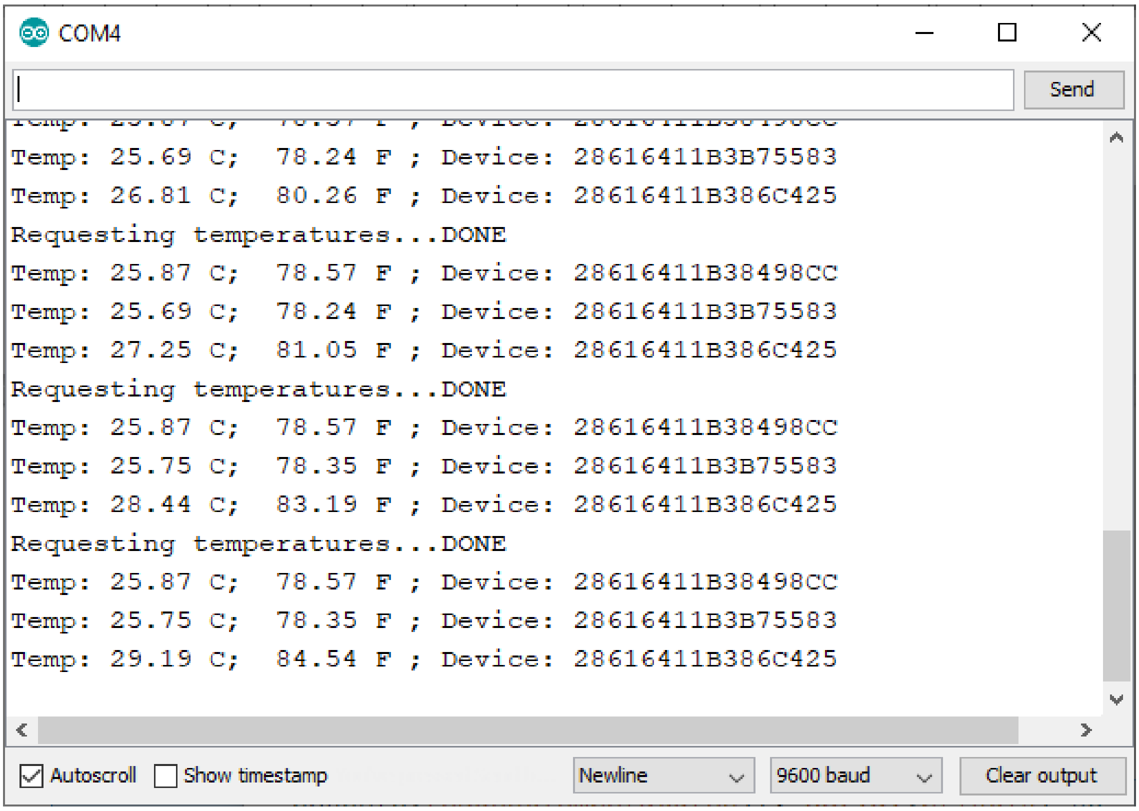
```
#include <OneWire.h>
#include <DallasTemperature.h>
#define ONE_WIRE_BUS 2 // Data wire is plugged into D2 pin
#define TEMPERATURE_PRECISION 12
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);
DeviceAddress one, two, three;

void setup() {
  Serial.begin(9600);
  Serial.println("Dallas Temperature IC Control Library Demo");
  sensors.begin();
  Serial.print("Locating devices...");
  Serial.print("Found ");
  Serial.print(sensors.getDeviceCount(), DEC);
  Serial.println(" devices.");
  Serial.print("Parasite power is: ");
  if(sensors.isParasitePowerMode()) {
    Serial.println("ON");
  }
  else {
    Serial.println("OFF");
  }
```

```
  // one tab
  if(!sensors.getAddress(one, 0)) {
    Serial.println("Unable to find address for Device 0"); }
  if(!sensors.getAddress(two, 2)) {
    Serial.println("Unable to find address for Device 2"); }
  if(!sensors.getAddress(three, 1)) {
    Serial.println("Unable to find address for Device 1"); }

  Serial.print("Device 0 Address: ");
  printAddress(one);
  Serial.println();
  Serial.print("Device 1 Address: ");
  printAddress(two);
  Serial.println();
  Serial.print("Device 2 Address: ");
  printAddress(three);
  Serial.println();
  sensors.setResolution(one, TEMPERATURE_PRECISION);
  sensors.setResolution(two, TEMPERATURE_PRECISION);
  sensors.setResolution(three, TEMPERATURE_PRECISION);
  Serial.print("Device 0 Resolution: ");
  Serial.print(sensors.getResolution(one), DEC);
  Serial.println();
  Serial.print("Device 1 Resolution: ");
  Serial.print(sensors.getResolution(two), DEC);
  Serial.println();
  Serial.print("Device 2 Resolution: ");
  Serial.print(sensors.getResolution(three), DEC);
  Serial.println();
}
```

```
void printAddress(DeviceAddress deviceAddress) {
  for(uint8_t i = 0; i < 8; i++) {
    if(deviceAddress[i] < 16) Serial.print("0");
    Serial.print(deviceAddress[i], HEX);
  }
}
void printTemperature(DeviceAddress deviceAddress) {
  float tempC = sensors.getTempC(deviceAddress);
  Serial.print("Temp: ");
  Serial.print(tempC);
  Serial.print(" C;  ");
  Serial.print(DallasTemperature::toFahrenheit(tempC));
  Serial.print(" F");
}
void printResolution(DeviceAddress deviceAddress) {
  Serial.print("Resolution: ");
  Serial.println(sensors.getResolution(deviceAddress));
}
void printData(DeviceAddress deviceAddress) {
  printTemperature(deviceAddress);
  Serial.print(" ; Device: ");
  printAddress(deviceAddress);
  Serial.println();
}
```

```
void loop() {
  Serial.print("Requesting temperatures...");
  sensors.requestTemperatures();
  Serial.println("DONE");
  printData(one);
  printData(two);
  printData(three);
  delay(1000);
}
```

Upload the sketch to the microcontroller board and open Serial Monitor (*Tools* > *Serial Monitor*). The result should look like the output as shown on the image below:



We use *ONE_WIRE_BUS* variable to define on which digital pin we will connect the One-Wire interface. For the purpose of this eBook, the value of the *ONE_WIRE_BUS* variable is set to *D2*, but you can use any other digital pin of the microcontroller board, except the ones used in Serial Interface, *D0* and *D1* (it is a recommendation, you can use them, but you have to be sure that these pins are disconnected when you are uploading sketches).

We used the *TEMPERATURE_PRECISION* variable to set precision for DS18B20 sensors. Number saved in this variable is a digital conversion number in bits and it can be in a range from *9* to *12*, any other number will result in an error. For the purpose of this eBook, we set it to the maximum value (*12*).

We used the following line of code:

```
DeviceAddress one, two, three
```

to create variables for sensor addresses, and in our example we created three.

We defined and created *oneWire* object, used for the bus interface:

```
OneWire oneWire(ONE_WIRE_BUS);
```

Then we used *oneWire* object to define and create *sensors* object, which is used for all connected sensors:

```
DallasTemperature sensors(&oneWire)
```

To initialize *sensors* object we used the following line of code:

```
sensors.begin()
```

With that line of the code *sensors* object detects all sensors connected on the bus interface. It also detects all the addresses of sensors.

Now we can check if sensors are working properly, by using the following lines of code for every sensor we connect to the One-Wire interface:

```
if(!sensors.getAddress(one, 0)) {
  Serial.println("Unable to find address: Device 0"); }
```

where *one* is the address of the first sensor.

To set-up analog to digital conversion precision of the specific sensor we used the following line of code:

```
sensors.setResolution(one, TEMPERATURE_PRECISION)
```

If you want to read analog to digital conversion precision of the specific sensor, you can use the following line of code:

```
sensors.getResolution(one)
```

The function returns hexadecimal value, and to convert it to a decimal value we use the following line of code:

```
Serial.print(sensors.getResolution(one), DEC);
```

In order to read the temperature data, we first have to request all data from all sensors, by using the following line of code:

```
sensors.requestTemperatures();
```

Only after that line of code, we can read data of a particular sensor, using the following line of code:

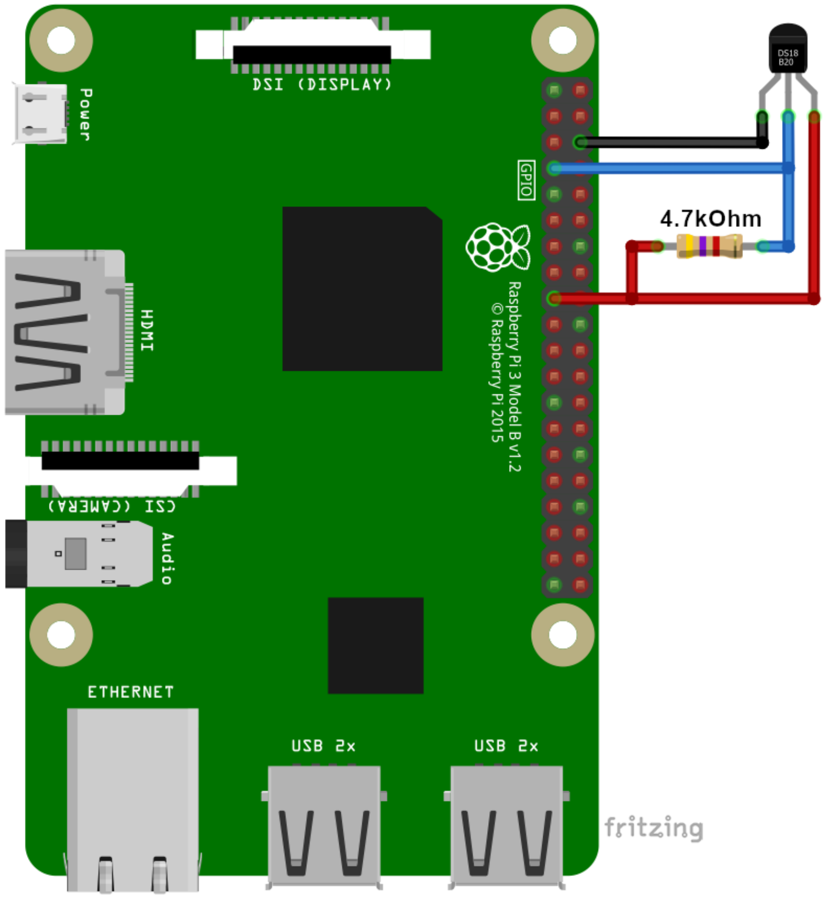
```
float tempC = sensors.getTempC(deviceAddress);
```

where we pass *`deviceAddress`* argument to the function in order to read temperature data from a specific sensor. This data is temperature value in Celsius, and to convert it into Fahrenheit we used the following line of code:

```
DallasTemperature::toFahrenheit(tempC)
```

Connecting the sensor with Raspberry Pi

Connect the DS18B20 sensor with the Raspberry Pi as shown on the following connection diagram:



DS18B20 pin	**>**	**Raspberry Pi pin**		
GND	>	GND	[pin 6]	**Black wire**
DATA	>	GPIO4	[pin 7]	**Blue wire**
VCC	>	3V3	[pin 17]	**Red wire**

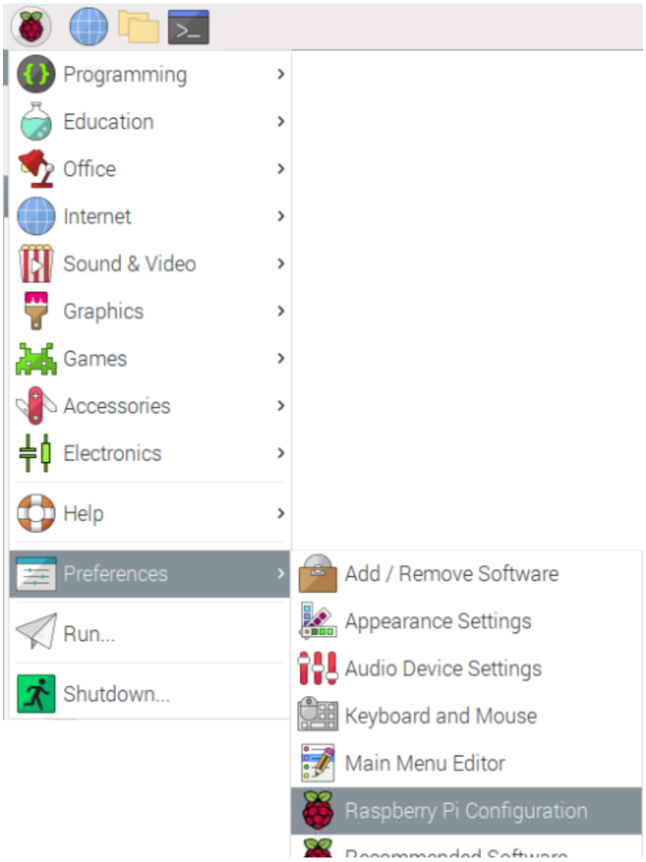
NOTE: Pull up *4.7kΩ* resistor is connected between *OUT* pin and *3V3* pin.

Enabling the interface

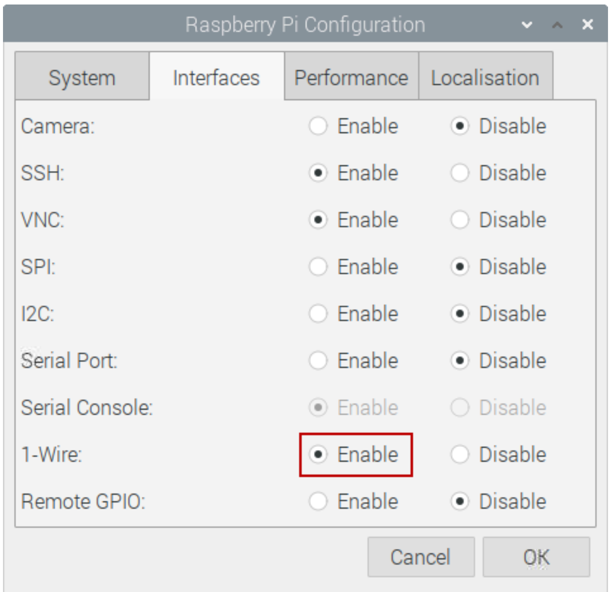
Before we can use the DS18B20 sensor with the Raspberry Pi, first we have to enable the bus interface in Raspbian. By default, the hardware bus interface is on pin GPIO4 (pin 7), but we first have to enable it. In order to enable the bus interface, go to:

Applications Menu > Preferences > Raspberry Pi Configuration

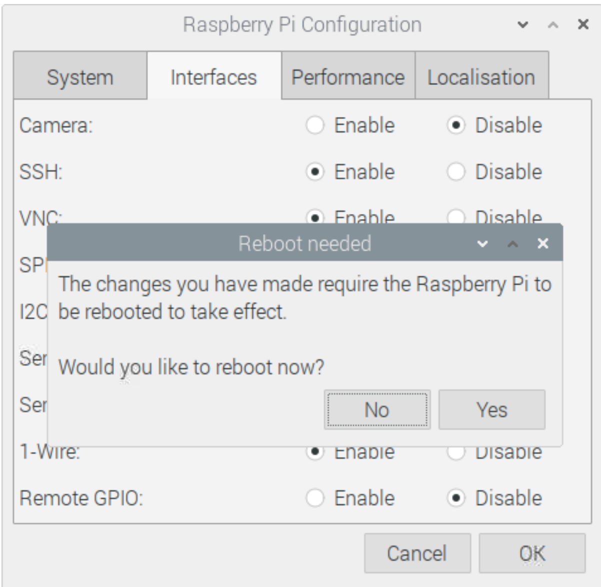
as shown on the following image:



When the new window appears, open `Interfaces` tab and search for radio buttons called *`1-Wire`*, and select the *`Enable`* radio button, as shown on the following image:



You will be prompted to reboot the system.



When Raspbian is booted again, open the terminal, and run the following commands, one by one:

sudo modprobe w1-gpio

sudo modprobe w1-therm

cd /sys/bus/w1/devices/

and when you run the following command:

ls

the output in the terminal should be as follows:

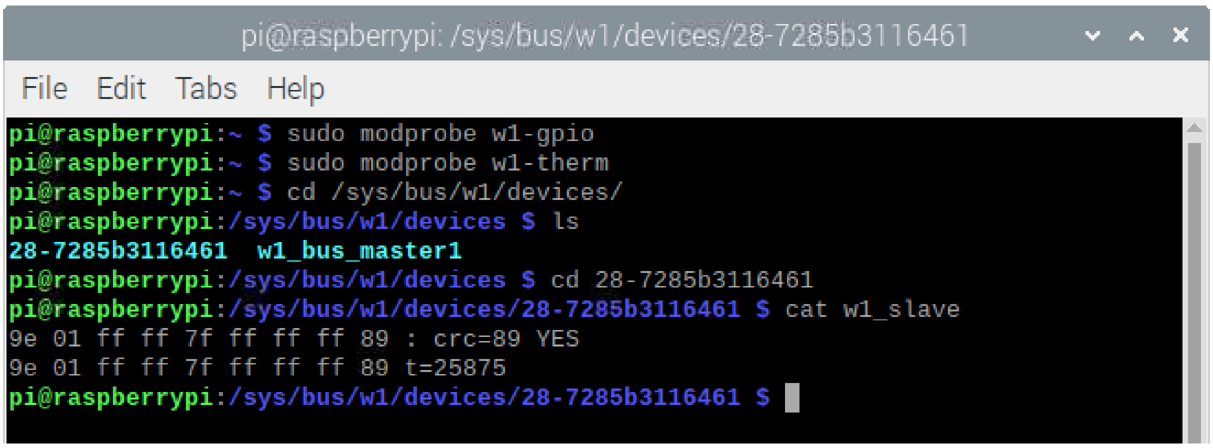
28-7285b3116461 and **w1_bus_master1**

the first number *28-7285b3116461* will be different for you, because this is the serial address of the specific sensor, and each sensor has its own unique serial address. Now, to test if everything works, run these two commands:

cd 28-7285b3116461 - a number or serial address from the last page

cat w1_slave

The result should look like the output as shown on the following image:



t=25875 - this is the temperature data in °C (Celsius) = 25.875°C.

Enabling multiple interfaces

To enable the bus interface, without a graphic user interface (GUI), before rebooting your Raspberry Pi, to the file located on:
/boot/config.txt
you need to add the following line:
`dtoverlay=w1-gpio`
or
`dtoverlay=w1-gpio,gpiopin=x`
where *x* is a custom pin, if you would like to use it (default is GPIO4 [pin 7], like we mentioned in the previous chapter).

Newer kernels (*4.9.28* and later) allow you to use dynamic overlay loading, including creating multiple One-Wire interfaces to be used at the same time:

```
sudo dtoverlay w1-gpio gpiopin=4 pullup=0 # pin 7
sudo dtoverlay w1-gpio gpiopin=17 pullup=0 # pin 11
sudo dtoverlay w1-gpio gpiopin=27 pullup=0 # pin 13
```

Once any of the steps above have been performed, and discovery is complete you can list the devices that the Raspberry Pi has discovered via all One-Wire interfaces by running the following command in the terminal:

```
ls /sys/bus/w1/devices/
```

NOTE: Using **`w1-gpio`** on the Raspberry Pi typically needs a *4.7k*Ω pull-up resistor connected between the GPIO pin and a *3.3V* supply.

Python script for using multiple DS18B20 sensors

We choose to split the code into two scripts, because of the better readability. The following is a code for the class subscript:

```
import os
import glob
import time
class DS18B20:

   def __init__(self):
      os.system('modprobe w1-gpio')
      os.system('modprobe w1-therm')
      base_dir = '/sys/bus/w1/devices/'
      device_folder = glob.glob(base_dir + '28*')
      self._count_devices = len(device_folder)
      self._devices = list()
      i = 0
      while i < self._count_devices:
         self._devices.append(device_folder[i] + '/w1_slave')
         i += 1

   def device_names(self):
      names = list()
      for i in range(self._count_devices):
         names.append(self._devices[i])
         temp = names[i][20:35]
         names[i] = temp
```

```
        return names

# (one tab)
    def _read_temp(self, index):
        f = open(self._devices[index], 'r')
        lines = f.readlines()
        f.close()
        return lines

    def tempC(self, index = 0):
        lines = self._read_temp(index)
        retries = 5
        while (lines[0].strip()[-3:] != 'YES') and (retries > 0):
            time.sleep(0.1)
            lines = self._read_temp(index)
            retries -= 1
        if retries == 0:
            return 998

        equals_pos = lines[1].find('t=')
        if equals_pos != -1:
            temp = lines[1][equals_pos + 2:]
            return float(temp) / 1000
        else:
            return 999 # error

    def device_count(self):
        return self._count_devices
```

(The most of code in the script is modified from the script on the adafruit site)

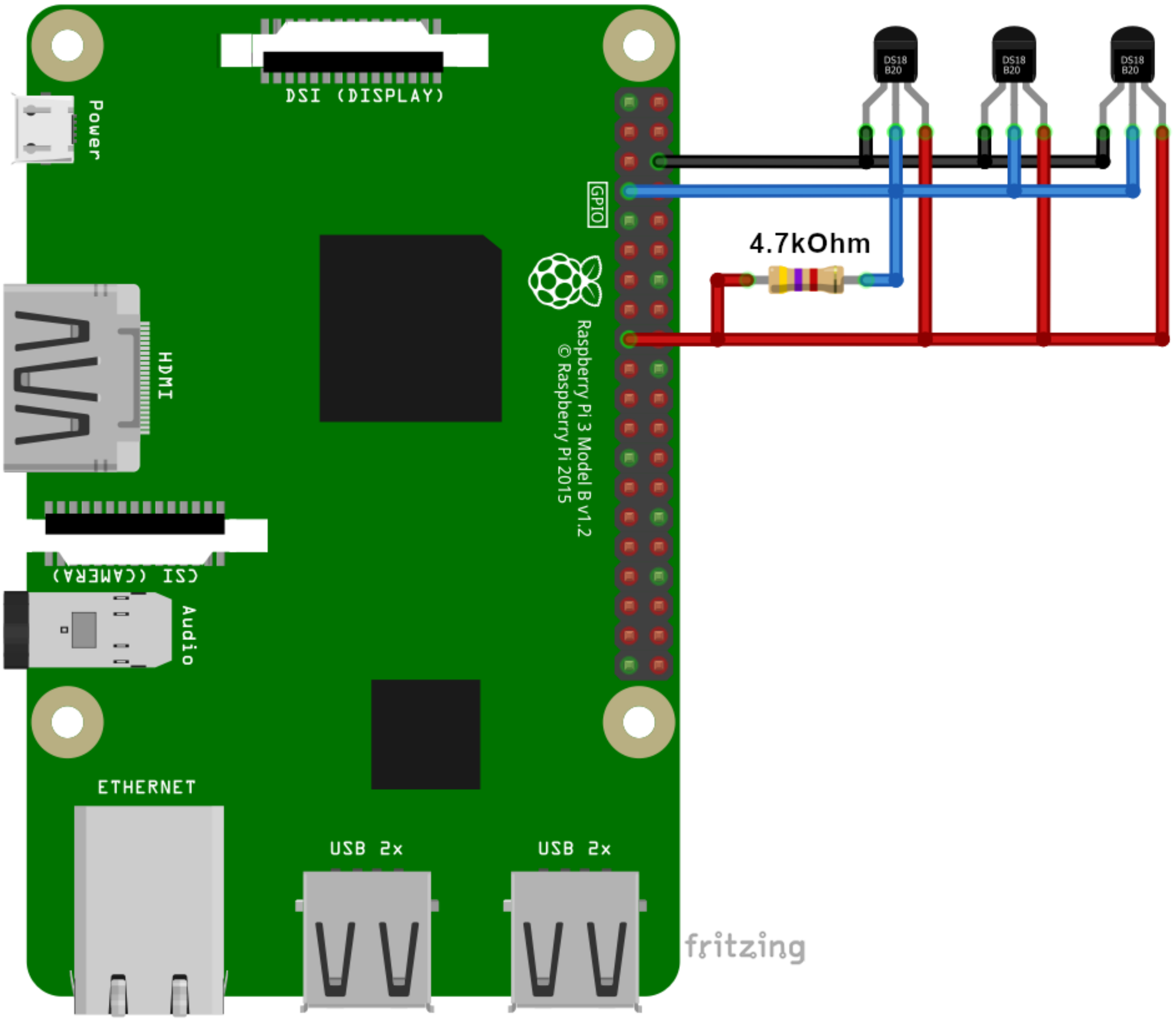
Save the script by the name *DS18B20classfile.py*.

The following is a code for the main script:

```
import time
from DS18B20classFile import DS18B20

degree_sign = u'\xb0' # degree sign
devices = DS18B20()
count = devices.device_count()
names = devices.device_names()

print('[Press CTRL+C to end the script]')
try: # Main program loop
    while True:
        i = 0
        print('\nReading temperature, number of sensors: {}'
                                .format(count))
        while i < count:
            container = devices.tempC(i)
            print('{}. Temp: {:.3f}{}C, {:.3f}{}F of the device {}'
                                .format(i+1, container, degree_sign,
                                container * 9.0 / 5.0 + 32.0, degree_sign,
                                names[i]))
            i = i + 1

        time.sleep(1)

# Scavenging work after the end of the program
except KeyboardInterrupt:
    print('\nScript end!')
```

Save the script by the name *`DS18B20multiple.py`* in the same directory where you saved the first script.

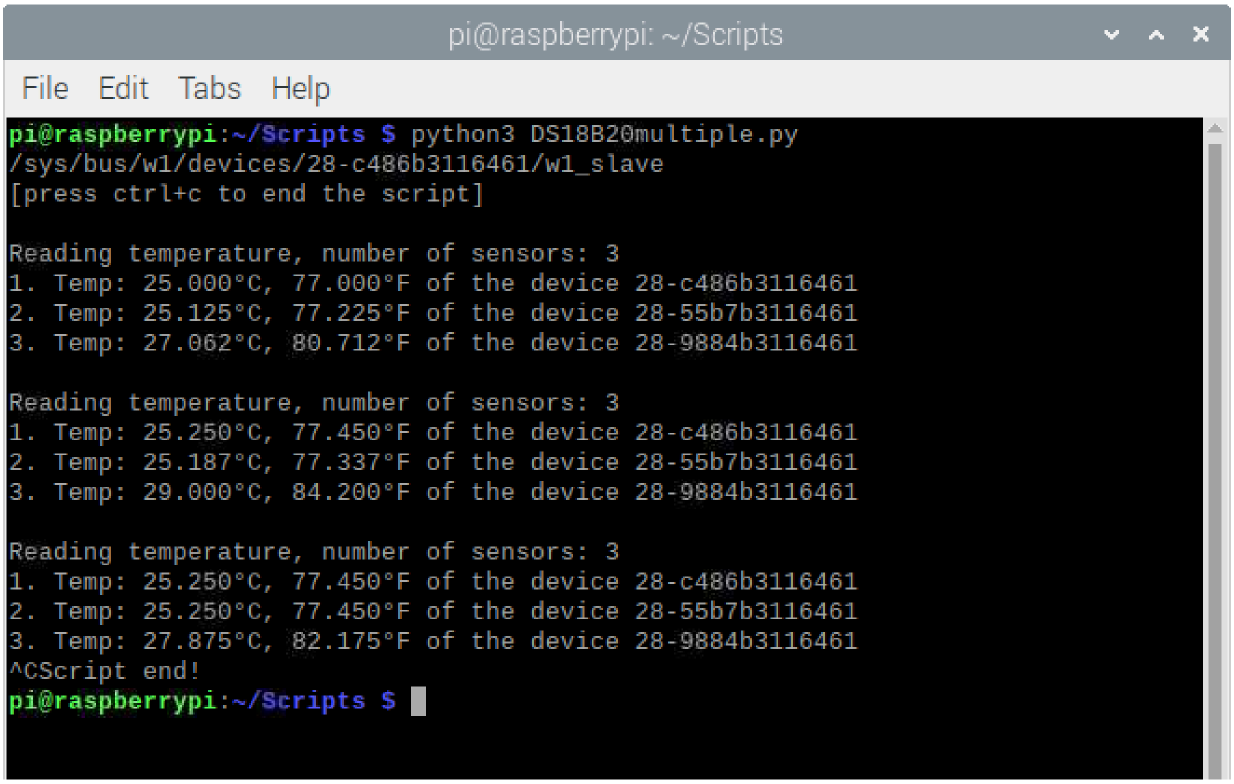
For example, we connected three DS18B20 sensors on the same One-Wire interface of the Raspberry Pi as shown on the following connection diagram:



To run the main script, open thr terminal in the directory where you saved both scripts, and run the following command:

`python3 DS18B20multiple.py`

The result should look like the output as shown on the image below:



To stop the script press CTRL + C on the keyboard.

You can easily use the script for one or multiple DS18B20 sensors.

You have done it!

Now you can use your module for various projects.

Now is the time to learn and make the Projects on your own. You can do that with the help of many example scripts and other tutorials, which you can find on the internet.

If you are looking for the high quality microelectronics and accessories, AZ-Delivery Vertriebs GmbH is the right company to get them from. You will be provided with numerous application examples, full installation guides, eBooks, libraries and assistance from our technical experts.

https://az-delivery.de

Have Fun!

Impressum

https://az-delivery.de/pages/about-us