

# AZ-Delivery

## Welcome!

Thank you for purchasing our *AZ-Delivery DS18B20 Temperature Sensor Sonde with 3m cable*. On the following pages, we will introduce you to how to use and set-up this handy device.

**Have fun!**





The DS18B20 temperature sensor is hermetically inclosed in the sonde. The tip of the sonde is made of stainless steel and it is best suitable for measuring temperature in wet environments.

The DS18B20 is a digital temperature sensor that provides 9 to 12 bit digital temperature measurements and has an alarm function with nonvolatile user programmable upper and lower trigger points. The sensor communicates over One-Wire bus that requires only one data pin, power supply pin and ground pin for communication with a microcontroller. In addition, the sensor can derive power directly from the data line (this mode is called “*parasite power*” mode), eliminating the need for an external power supply.

Each sensor has a unique 64 bit serial address, which allows multiple sensors to function on the same One-Wire bus. Thus, it is simple to use one microcontroller to control many sensors distributed over a large area. Applications that can benefit from this feature include temperature monitoring systems inside buildings, equipment, or machinery, process monitoring and control systems.

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

Measurement temperature range is from  $-55^{\circ}\text{C}$  to  $+125^{\circ}\text{C}$  ( $67^{\circ}\text{F}$  to  $257^{\circ}\text{F}$ ), with an accuracy of  $\pm 0.5^{\circ}\text{C}$  (9 bit);  $\pm 0.25^{\circ}\text{C}$  (10 bit);  $\pm 0.125^{\circ}\text{C}$  (11 bit); and  $\pm 0.0625^{\circ}\text{C}$  (12 bit) resolution.



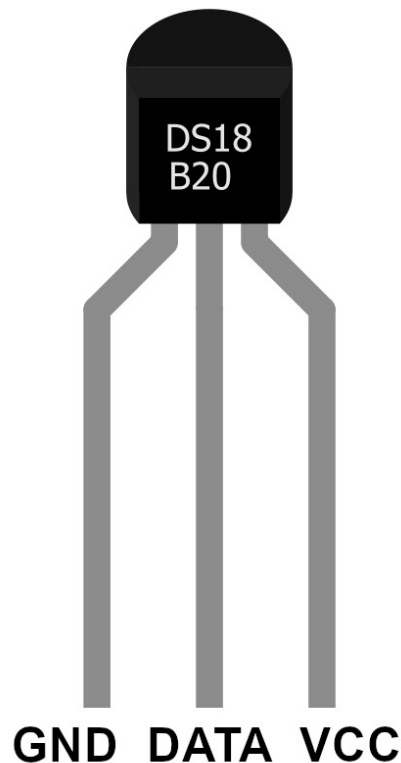
**NOTE:** If you experience any communication issues, try adding a  $4.7k\Omega$  pull-up resistor on the DATA pin.

**NOTE:** The relevant technical data on the One-Wire interface do not mention maximum number of sondes that can be linked on the same interface, but in practical application this number is not as high, and you should pay attention to that.

**NOTE:** There is a cable length limitation that should be taken in 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 sonde. Although supply voltage can range between 3.3V and 5.5V; 5V supply is recommended. In case of 5V power supply, you can use cable that connect sonde and microcontroller as long as 20 meters. However, with 3.3V supply voltage, cable length shall not be greater than one meter. Otherwise, the line voltage drop will lead to errors in measurement.

**"DATA" pin** - is data pin, and is used to communication between the sonde 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 Arduino or Raspberry Pi).

# Az-Delivery

## How to set-up Arduino IDE

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

### Download the Arduino IDE



The screenshot shows the Arduino IDE download page. On the left, there is a teal circle with a white infinity symbol containing a minus and a plus sign. To its right, the text reads: **ARDUINO 1.8.9**, followed by a description of the IDE as open-source software written in Java, based on Processing, and compatible with any Arduino board. It refers to the 'Getting Started' page for installation instructions. On the right side of the page, there is a teal sidebar with links for different operating systems: Windows (installer and ZIP file), Windows app (with a 'Get' button), Mac OS X (10.8 Mountain Lion or newer), and Linux (32 bits, 64 bits, ARM 32 bits, and ARM 64 bits). At the bottom of the sidebar are links for Release Notes, Source Code, and Checksums (sha512).

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

# Az-Delivery

For *Linux* users, you will download a file with 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, first called `"arduino-linux-setup.sh"` and second called `"install.sh"`.

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

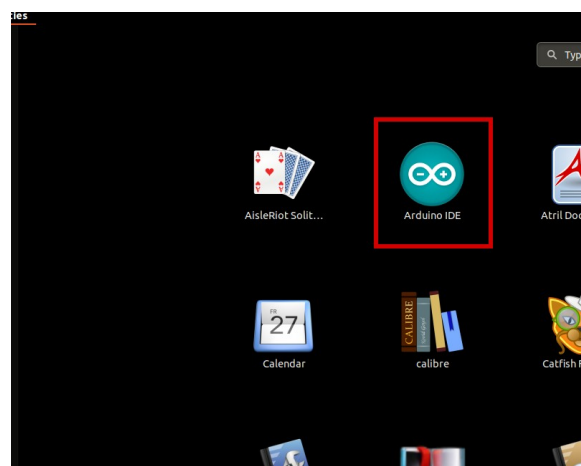
```
sh arduino-linux-setup.sh user_name
```

**user\_name** - is the name of 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 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.



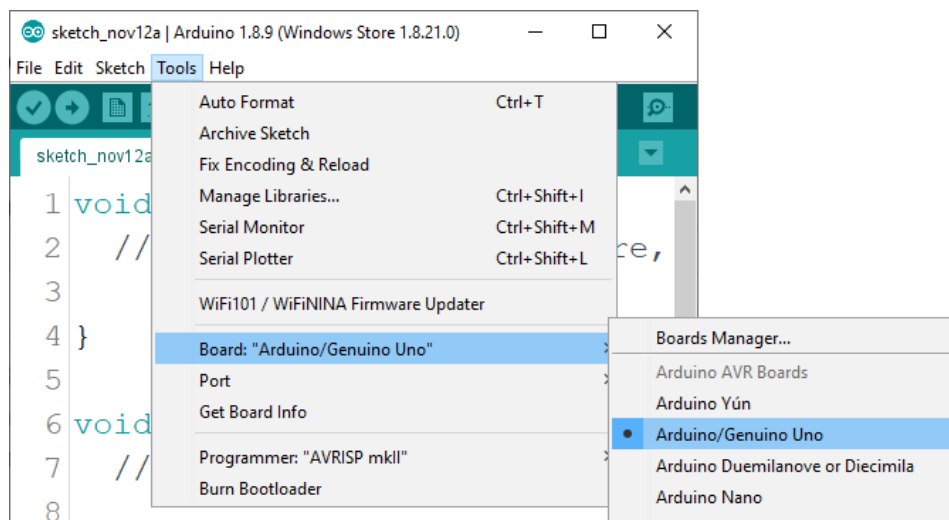
# Az-Delivery

Almost all operating systems come with a text editor preinstalled (*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 Arduino board. Open freshly installed Arduino IDE, and go to:

*Tools > Board > {your board name here}*

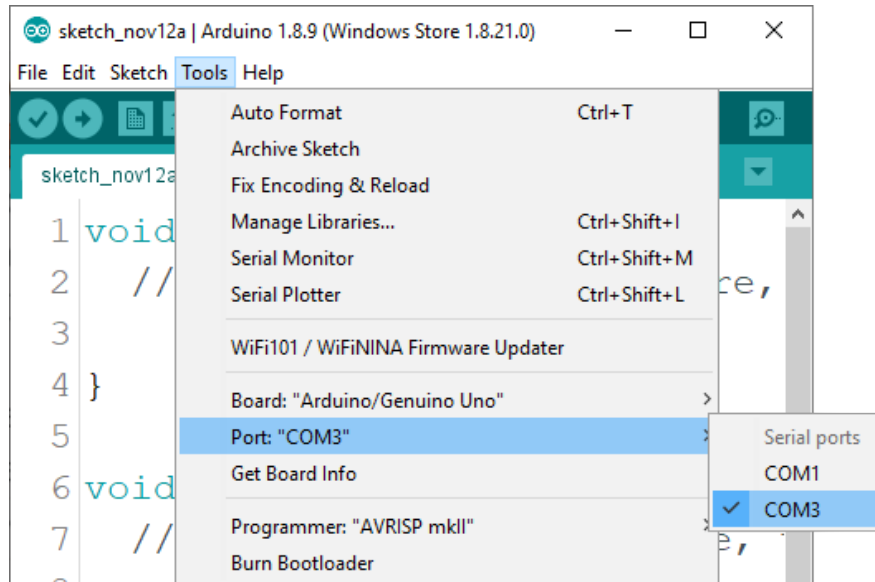
*{your board name here}* should be the *Arduino/Genuino Uno*, as you can see on the image below:



You need to select the port on which the Arduino board is connected. Go to *Tools > Port > {port name goes here}* and if you connected Arduino board on the usb port you should see a port name.

# Az-Delivery

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” represent integer number between 0 and 9, for instance.





## How to set-up the Raspberry Pi and the Python

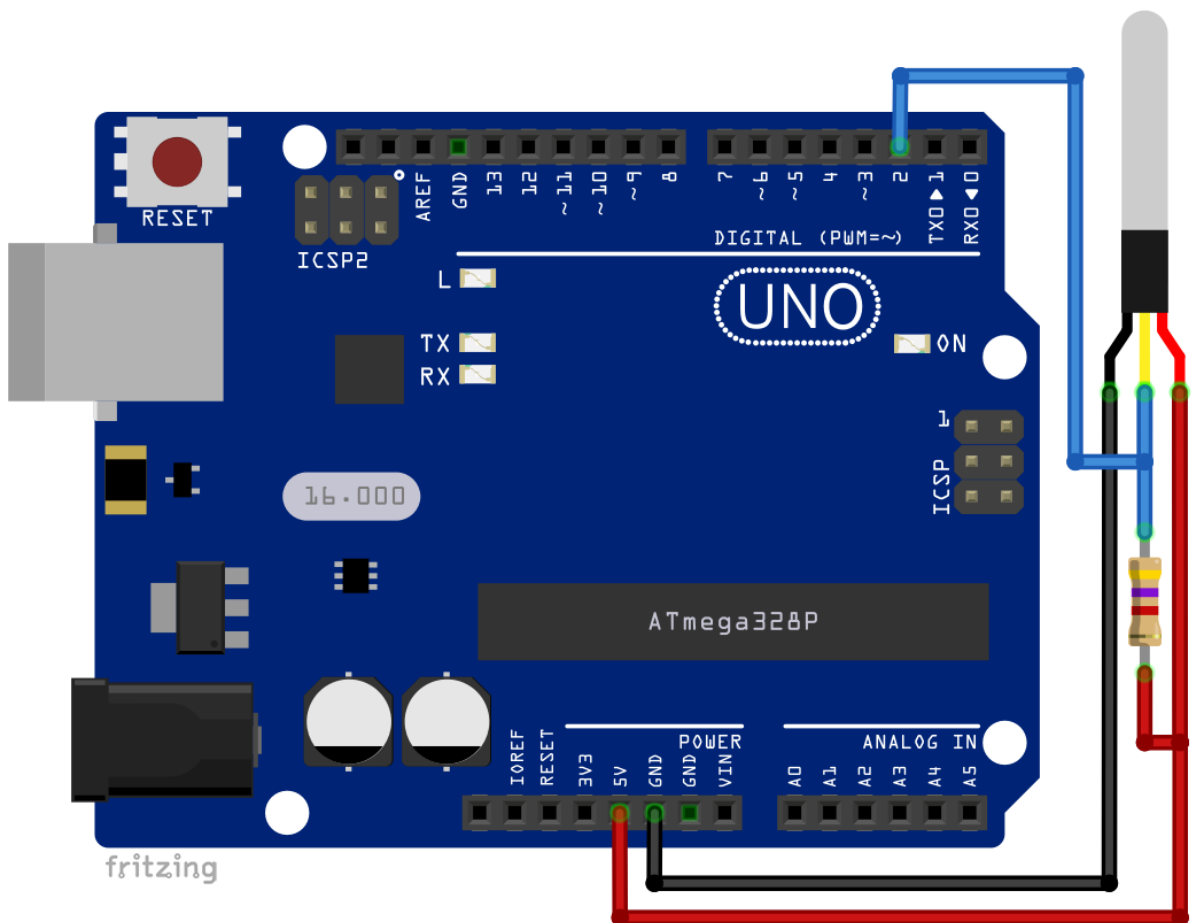
For the Raspberry Pi, you will first have to install operating system on it, then to set-up everything so that you can use it in the “*Headless*” mode. The *Headless* mode enables you to remotely connect to the Raspberry Pi, without the need for *PC* screen Monitor, mouse or keyboard. Only things that 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 *Rasbian* operating system comes with the *Python* preinstalled.

## Connecting the sonde with Uno

Connect the DS18B20 sonde with the Uno board as shown on the following connection diagram:



DS18B20 pin	>	Uno pin
DATA	>	D2
VCC	>	5V
GND	>	GND

Blue wire

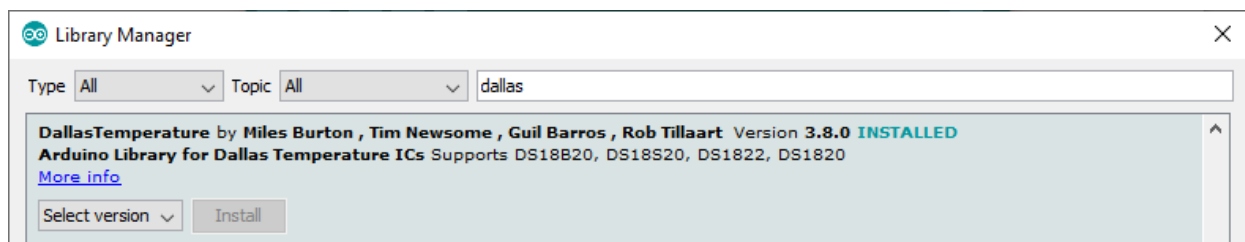
Red wire

Black wire

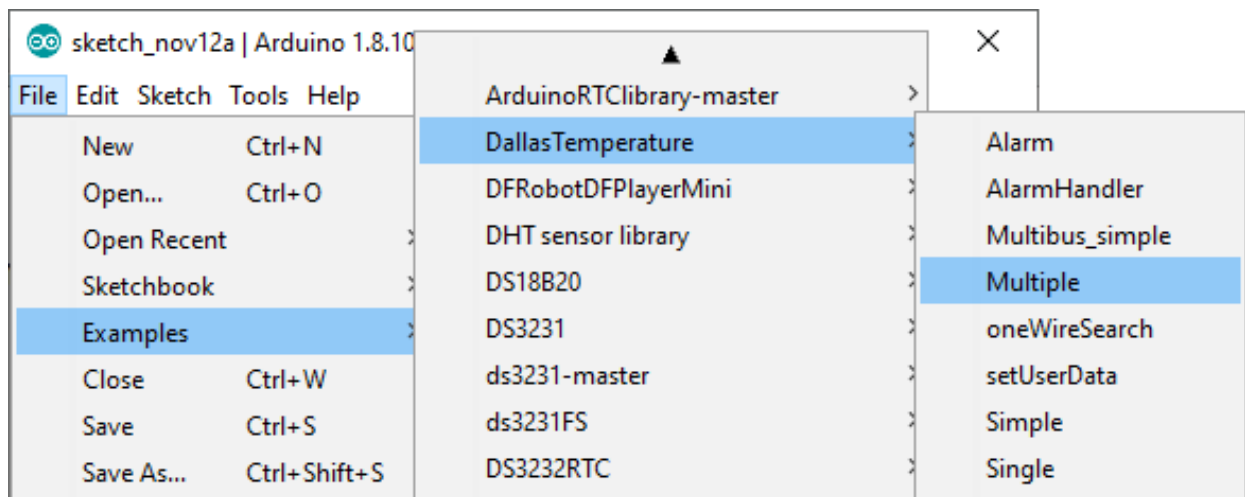
**NOTE:** Pull up 4.7kΩ resistor is connected between *DATA* pin and *VCC*.

# Az-Delivery

To use the DS18B20 sonde with Arduino boards, first we have to download a library for it. Go to: *Tools > Manage Libraries*, a new window will appear. 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:



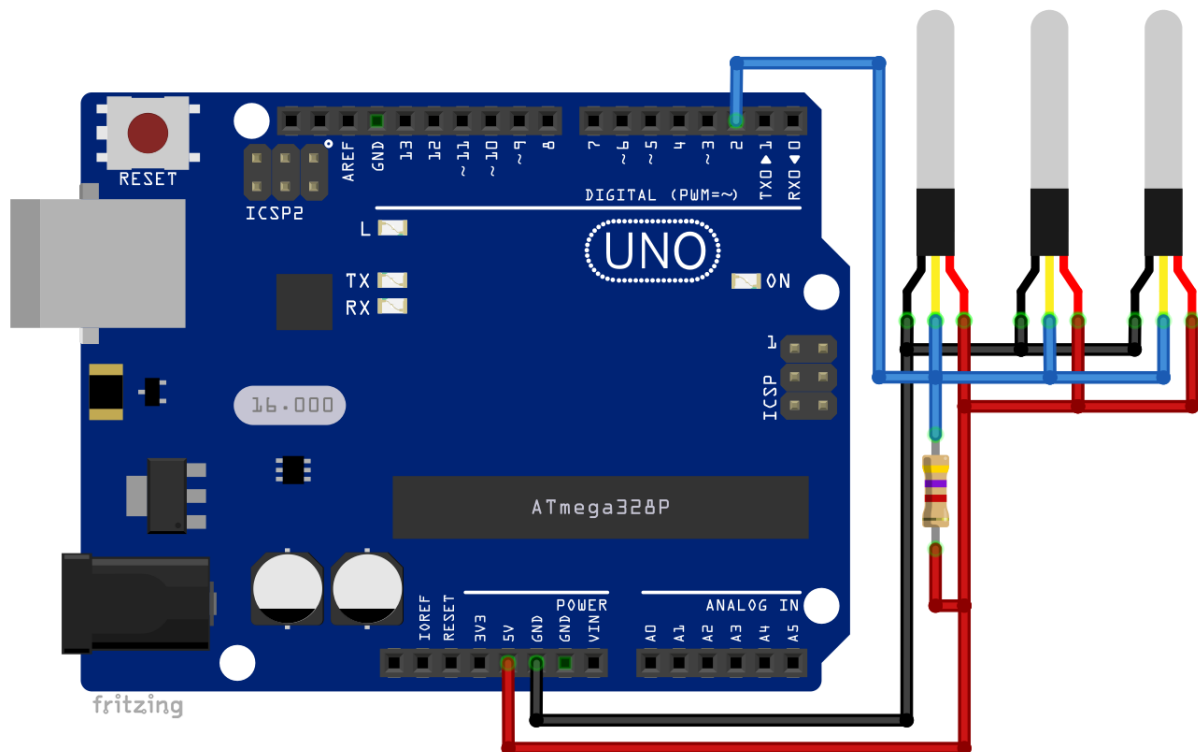
Now, go to: *File > Examples > DallasTemperature > ...* and you will find many sketch examples. We will use and modify a sketch called “*Multiple*” in order to read temperature data from three different DS18B20 sondes.



If you use only one DS18B20 sonde, sketch called “*Simple*” is good enough.

# Az-Delivery

Connect three DS18B20 sonde with the Uno board as shown on the following connection diagram:



The following is the sketch example for three DS18B20 sondes 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;
```

# Az-Delivery

```
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");
  }
  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);
}
```

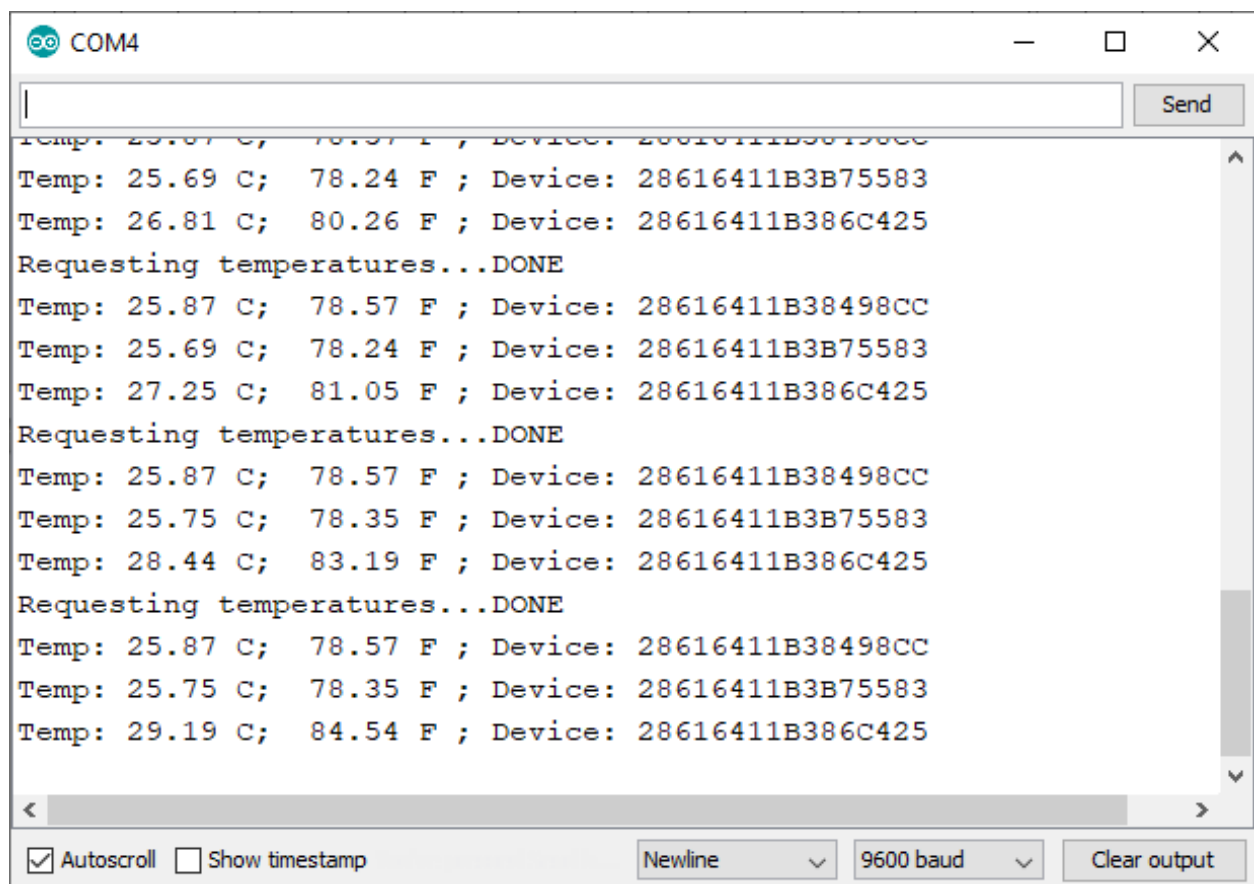
# Az-Delivery

```
// one tab
    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();
}
```

# Az-Delivery

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

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



# Az-Delivery

The following is an explanation of the sketch.

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 Uno, except the ones used in Serial Interface, `D0` and `D1` (it is a recommendation, you can use them, but you have to make sure that these pins are disconnected when you are uploading sketches).

We used `TEMPERATURE_PRECISION` variable to set precision for DS18B20 sondes. Number saved in this variable is a digital conversion number in bits and it can be in range from 9 to 12, any other number will result in 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 sonde addresses, and in our example we created three.

We defined and created `oneWire` object, used for One-Wire interface:

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

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

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



# Az-Delivery

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

```
sensors.begin()
```

With that line of the code *sensors* object detects all sondes connected on the One-Wire interface. It also detects all addresses of sondes.

Now we can check if sondes are working properly, by using the following lines of code for every sonde 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 sonde.

To set-up analog to digital conversion precision of the specific sonde 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 sonde, you can use the following line of code:

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

The function returns hexadecimal value, and to convert it to 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 sondes, by using the following line of code:

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

# Az-Delivery

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

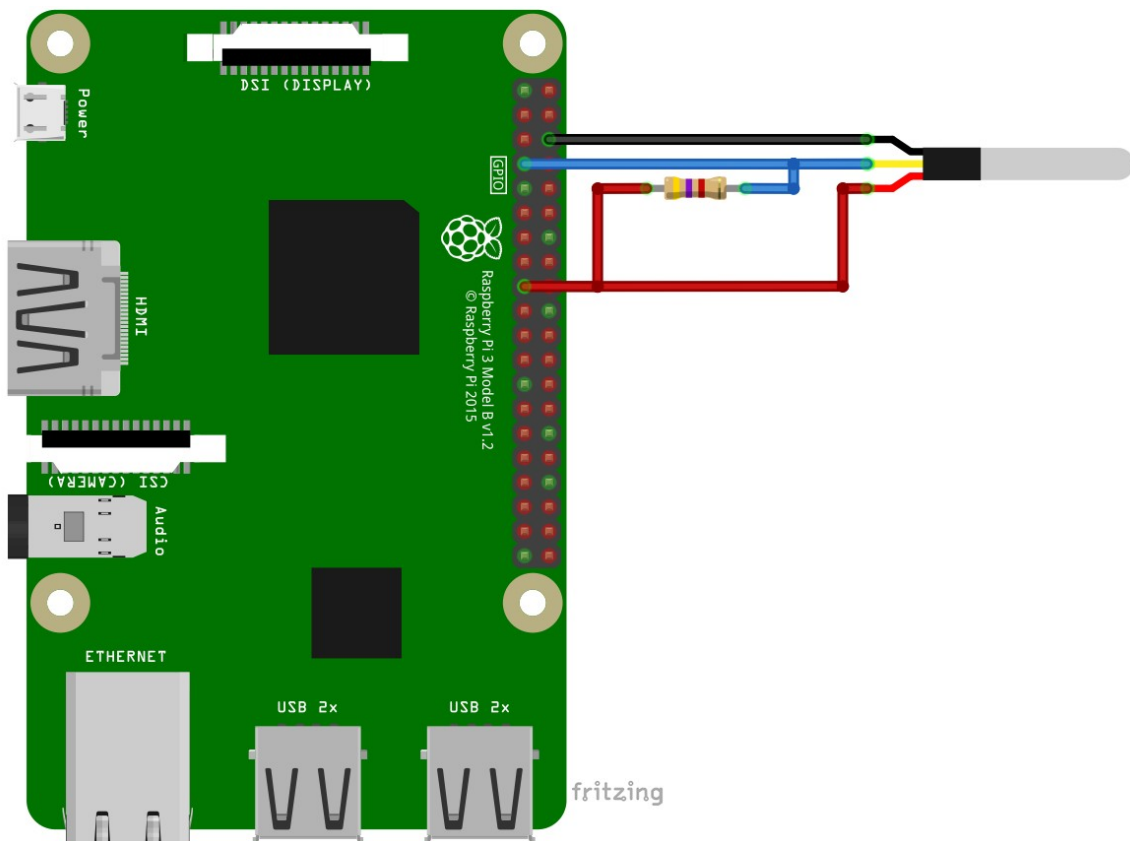
```
float tempC = sensors.getTempC(deviceAddress);
```

where we pass *deviceAddress* argument to the function in order to read temperature data from specific sonde. 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 sonde with Raspberry Pi

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



DS18B20 pin	>	Raspberry Pi pin
GND	>	GND [pin 4]
DATA	>	GPIO4 [pin 7]
VCC	>	3V3 [pin 17]

**Black wire**

**Blue wire**

**Red wire**

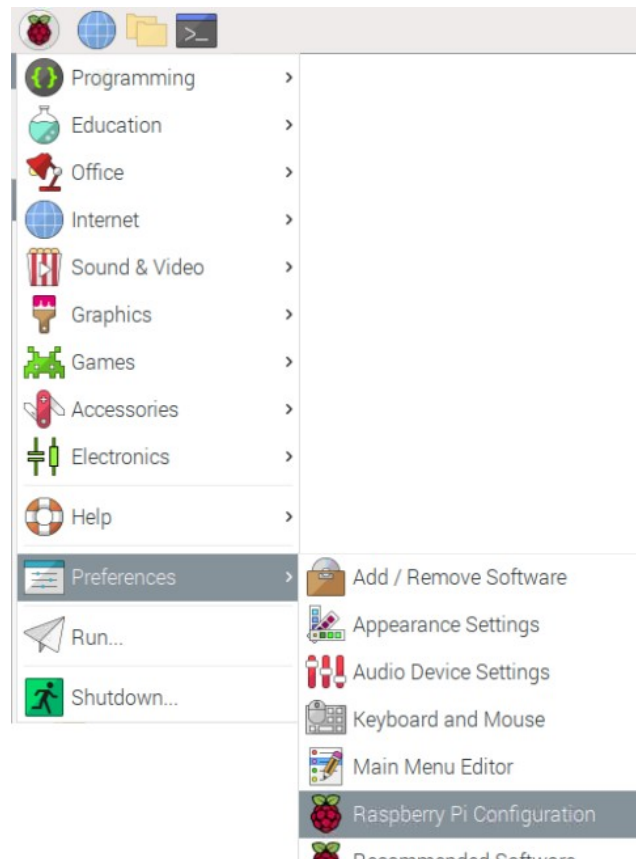
**NOTE:** Pull up 4.7kΩ resistor is connected between *OUT PIN* and 3V3 pin.

# Az-Delivery

## Enabling One-Wire interface

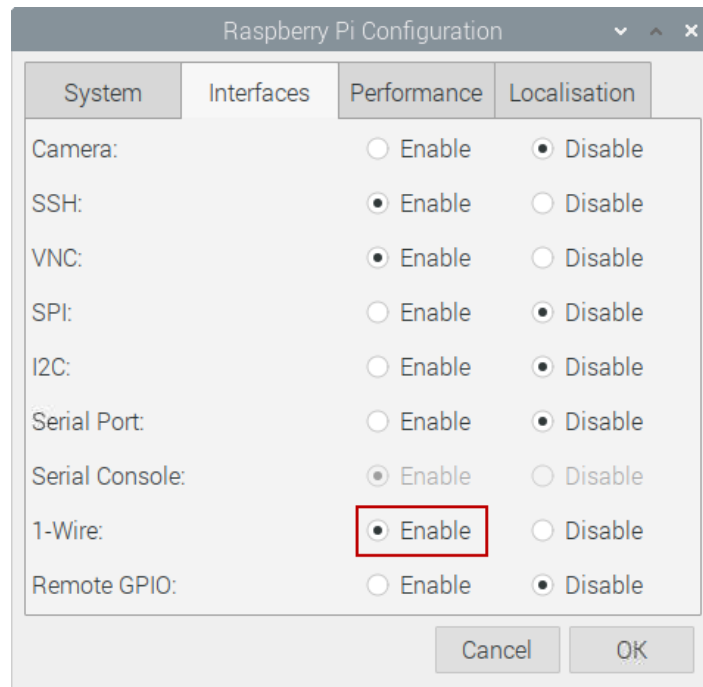
Before we can use the DS18B20 sonde with the Raspberry Pi, first we have to enable the One-Wire interface in Raspbian. By default, the hardware One-Wire interface is on pin GPIO4 (pin 7), but we first have to enable it. In order to enable One-Wire interface, open *All Apps* and go to:

*Preferences > Raspberry Pi Configuration*, as shown below:

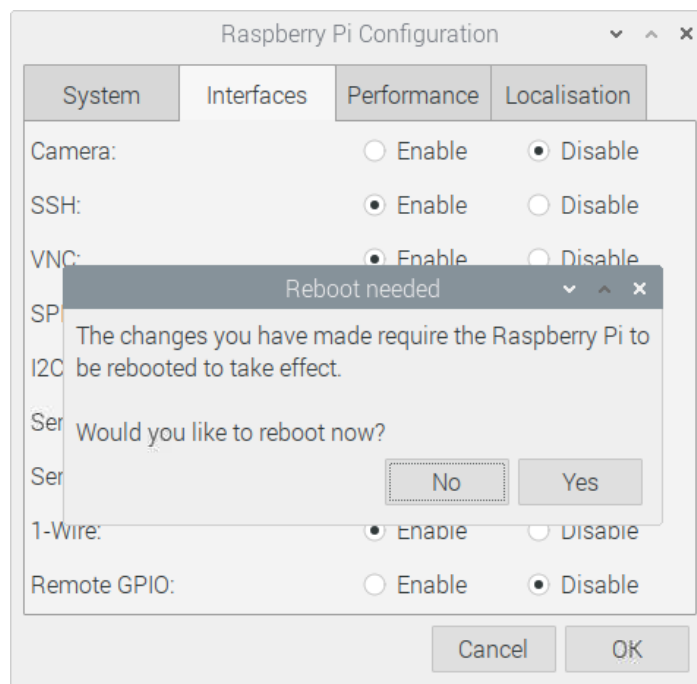


# Az-Delivery

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.



# Az-Delivery

When the 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:

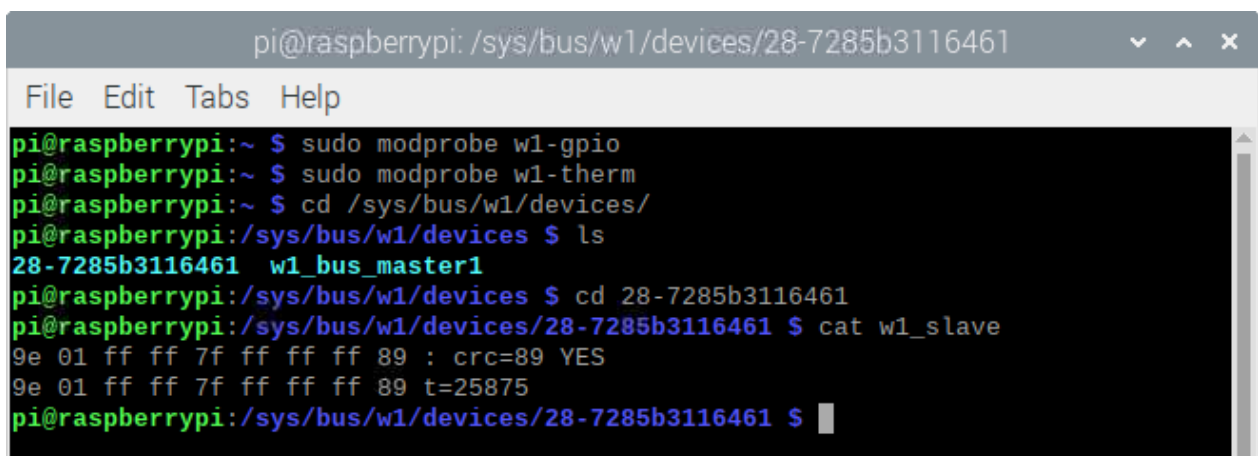
```
28-7285b3116461 and w1_bus_master1
```

the first number (*28-7285b3116461*) will be different for you, because this is the serial address of the specific sonde, and each sonde 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 last page
```

```
cat w1_slave
```

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

A screenshot of a terminal window titled 'pi@raspberrypi: /sys/bus/w1/devices/28-7285b3116461'. The terminal shows the following sequence of commands and outputs:

```
pi@raspberrypi:~ $ sudo modprobe w1-gpio
pi@raspberrypi:~ $ sudo modprobe w1-therm
pi@raspberrypi:~ $ cd /sys/bus/w1/devices/
pi@raspberrypi:/sys/bus/w1/devices $ ls
28-7285b3116461  w1_bus_master1
pi@raspberrypi:/sys/bus/w1/devices $ cd 28-7285b3116461
pi@raspberrypi:/sys/bus/w1/devices/28-7285b3116461 $ cat w1_slave
9e 01 ff ff 7f ff ff ff 89 : crc=89 YES
9e 01 ff ff 7f ff ff ff 89 t=25875
pi@raspberrypi:/sys/bus/w1/devices/28-7285b3116461 $
```

**t=25875** - this is the temperature data in °C (Celsius) = 25,875°C.



## Enabling multiple One-Wire interfaces

To enable the One-Wire interface, without 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 instead, including creating multiple One-Wire interfaces to be used at the same time:

```
sudo dtoverlay w1-gpio gpiopin=4 pullup=0 # header pin 7  
sudo dtoverlay w1-gpio gpiopin=17 pullup=0 # header pin 11  
sudo dtoverlay w1-gpio gpiopin=27 pullup=0 # header 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  $4.7k\Omega$  pull-up resistor connected between the GPIO pin and a 3.3V supply.



## Python script for reading multiple DS18B20 sondes

We choose to split the code into two scripts, because of 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
```



# Az-Delivery

```
# (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
```

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

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

# Az-Delivery

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

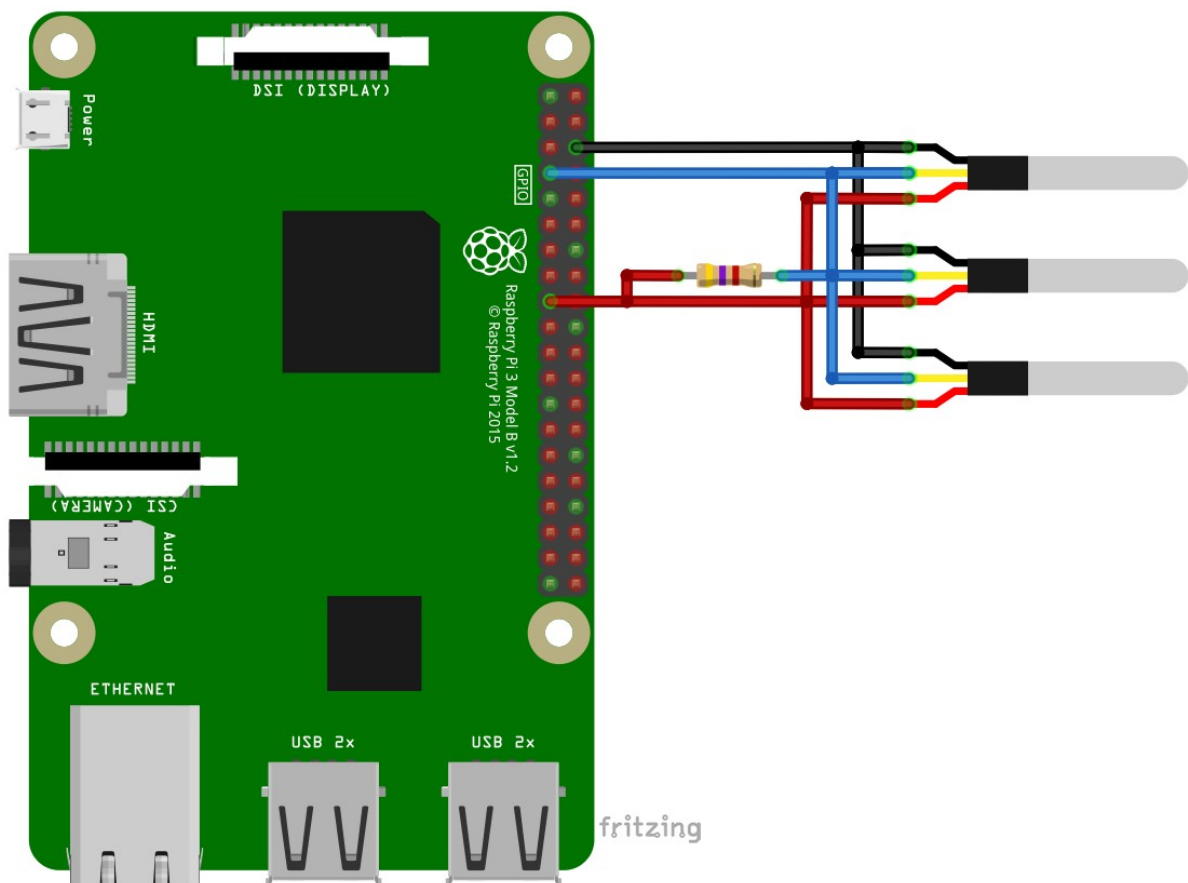
        time.sleep(1)

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

# Az-Delivery

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

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

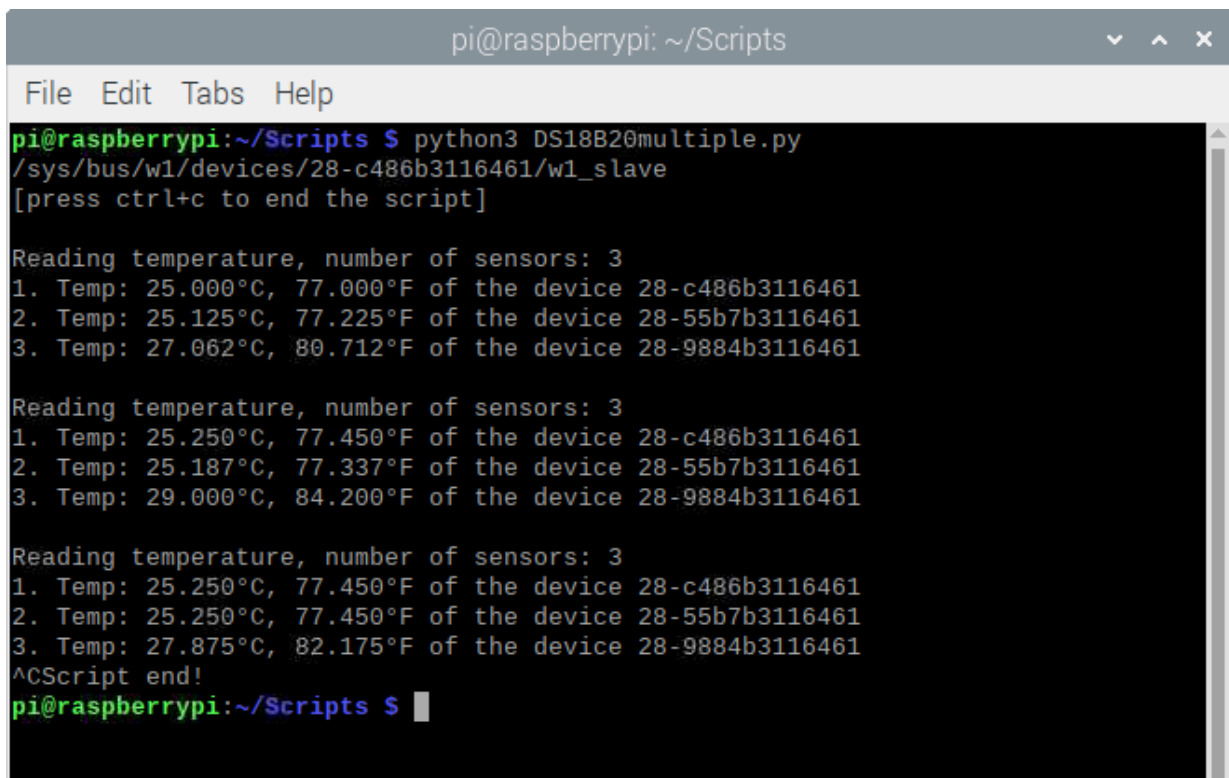


# Az-Delivery

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

```
python3 DS18B20multiple.py
```

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



```
pi@raspberrypi: ~/Scripts
File Edit Tabs Help
pi@raspberrypi:~/Scripts $ python3 DS18B20multiple.py
/sys/bus/w1/devices/28-c486b3116461/w1_slave
[press ctrl+c to end the script]

Reading temperature, number of sensors: 3
1. Temp: 25.000°C, 77.000°F of the device 28-c486b3116461
2. Temp: 25.125°C, 77.225°F of the device 28-55b7b3116461
3. Temp: 27.062°C, 80.712°F of the device 28-9884b3116461

Reading temperature, number of sensors: 3
1. Temp: 25.250°C, 77.450°F of the device 28-c486b3116461
2. Temp: 25.187°C, 77.337°F of the device 28-55b7b3116461
3. Temp: 29.000°C, 84.200°F of the device 28-9884b3116461

Reading temperature, number of sensors: 3
1. Temp: 25.250°C, 77.450°F of the device 28-c486b3116461
2. Temp: 25.250°C, 77.450°F of the device 28-55b7b3116461
3. Temp: 27.875°C, 82.175°F of the device 28-9884b3116461
^CScript end!
pi@raspberrypi:~/Scripts $
```

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

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

**You've 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 products for Arduino and Raspberry Pi, 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>