Welcome!

Thank you for purchasing our *AZ-Delivery KY-039 Heartbeat Sensor Module.* On the following pages, you will be introduced to how to use and set up this handy device.

Have fun!

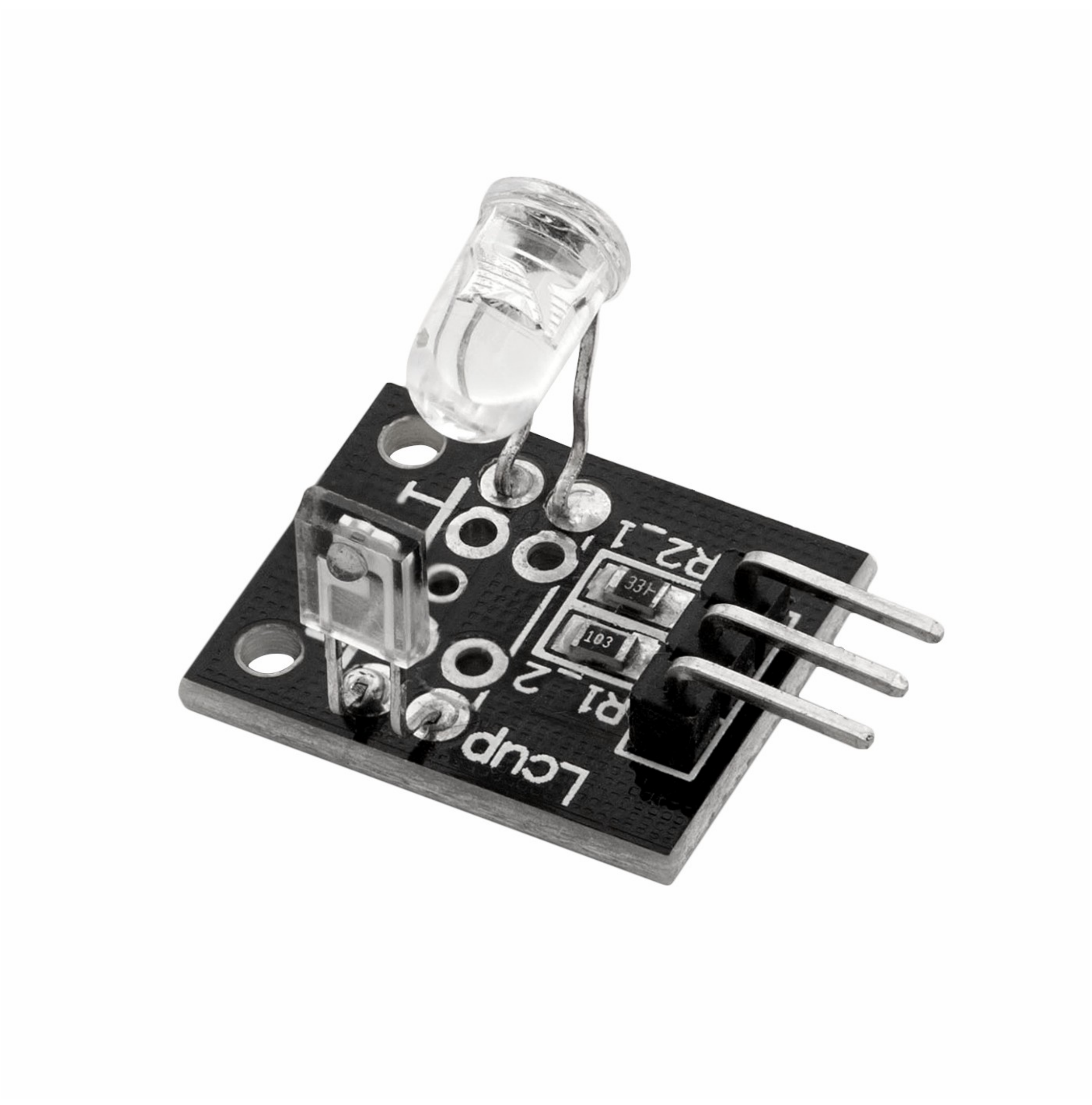


Table of Contents

Introduction

The KY-039 heartbeat sensor module has an IR LED, a photosensitive diode and a resistor on-board. The output of the module is an analog voltage that represents how much the infrared light a photosensitive diode receives. The higher the value, the stronger the intensity of the infrared light.

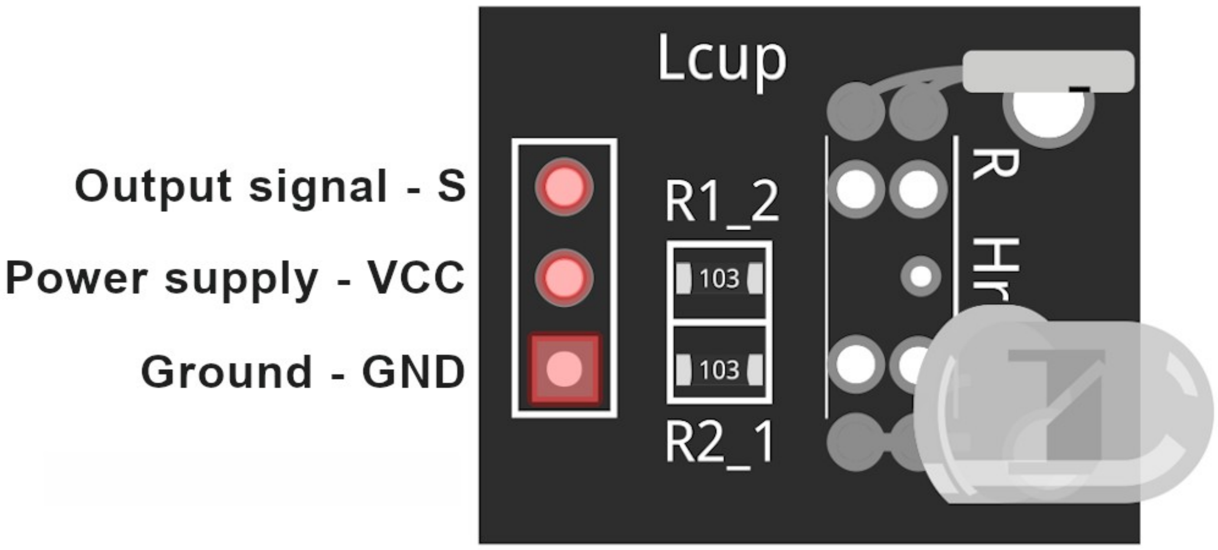
Place a finger between the IR LED and the photosensitive diode of the module. Heartbeats dilate the blood vessels in a finger, which filters the infrared light at different levels. This creates a pulsating signal which can be seen on the image of Serial Plotter (which is shown later in the text).

Specifications

»	Operating voltage range:	from 3.3V to 5V DC
»	Operating temperature range:	from -40°C to 85°C
»	Dimensions:	19 x 15mm [0.73 x 0.6in]

The pinout

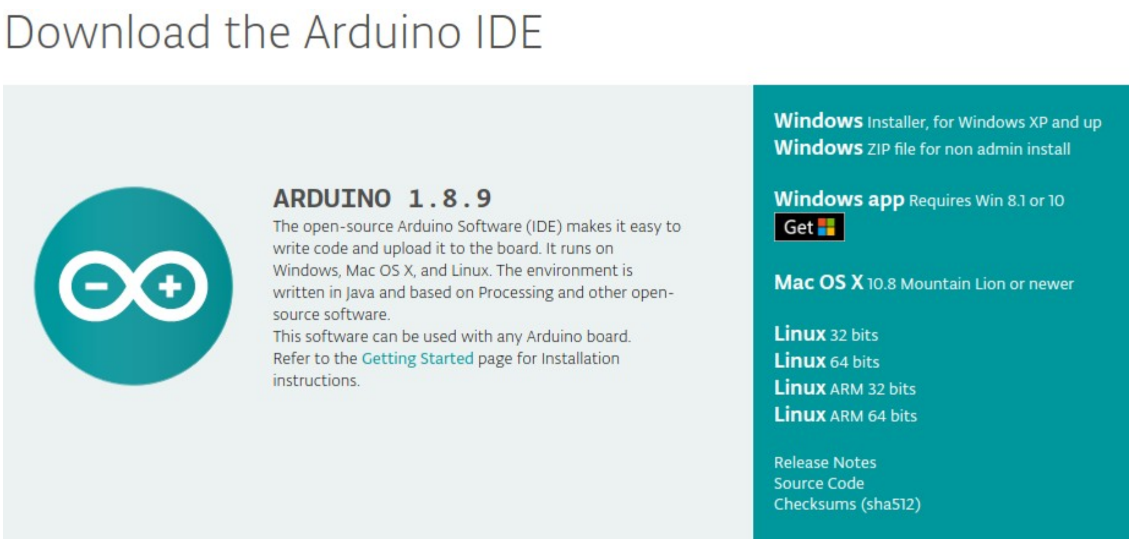
The KY-039 heartbeat sensor module has three pins. The pinout diagram is shown on the following image:



AZ-Delivery

How to set-up Arduino IDE

If the Arduino IDE is not installed, follow the *link* and download the installation file for the operating system of choice.



Download the Arduino IDE

ARDUINO 1.8.9

The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open-source software.
This software can be used with any Arduino board.
Refer to the Getting Started page for Installation instructions.

Windows Installer, for Windows XP and up
Windows ZIP file for non admin install

Windows app Requires Win 8.1 or 10
Get

Mac OS X 10.8 Mountain Lion or newer

Linux 32 bits
Linux 64 bits
Linux ARM 32 bits
Linux ARM 64 bits

Release Notes
Source Code
Checksums (sha512)

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

For *Linux* users, download a file with the extension *.tar.xz*, which has to be extracted. When it is extracted, go to the extracted directory and open the terminal in that directory. Two *.sh* scripts have to be executed, the first called *arduino-linux-setup.sh* and the second called *install.sh*.

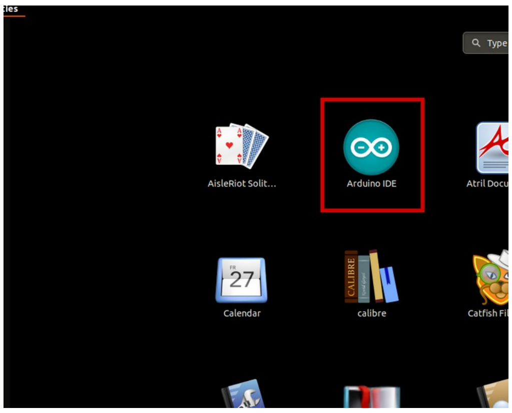
To run the first script in the terminal, open the terminal in the extracted directory and run the following command:

sh arduino-linux-setup.sh user_name

user_name - is the name of a superuser in the Linux operating system. A password for the superuser has to be entered when the command is started. Wait for a few minutes for the script to complete everything.

The second script called *install.sh* script has to be used after installation of the first script. Run the following command in the terminal (extracted directory): **sh install.sh**

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

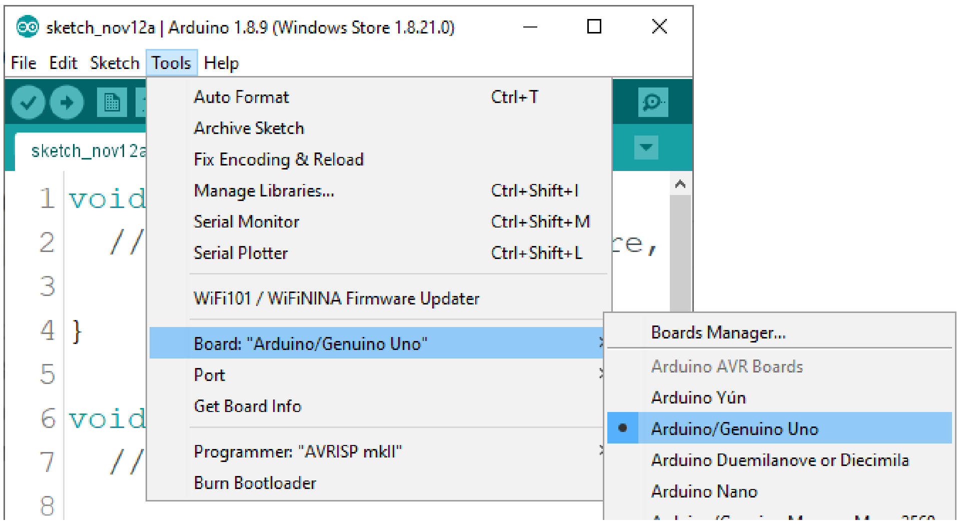


Almost all operating systems come with a text editor preinstalled (for example, *Windows* comes with *Notepad*, *Linux Ubuntu* comes with *Gedit*, *Linux Raspbian* comes with *Leafpad*, etc.). All of these text editors are perfectly fine for the purpose of the eBook.

Next thing is to check if your PC can detect an 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 it can be seen on the following image:



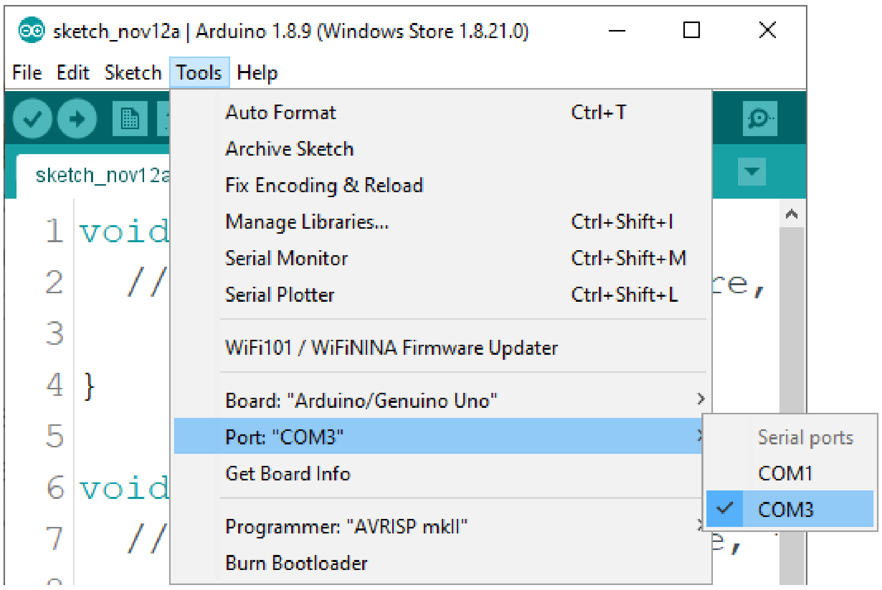
sketch_nov12a | Arduino 1.8.9 (Windows Store 1.8.21.0)
File Edit Sketch Tools Help
sketch_nov12a
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Board: "Arduino/Genuino Uno"
Port
Get Board Info
Programmer: "AVRISP mkII"
Burn Bootloader
Boards Manager...
Arduino AVR Boards
Arduino Yún
Arduino/Genuino Uno
Arduino Duemilanove or Diecimila
Arduino Nano

The port to which the Arduino board is connected has to be selected. Go to:

Tools > Port > {port name goes here}

and when the Arduino board is connected to the USB port, the port name can be seen in the drop-down menu on the previous image.

If the Arduino IDE is used on Windows, port names are as follows:


sketch_nov12a | Arduino 1.8.9 (Windows Store 1.8.21.0)
File Edit Sketch Tools Help
sketch_nov12a
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Board: "Arduino/Genuino Uno"
Port: "COM3"
Get Board Info
Programmer: "AVRISP mkII"
Burn Bootloader
Serial ports
COM1
✓ COM3

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

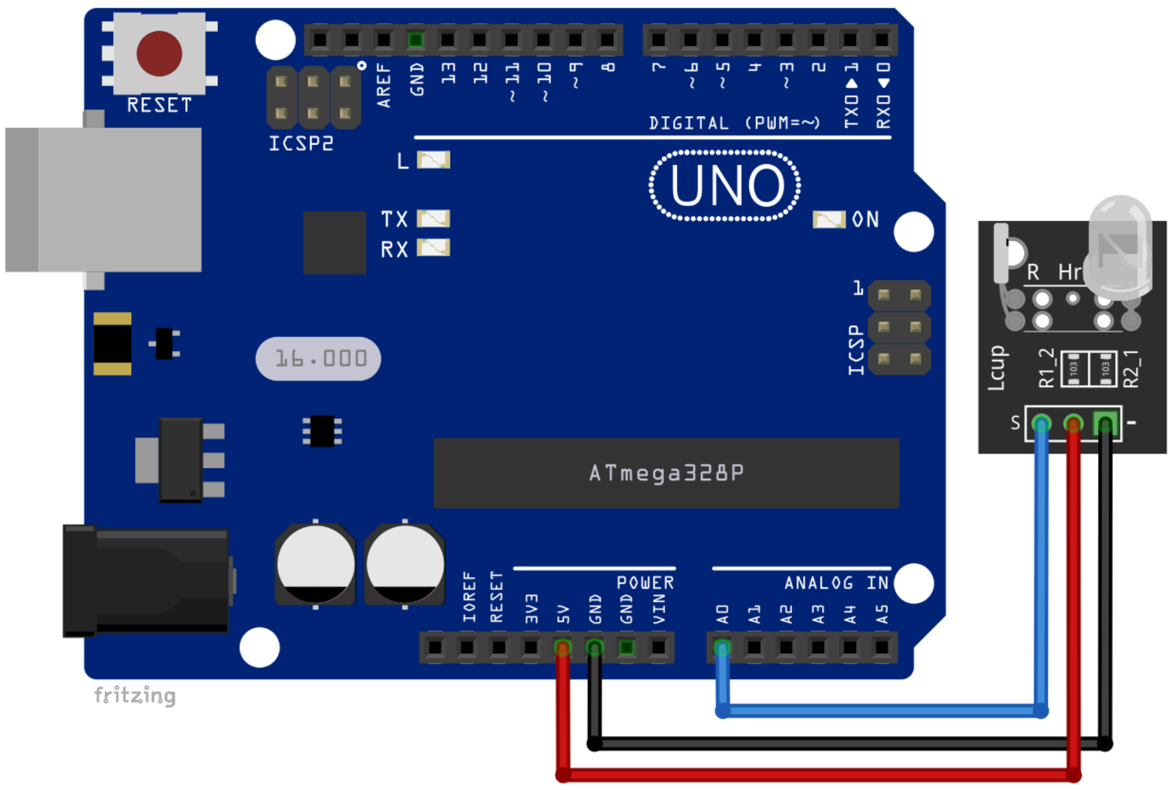
How to set-up the Raspberry Pi and Python

For the Raspberry Pi, first the operating system has to be installed, then everything has to be set-up so that it can be used in the *Headless* mode. The *Headless* mode enables remote connection to the Raspberry Pi, without the need for a *PC* screen Monitor, mouse or keyboard. The only things that are used in this mode are the Raspberry Pi itself, power supply and internet connection. All of this is explained minutely in the free eBook:
Raspberry Pi Quick Startup Guide

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

Connecting the module with Uno

Connect the KY-039 module with the Uno as shown on the following connection diagram:

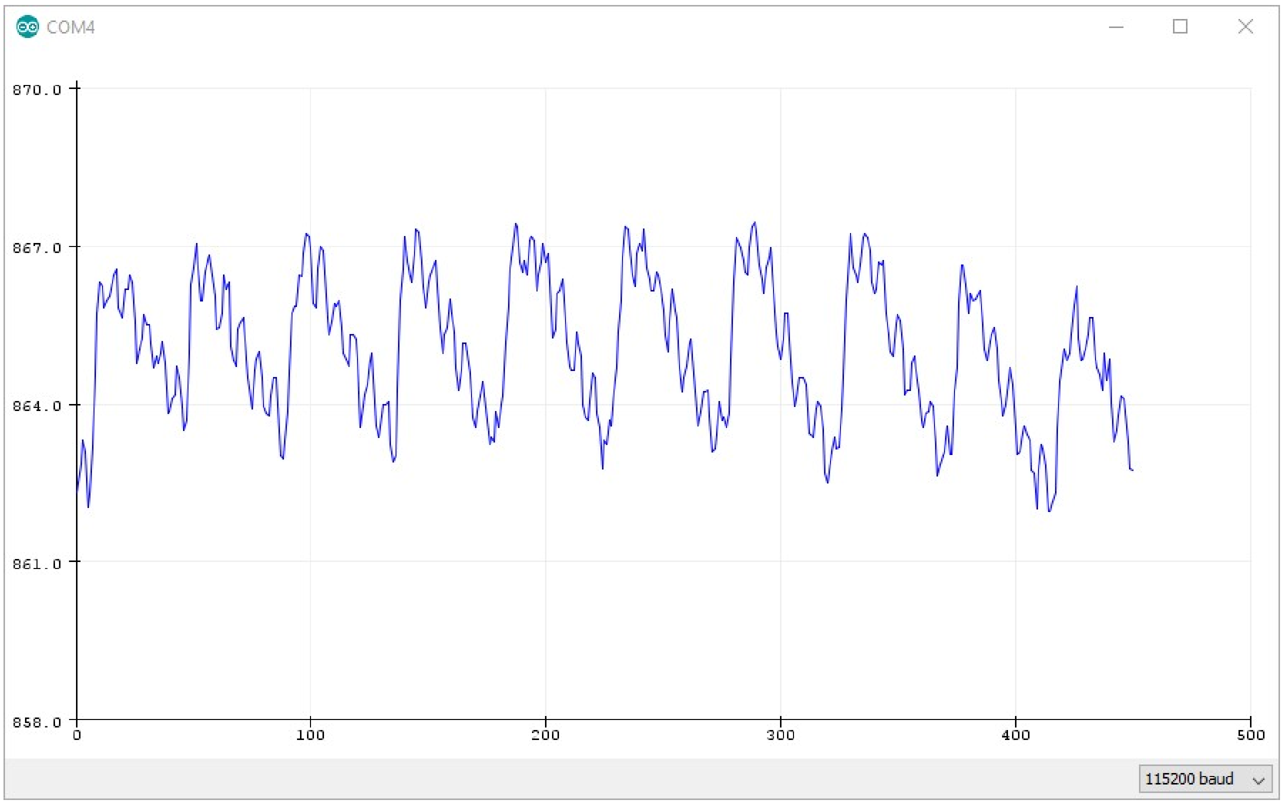


KY-039 pin	**>**	**Uno pin**	
S	>	A0	**Blue wire**
- (GND)	>	GND	**Black wire**
Middle pin (VCC)	>	5V	**Red wire**

Sketch example

```
#define ANALOG_PIN 0

float average = 0.0;

void setup() {
  Serial.begin(9600);
}

void loop() {
  for(uint8_t i = 0; i <= 19; i++) {
    average = average + analogRead(0);
    delay(1);
  }
  average = average / 20.0;

  Serial.println(average);
  average = 0.0;
}
```

Upload the sketch to the Uno and open the Serial Plotter (*`Tools > Serial Plotter`*). The result should look like the output on the following image:



On the image of a Serial Plotter, you can easily see the heartbeats (larger peaks). All that has to be done is to smooth the signal and then to count the heartbeats or peaks on the diagram curve. This is done in the Python script of the *Connecting the module with Raspberry Pi* chapter.

NOTE: To get these values, protect the sensor from any other light source. The nontransparent box can be created where the sensor can be protected from the external light source. If the sensor is not hidden from other light sources, it reads the infrared light from other light sources which distort the output signal. If that happens the signal has to be cleaned from that distortions in order for measurements to be useful.

Connecting the module with Raspberry Pi

Because the Raspberry Pi does not have Analog to Digital Converter (ADC), but for purpose of using the KY-013 module with the Raspberry Pi, the Raspberry Pi has to be able to read analog voltages. The Uno can be used for the purpose. In order to do so, the Uno is used on the *Linux Raspbian* operating system. Uno can read analog voltages, and it can use *Serial Interface* via *USB* port to send data to the Raspberry Pi.

First, the Arduino IDE has to be downloaded and installed on the Raspbian. Second, the firmware for Uno has to be downloaded and uploaded to the Uno, and lastly, the library for Python has to be downloaded and installed.

To do this, power on the Raspberry Pi and connect it to the internet. Start the *RealVNC* app on the remote computer and connect the app to the Raspberry Pi (like explained in the eBook for the Raspberry Pi).

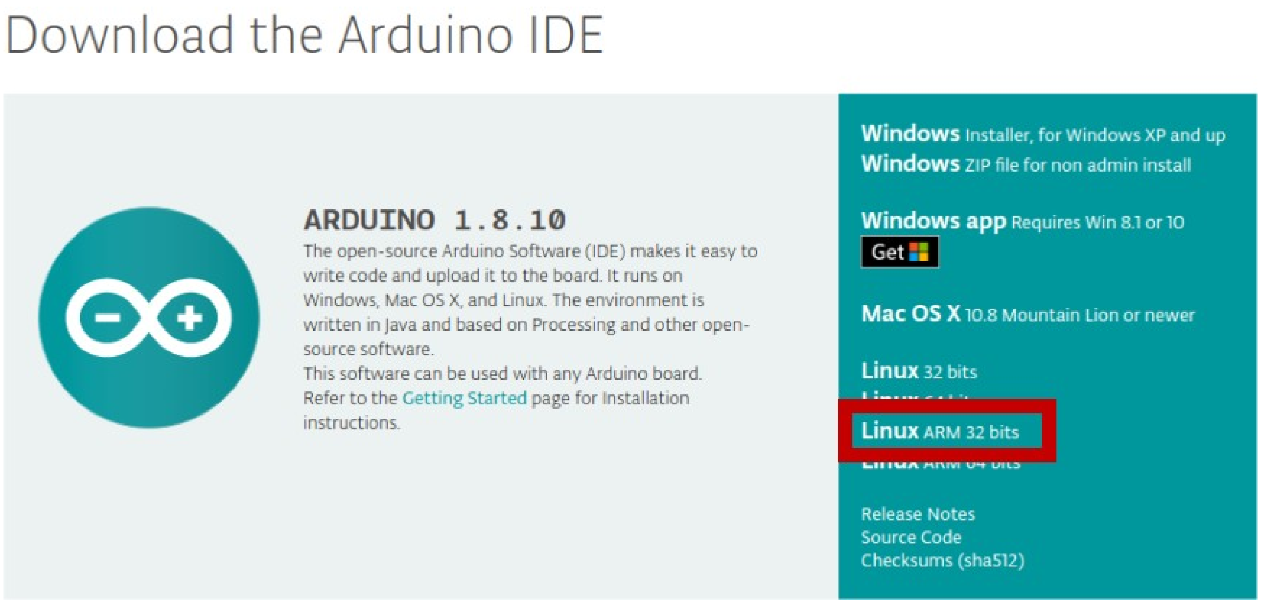
First thing that has to be done when the Raspberry Pi is booted up is to update the Raspbian; open the terminal and run the following command:

```
sudo apt-get update && sudo apt-get upgrade -y
```

And wait for the command to finish its job.

Now, the Raspbian operating system is up to date.

To download and install the Arduino IDE, go on the Arduino *site* and download the `tar.xz` file of Arduino IDE for `Linux ARM 32 bits` as shown on the following image:

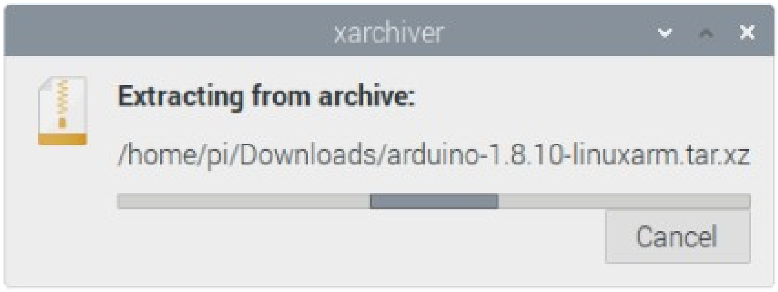

Download the Arduino IDE

ARDUINO 1.8.10

The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open-source software.
This software can be used with any Arduino board.
Refer to the Getting Started page for Installation instructions.

Windows Installer, for Windows XP and up
Windows ZIP file for non admin install
Windows app Requires Win 8.1 or 10
Get
Mac OS X 10.8 Mountain Lion or newer
Linux 32 bits
Linux ARM 32 bits
Release Notes
Source Code
Checksums (sha512)

Then, the `tar.xz` file has to be extracted. Open the `File explorer` in directory where the `tar.xz` file is downloaded, right click on it, and run the option `Extract Here`. Wait for a few minutes for the extracting process to complete itself.


xarchiver

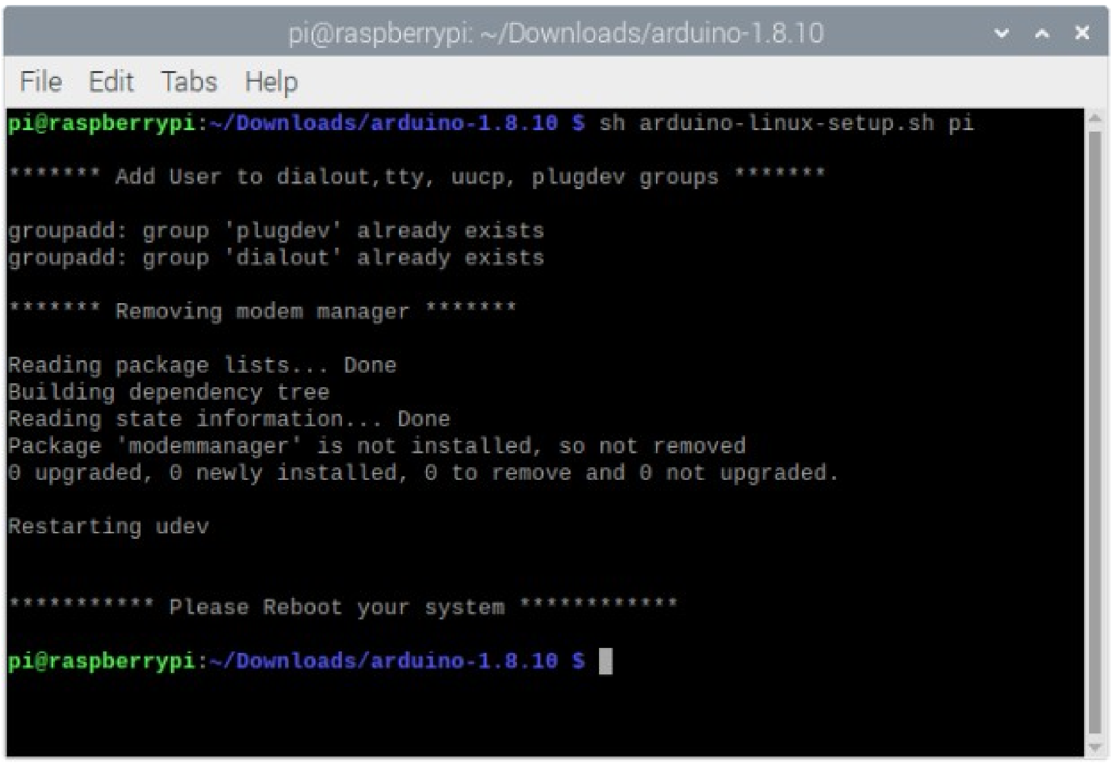
Extracting from archive:

/home/pi/Downloads/arduino-1.8.10-linuxarm.tar.xz

Open terminal in extracted folder and run the following command:

`sh arduino-linux-setup.sh pi`

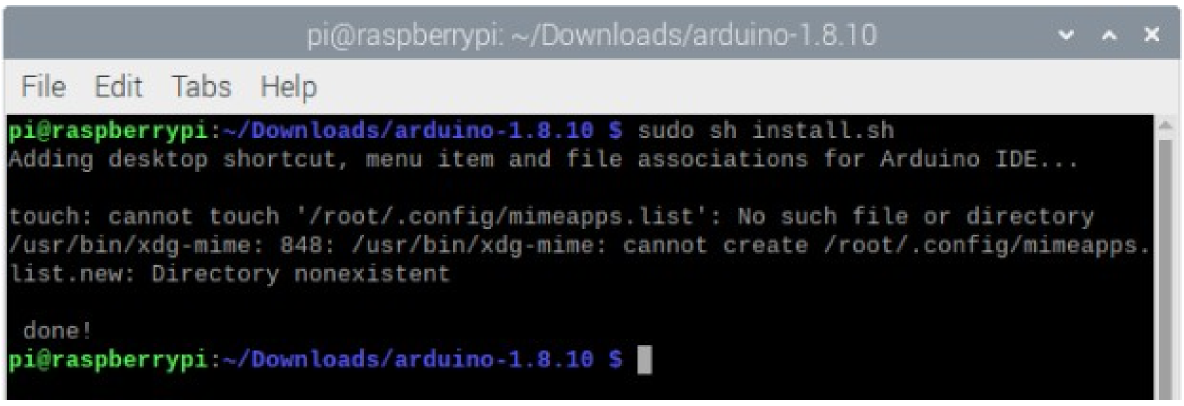
where *`pi`* is the name of the superuser in Raspbian.



```
pi@raspberrypi:~/Downloads/arduino-1.8.10 $ sh arduino-linux-setup.sh pi

******* Add User to dialout,tty, uucp, plugdev groups *******

groupadd: group 'plugdev' already exists
groupadd: group 'dialout' already exists

******* Removing modem manager *******

Reading package lists... Done
Building dependency tree
Reading state information... Done
Package 'modemmanager' is not installed, so not removed
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.

Restarting udev


*********** Please Reboot your system ************

pi@raspberrypi:~/Downloads/arduino-1.8.10 $
```

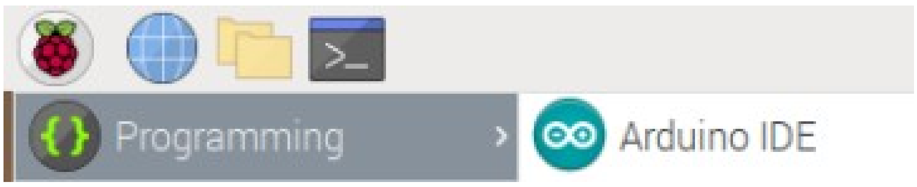
After this, to install the Arduino IDE, run the following command:

`sudo sh install.sh`



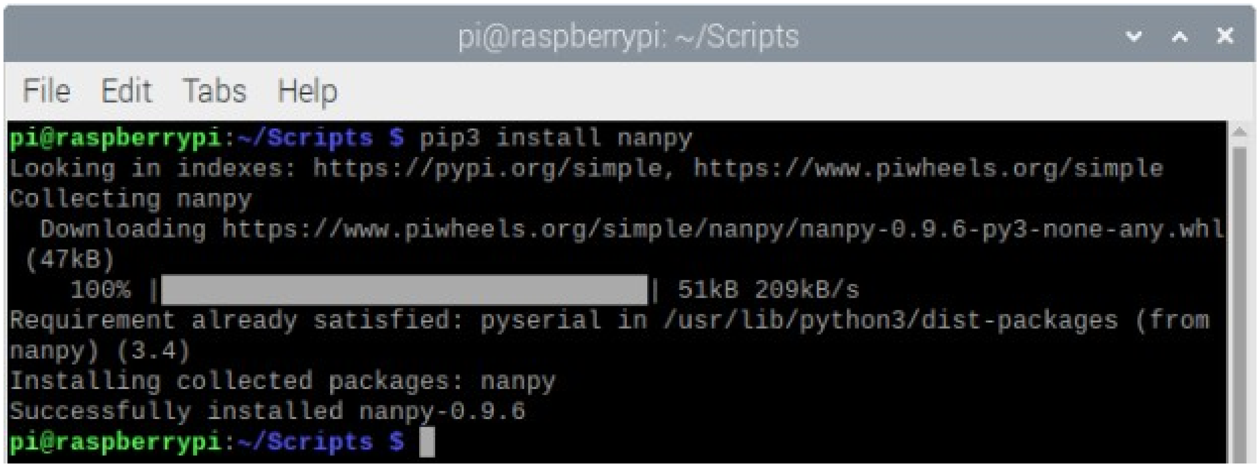
```
pi@raspberrypi:~/Downloads/arduino-1.8.10 $ sudo sh install.sh
Adding desktop shortcut, menu item and file associations for Arduino IDE...

touch: cannot touch '/root/.config/mimeapps.list': No such file or directory
/usr/bin/xdg-mime: 848: /usr/bin/xdg-mime: cannot create /root/.config/mimeapps.
list.new: Directory nonexistent

 done!
pi@raspberrypi:~/Downloads/arduino-1.8.10 $
```

The Arduino IDE is now installed. To run Arduino IDE, open the app:
`Applications Menu > Programming > Arduino IDE`



Before next steps, first the *`pip3`* and *`git`* apps have to be installed; run the following command:
`sudo apt install python3-pip git -y`

The library for Python is called *nanpy*. To install it, open terminal and run the following command: **`pip3 install nanpy`**



```
pi@raspberrypi:~/Scripts $ pip3 install nanpy
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting nanpy
  Downloading https://www.piwheels.org/simple/nanpy/nanpy-0.9.6-py3-none-any.whl
 (47kB)
    100% |████████████████████████████████| 51kB 209kB/s
Requirement already satisfied: pyserial in /usr/lib/python3/dist-packages (from
nanpy) (3.4)
Installing collected packages: nanpy
Successfully installed nanpy-0.9.6
pi@raspberrypi:~/Scripts $ 
```

After installing the *nanpy* library, download an Arduino firmware by running the following command:

`git clone https://github.com/nanpy/nanpy-firmware.git`

Change directory to *`nanpy-firmware`* by running the following command:

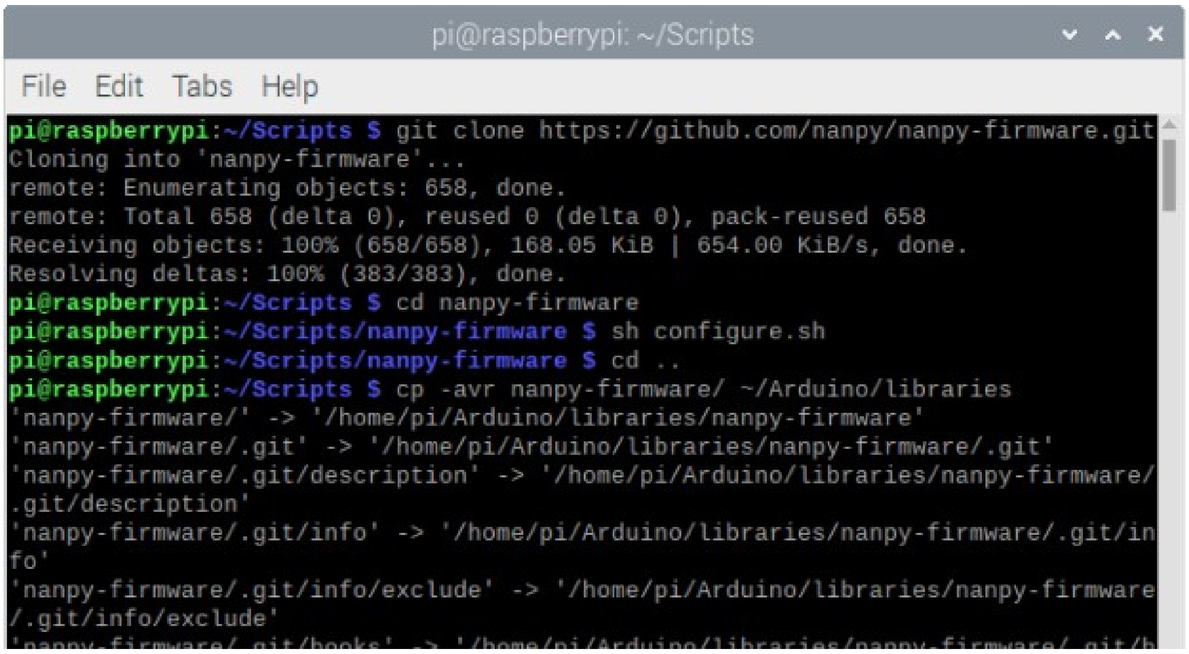
`cd nanpy-firmware`

And run the following command:

`sh configure.sh`

Next, copy the *`nanpy-firmware`* directory into *`Arduino/libraries`* directory. To do so, run the following command:

`cp -avr nanpy-firmware/ ~/Arduino/libraries`

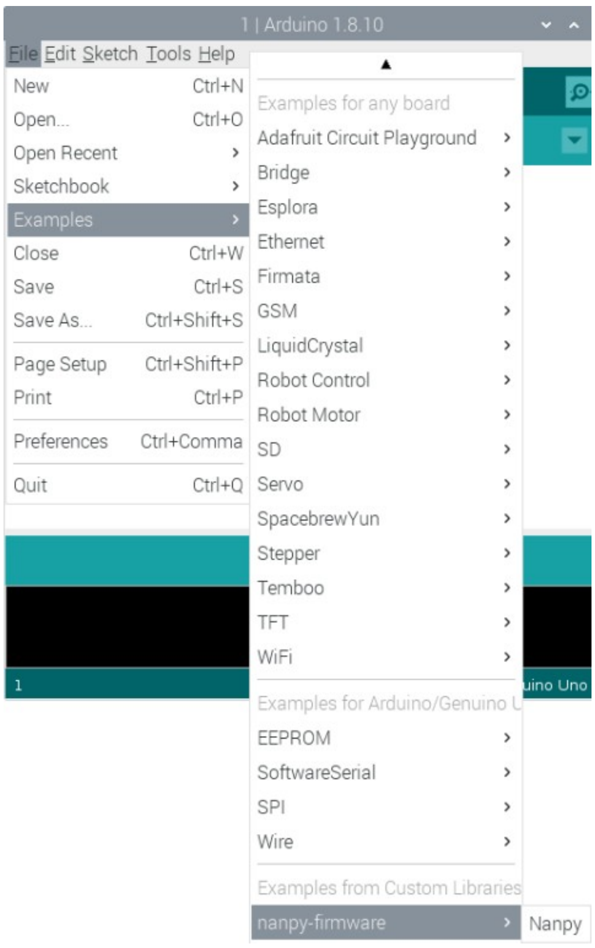


```
pi@raspberrypi:~/Scripts $ git clone https://github.com/nanpy/nanpy-firmware.git
Cloning into 'nanpy-firmware'...
remote: Enumerating objects: 658, done.
remote: Total 658 (delta 0), reused 0 (delta 0), pack-reused 658
Receiving objects: 100% (658/658), 168.05 KiB | 654.00 KiB/s, done.
Resolving deltas: 100% (383/383), done.
pi@raspberrypi:~/Scripts $ cd nanpy-firmware
pi@raspberrypi:~/Scripts/nanpy-firmware $ sh configure.sh
pi@raspberrypi:~/Scripts/nanpy-firmware $ cd ..
pi@raspberrypi:~/Scripts $ cp -avr nanpy-firmware/ ~/Arduino/libraries
'nanpy-firmware/' -> '/home/pi/Arduino/libraries/nanpy-firmware'
'nanpy-firmware/.git' -> '/home/pi/Arduino/libraries/nanpy-firmware/.git'
'nanpy-firmware/.git/description' -> '/home/pi/Arduino/libraries/nanpy-firmware/
.git/description'
'nanpy-firmware/.git/info' -> '/home/pi/Arduino/libraries/nanpy-firmware/.git/in
fo'
'nanpy-firmware/.git/info/exclude' -> '/home/pi/Arduino/libraries/nanpy-firmware
/.git/info/exclude'
```

The *`nanpy-firmware`* is now installed and ready to be used.

Connect the Uno via the *USB* cable to the Raspberry Pi and then open the Arduino IDE in the Raspbian operating system. Check if the Arduino IDE can detect the *USB* port to which the Uno is connected:
`Tools > Port > dev/ttyUSB0`

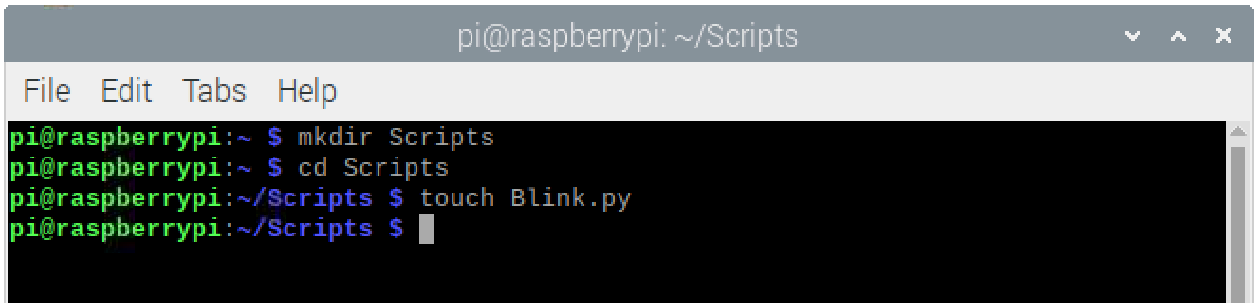
Then go to: *`Tools > Board > {board name}`*
and select *`Arduino Uno`* board.
Then, to open a sketch for the *`nanpy-firmware`*, go to:
`File > Examples > nanpy-firmware > Nanpy`



Upload the sketch to the Uno. To test if eveyrhing works properly, the simple *Blink* script has to be created, where the on-board LED of the Uno is blinked.

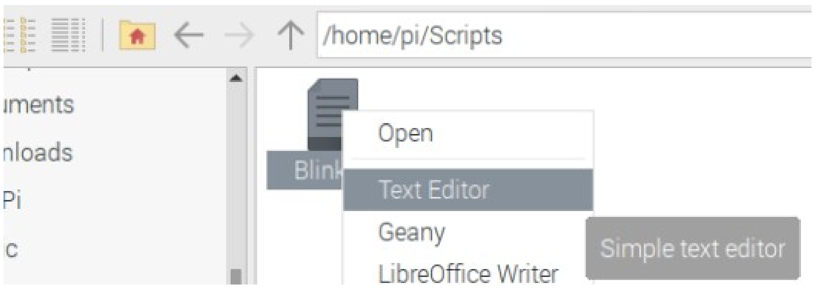
Open terminal, create the *Scripts* directory and the *Blink.py* script. To do so, run the following commands, one by one:

mkdir Scrpits	- create to the *Scripts* directory
cd /Scrpits	- change to the *Scripts* directory
touch Blink.py	- create new file called *Blink.py*



```
pi@raspberrypi:~ $ mkdir Scripts
pi@raspberrypi:~ $ cd Scripts
pi@raspberrypi:~/Scripts $ touch Blink.py
pi@raspberrypi:~/Scripts $ 
```

Open *File Explorer*, navigate to the *Scrpits* directory and open *Blink.py* script in the default text editor:



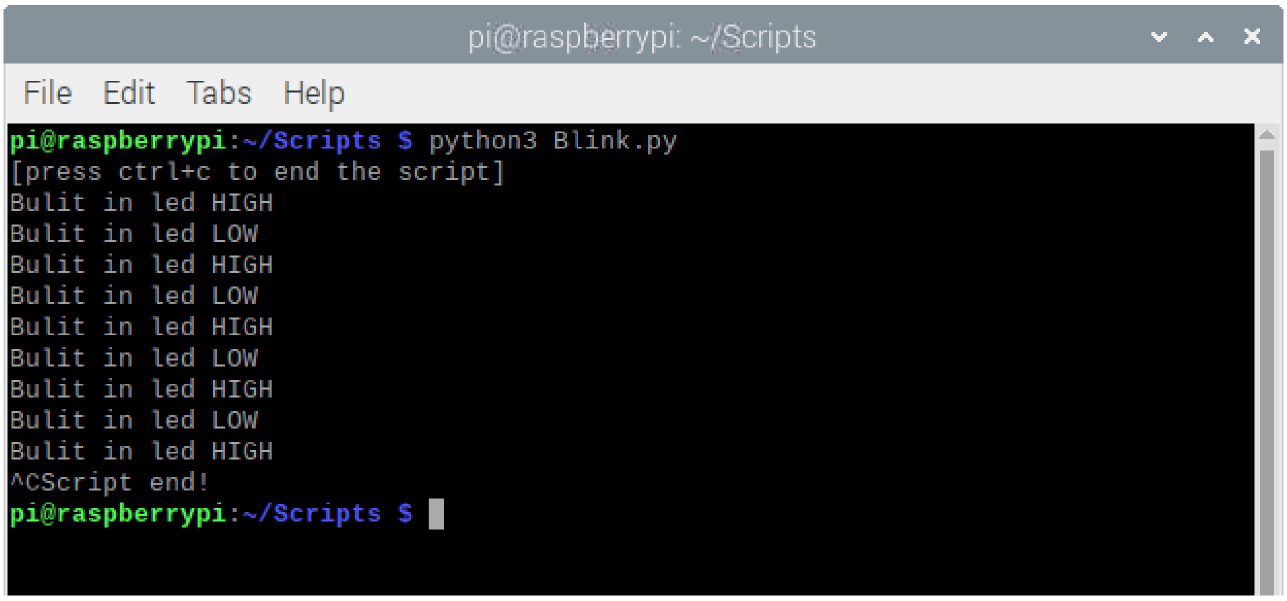
In the *Blink.py* script write the following lines of code:

```
from nanpy import (ArduinoApi, SerialManager)
from time import sleep

ledPin = 13
try:
    connection1 = SerialManager()
    a = ArduinoApi(connection=connection1)
except:
    print('Failed to connect to the Arduino')

print('[Press CTRL + C to end the script!]')
a.pinMode(ledPin, a.OUTPUT) # Setup Arduino

try:
    while True:
        a.digitalWrite(ledPin, a.HIGH)
        print('Bulit in led HIGH')
        sleep(1)
        a.digitalWrite(ledPin, a.LOW)
        print('Bulit in led LOW')
        sleep(1)

except KeyboardInterrupt:
    print('\nScript end!')
    a.digitalWrite(ledPin, a.LOW)
```

Save the script. To run the script, open terminal in the directory where the script is saved and run the following command:

`python3 Blink.py`

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



```
pi@raspberrypi:~/Scripts $ python3 Blink.py
[press ctrl+c to end the script]
Bulit in led HIGH
Bulit in led LOW
Bulit in led HIGH
Bulit in led LOW
Bulit in led HIGH
Bulit in led LOW
Bulit in led HIGH
Bulit in led LOW
Bulit in led HIGH
^CScript end!
pi@raspberrypi:~/Scripts $
```

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

The LED connected to the digital pin *13* of the Uno should start blinking every second.

The script starts with importing two libraries, the *nanpy* library functions, and the *time*.

Then, the variable called *ledPin* is created and initialized with number *13*. The number *13* represents the number of the digital pin on which LED is connected (on-board LED of the UNO).

After that, the *try-except* block of code is used to try and connect to the Uno. If the connection is not successful, the message:
Failed to connect to the Arduino
is displayed in the terminal.

If the connection is successful, the communication object called "*a*" is created and initialized. The object "*a*" represents the Uno board. Any function used in the Arduino IDE can be used with the "*a*" object, as it can been seen in the code.

With the following line of code, the pin mode of the digital pin *13* is set-up:
`a.pinMode(ledPin, a.OUTPUT)`

Then, in the indefinite loop block (*while True:*) the function *digitalWrite()* is used to set the state of the digital pin *13* (*HIGH* or *LOW* state). With *digitalWrite()* function the LED connected to the pin *13* can be turned *ON* or *OFF* .

In the indefinite loop block, the LED is first turned *ON* for a second, and then turned *OFF* for a second. This is called *`blinking the LED`*.

The time interval of a single blink can be changed in the following line of code: *`sleep(1)`*
Where the number *1* represents the number of seconds for the duration of the time interval.

To end the infinite loop press *`CTRL + C`* on the keyboard. This is called the keyboard interrupt, which is set in the *`except`* block (`except` *`KyeboardInterrupt`*). In the *`expect`* block the on-board LED is turned *OFF*.

Basic script

Connect the KY-039 module with the Uno as shown on the connection diagram from the chapter *Connecting the module with Uno*, and then connect the Uno with the Raspberry Pi via USB cable. Next, upload *nanpy* firmware to the Uno, and use the following code to control the KY-039 module:

```
from nanpy import (ArduinoApi, SerialManager)
from time import sleep
try:
     connection1 = SerialManager()
     a = ArduinoApi(connection=connection1)
except:
     print('Failed to connect to the Arduino')

value = 0
print('[Press CTRL + C to end the script!]')
try: # Main program loop
     while True:
          value = a.analogRead(0)
          print('Output: {}'.format(value))
          sleep(0.01)

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

Save the script by the name *Heartbeat.py*. To run the script open the terminal in the directory where the scriptis saved and run the following command:

```
python3 Heartbeat.py
```

Full Python script

Now, a new script will be created, to read and smooth the signal and then to count the heartbeats. In the previous Python script, *`Heartbeat.py`* the analog signal is read and the values are displayed in the terminal.

First, a library and a tool has to be installed. If the *`pip3`* tool is not installed, open the terminal and run the following commands, one by one:

```
sudo apt-get update
sudo apt-get install python3-pip
```

Now, install the library for plotting in Python by running the following command in terminal:

```
pip3 install matplotlib
```

and wait for the installation to finish (it can last more than *10* minutes).

```
from nanpy import (ArduinoApi, SerialManager)
import time
import matplotlib.pyplot as plt
import os
def plot_em(et):
    elapsed_time = time.time() - et
    print('Script end! Elapsed time {:.3f}'.format(elapsed_time))
    y = []
    with open('new_data.txt', 'rt') as reader:
        lines = reader.readlines()
        for line in lines:
            y.append(float(line))
    x = range(len(y))
    plt.plot(x, y)
    plt.show()

try:
    connection1 = SerialManager()
    a = ArduinoApi(connection=connection1)
except:
    print('Failed to connect to the Arduino')

check_file = False
sum_a = 0
sample_range = 44
last, new = 0.0, 0.0
last_rise, new_rise = False, False
count_beats, last_beats = 0, 0
measurements = 0
```

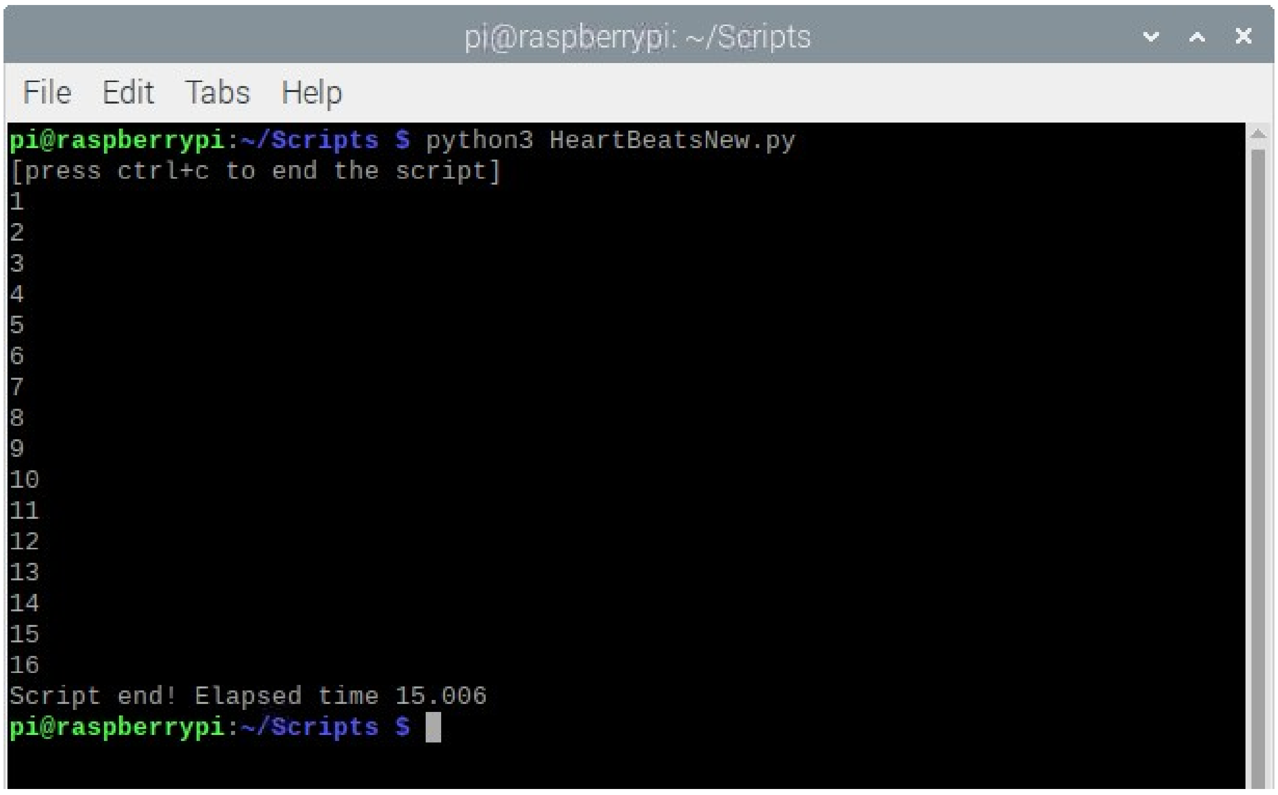
```
elapsed = time.time() # used to stop script
print('[Press CTRL + C to end the script!]')
try: # Main program loop
     t = time.time()
     while True:
          sum_a += a.analogRead(0) # sensor is connected to A0
          time.sleep(0.001)
          measurements += 1
          if measurements == sample_range:
               new = sum_a / sample_range
               if not check_file:
                    if os.path.isfile('new_data.txt'):
                         os.remove('new_data.txt')
                    check_file = True
                    with open('new_data.txt', 'at') as f:
                         f.write('{}\n'.format(new))
               else:
                    with open('new_data.txt', 'at') as f:
                         f.write('{}\n'.format(new))
               sum_a = 0
               measurements = 0
     if new > last:
             rise = True
               if new < last:
                    rise = False
               new_rise = rise
               if last_rise == True and new_rise == False:
                    count_beats += 1
               last = new
               last_rise = new_rise
```

```
# 2 tabs
          if last_beats != count_beats:
               print(count_beats)
               last_beats = count_beats
          if time.time() - elapsed > 15.0: # stop after 15sec
               plot_em(t)
               break

# Scavenging work after the end of the program
except KeyboardInterrupt:
     plot_em(t)
```

Save the script by the name *HeartbeatsNew.py*. To run the script open the terminal in the directory where the script is saved and run the following command:

`python3 HeartbeatsNew.py`

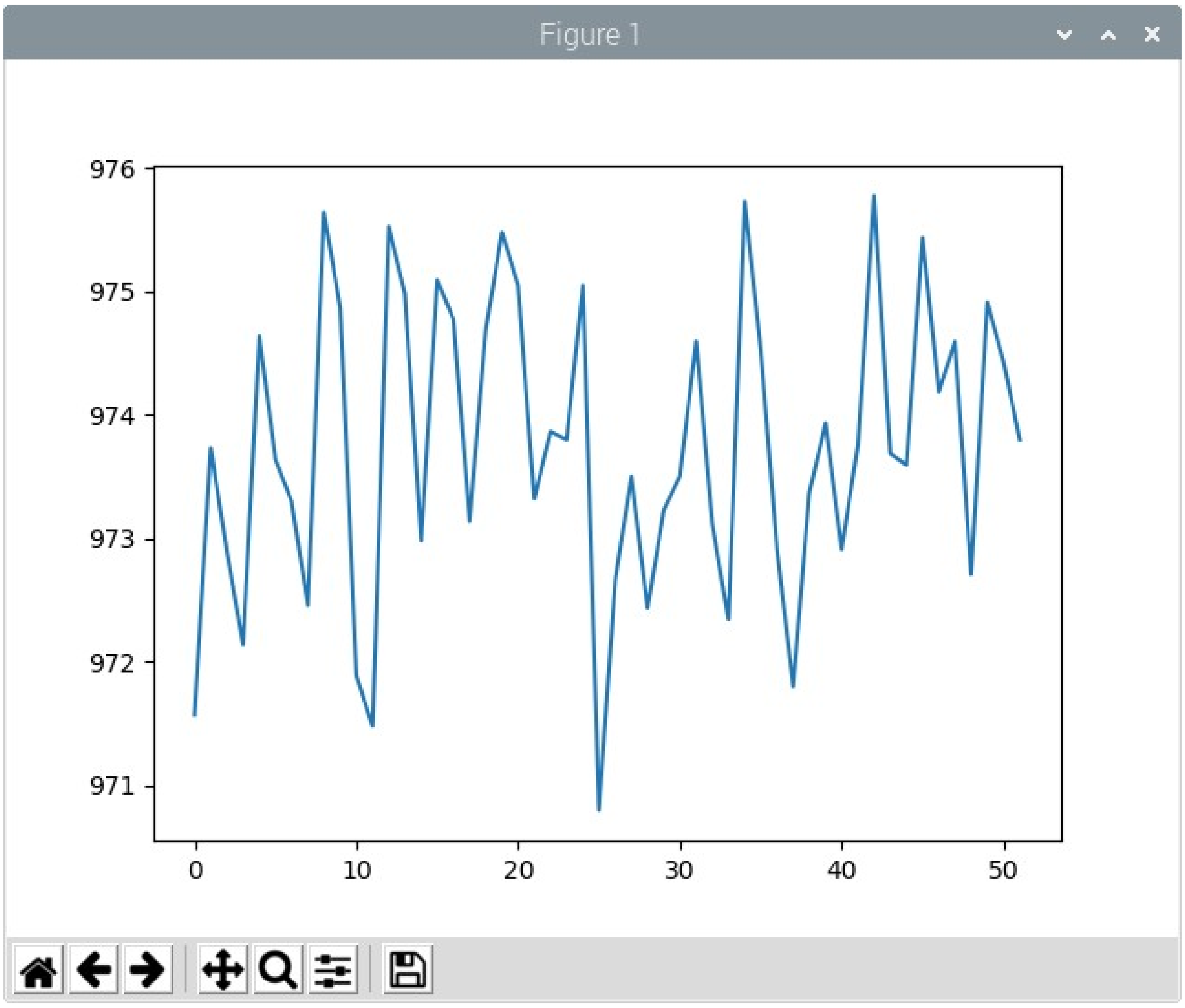
After *15* seconds script will end and the plot diagram is displayed on the screen. The result in the terminal should look like the output on the following image:



```
pi@raspberrypi:~/Scripts $ python3 HeartBeatsNew.py
[press ctrl+c to end the script]
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
Script end! Elapsed time 15.006
pi@raspberrypi:~/Scripts $ 
```

To close the window of the plot diagram press *Q* letter on the keyboard or close it with the mouse by clicking “x”.

What the script does is counting and displaying heartbeats in the terminal with the error of ±*2* beats per *20* beats. The script ends itself after *15* seconds and the plot diagram should look like the diagram on the following image:



First, the measurements are done on every *1ms*. After a number of measurements, the measurements get averaged. The average value gets saved in the file called *new_data.txt*. The number of measurements is saved in the *sample_range* variable that can be changed in other to adjust the accuracy of the counting beats.

Next, peaks on the diagram curve are detected. It is assumed that every peak is one heartbeat (which has to be tested more). After this, these peaks are counted and the number of peaks is displayed in the terminal.

To plot everything, the data has to be stored in a file. In this case the data is stored in the file called *new_data.txt*. The data from that file is read in the *plor_em()* function after which the plot diagram is displayed.

At the end of the *try* block, in the last *if* statement the time interval for script execution is created. Time interval is set to *15* seconds, but it can be easily changed to any other value:

```
if time.time() - elapsed > 15.0: # stop after 15sec
     plot_em(t)
     break
```

The script execution can be ended at any time by pressing `CTRL + C` on the keyboard, after which the plot diagram is shown.

NOTE: To get these values, protect the sensor from any other light source. The nontransparent box can be created where the sensor can be protected from the external light source. If the sensor is not hidden from other light sources, the sensor reads the infrared light from other light sources which disrort the output signal. If that happens the signal has to be cleaned from that distortions in order for measurements to be useful.

Now it is the time to learn and make your own projects. You can do that with the help of many example scripts and other tutorials, which can be found 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