JoyPi

TABLE OF CONTENTS

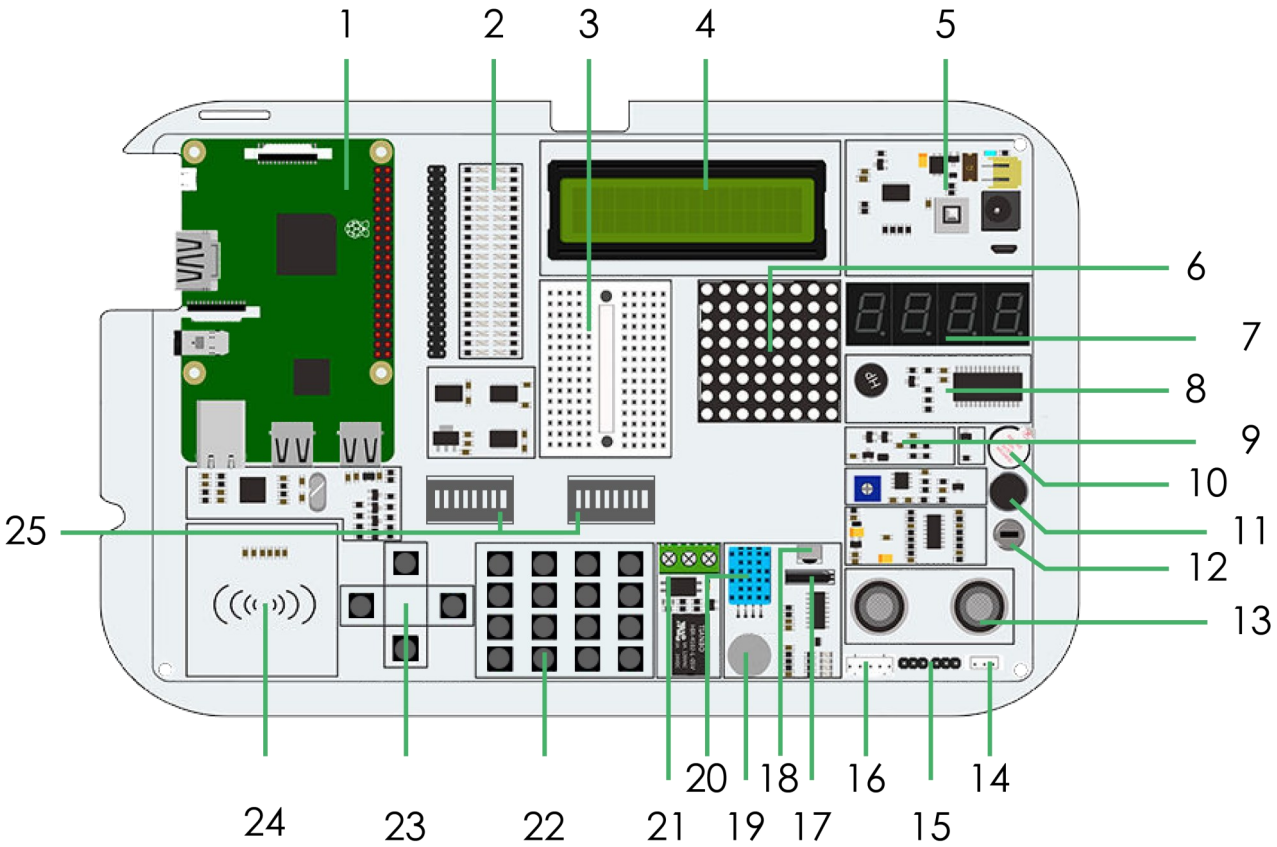
1. OVERVIEW

Dear customer,
Thank you very much for choosing our product. In the following we will show you what has to be observed during commissioning and use. Should you encounter any unexpected problems during use, please feel free to contact us.

The following lessons are designed so that, regardless of how much prior knowledge you already have, you can complete all lessons without any problems. For the different lessons you have to download sample files and run them on the JoyPi. How to do this can also be found in this manual.

But these tutorials are only the beginning.
We look forward to seeing what you will do with our Joy-Pi.

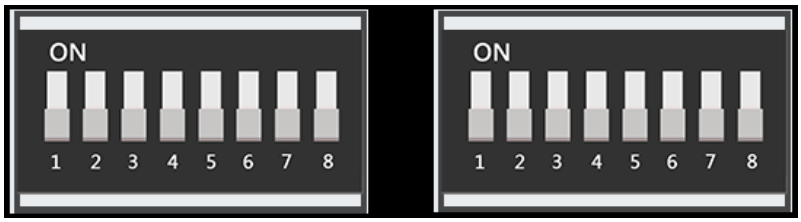
2. DETAILS



1	Raspberry Pi
2	GPIO LED Display
3	Breadboard - for creating custom curciuts with external modules
4	16x2 LCD Module (MCP23008)
5	Power supply
6	8x8 LED Matrix (MAX7219)
7	7 Segment LED display (HT16K33)
8	Vibration module
9	Light sensor - to measure the light intensity (BH1750)
10	Buzzer - to generate alarm tones
11	Sound sensor
12	Motion sensor (LH1778)
13	Ultrasonic sensor - Used for distance measurement
14 / 15	Servo interfaces - for connecting servo motors
16	Stepper motor interface
17	Tilt sensor (SW-200D)
18	Infrared sensor
19	Touch sensor
20	DH11 Sensor - for measuring humidity and temperature
21	Relay - for opening and closing electronic circuits
22	Key matrix
23	Independent keys
24	RFID module - for reading and writing data via RFID/NFC (MFRC522)
25	Switch - for switching between sensors and modules

3. CHANGING MODULES AND USING THE GPIOS

CHANGING MODULES



The JoyPi board contains 2 switching units. Each unit contains 8 switches. The switches make it possible to switch between the use of sensors and modules. Since the Raspberry Pi has only a limited number of GPIO pins, we need the switches to be able to use more sensors.

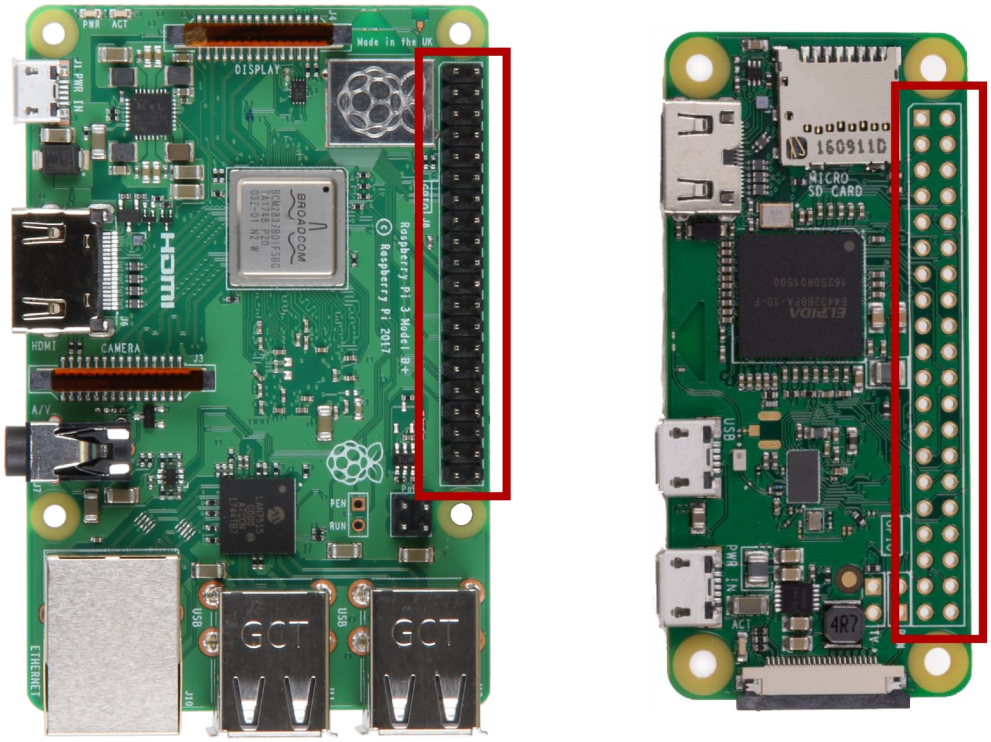
Using these switches is quite simple and is required in some of the following lessons.

Switches must be used for the following sensors:

Sensors / Modules	Switching Unit	Switches
Tastenarray	Links	1 - 8
Unabhängige Tasten	Links	1 - 8
Vibrationsmodul	Rechts	1
Neigungssensor	Rechts	2
Schrittmotor	Rechts	3, 4, 5, 6
Servomotor	Rechts	7, 8

USING THE GPIOS

In the following we will explain in more detail what GPIO's are, how they work and how they are controlled.



GPIO stands for: "General-purpose input / output" (universal input / output).
GPIO pins have no specific purpose. They can be configured as either input or output and have a general purpose. This depends on what you want to achieve.

Input pin example: A button would be considered an input because you click on it.

Output pin example: A buzzer is considered an output because you send the signal through the buzzer.

The GPIO pins are located on the right side of the Raspberry Pi board if you start from the Joy-Pi perspective.

There are 2 possible Raspberry Pi GPIO schemes: **GPIO-BOARD** and **GPIO-BCM**
The GPIO-BOARD option indicates that you are referring to the pins by the pin number. This means that the pin numbers listed below will be used.

The GPIO.BCM option means that you refer to the pins of the "Broadcom SOC Channel". These are the numbers after "GPIO" .

1	3.3V DC		2	5V DC
3	GPIO 2 (SDA1, I2C)		4	5V DC
5	GPIO 3 (SCL1, I2C)		6	Ground
7	GPIO 4		8	GPIO 14 (TXD0)
9	Ground		10	GPIO 15 (RXD0)
11	GPIO 17		12	GPIO 18
13	GPIO 27		14	Ground
15	GPIO 22		16	GPIO 23
17	3.3V		18	GPIO 24
19	GPIO 10 (SPI, MOSI)		20	Ground
21	GPIO 9 (SPI, MISO)		22	GPIO 25
23	GPIO 11 (SPI, CLK)		24	GPIO 8 (SPI)
25	Ground		26	GPIO 7 (SPI)
27	ID_SD (I2C, EEPROM)		28	ID_SC
29	GPIO 5		30	Ground
31	GPIO 6		32	GPIO 12
33	GPIO 13		34	Ground
35	GPIO 19		36	GPIO 16
37	GPIO 26		38	GPIO 20
39	Ground		40	GPIO 21

In our examples we use Python language to control the GPIO pins. In Python there is a library called "RPi.GPIO". This is a library that helps to control the pins programmatically with Python.

Take a look at the following example and the comments in the code to better understand how it works.

The first step will be to import the library by typing the command "**RPi.GPIO as GPIO**", then the "time" library comes with the command "**import time**".
Then we set the GPIO mode to GPIO.BOARD. We declare the input pin as pin number 11 for our example and the output pin as pin 12 (the input is the touch sensor and the output is the buzzer). We send a signal to the output pin, wait 1 second and then turn it off. Then, to confirm the input, we go through a loop until the **GPIO.input** input signal is received. We print "**Input Given**" to make sure that the click was confirmed, clean up the GPIO with **GPIO.cleanup ()** and finish the script.

```
import RPi.GPIO as GPIO
import time                              #importing the libraries
GPIO.setmode(GPIO.BOARD)                 #GPIO-Mode set to GPIO.BOARD
input_pin = 11
output_ = 12
GPIO.setup(input_pin, GPIO.IN)
GPIO.setup(output_pin, GPIO.OUT)         #Defining Outputs and Inputs
GPIO.output(output_pin, GPIO.HIGH)            #Output signal
Time.sleep (1)                           #wait 1 second
GPIO.output(output_pin, GPIO.LOW)             #Stop signal output
try:
      while True:                        #loop
            if (GPIO.input(touch_pin)):#check if input signal is present
                  print(“Input given!“)
            time.sleep(0.1)
Except KeyoardInterrupt:
      #ctrl+c ends the program
      GPIO.cleanup()
```

To learn more about the purpose and use of GPIO, we recommend that you read the official documentation on the RFID GPIO modules.

Follow this link to learn more about the official Raspberry Pi GPIO documentation:

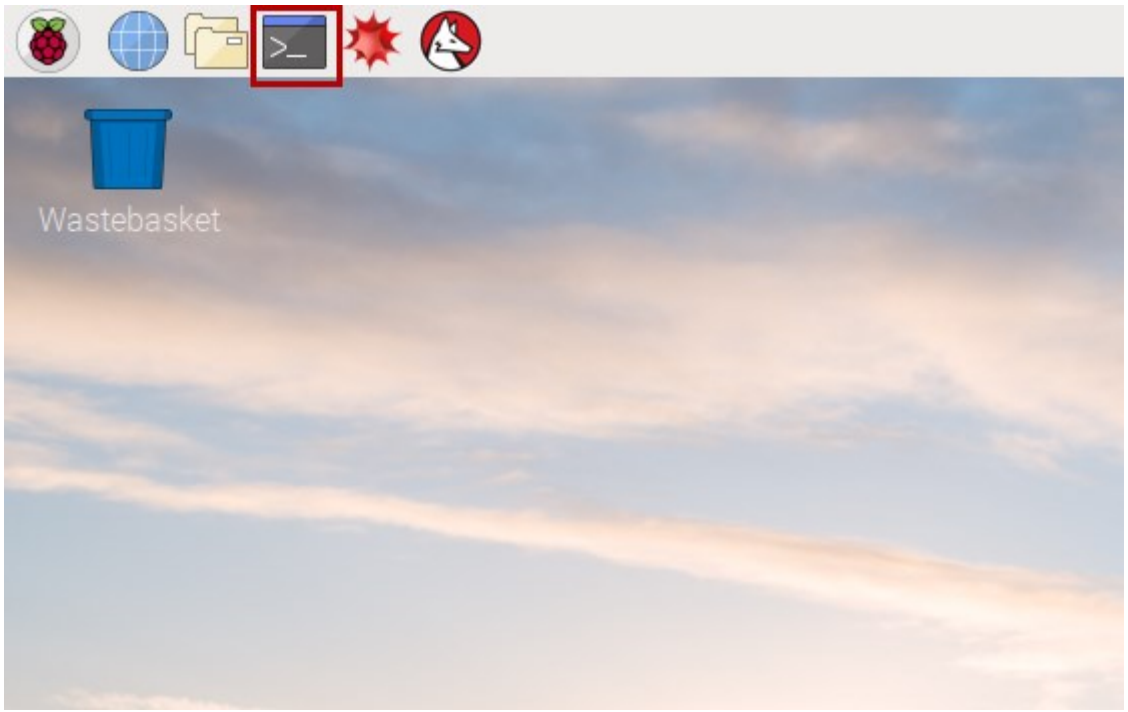
https://pythonhosted.org/RPIO/

4. USE OF PYTHON AND LINUX

This step is optional, but makes it easier to execute scripts without having to create them individually.

The scripts used in this guide can be downloaded directly from a package. Simply follow the instructions below to do this:

1. open the "**Terminal**". We use this to run most of our Python scripts and download extensions and scripts.



2. After successfully opening the terminal, we need to download the script archive with the following command:

```
wget http://anleitung.joy-it.net/wp-content/uploads/2018/10/JoyPi.zip
```

3. Press "Enter" on your keyboard. Now all you have to do is unpack the archive:

```
unzip JoyPi.zip
```

4. press "Enter" and wait until the process is completed.

5. with the command "cd" we change to the correct directory so that we can use the scripts that are in it:

```
cd Joy-Pi
```

Attention! Every time you switch off your Joy-Pi, you have to repeat the steps of changing the directory.

EXECUTING PYTHON SCRIPTS

After we have successfully downloaded our script, we would like to execute it now. Open the terminal again and follow the instructions below to run the script:

Write the command "**sudo python** <**script name**>" to execute a Python script.
For example:

```
sudo python buzzer.py
```

The sudo command gives us root permissions (admin permissions), which are later required by the GPIO library. We write "python" to tell the system that we want to execute the command with Python. At the end, we write the script name as we put it on the desktop.
downloaded.

You have successfully executed the Python script.

5. LESSONS

5.1 LESSON 1: USING THE BUZZER FOR WARNING SOUNDS

In the previous explanation, we learned how to use the GPIO pin both as output and input. To test this now, we go ahead with a real example and apply our knowledge from the previous lesson. The module we will use is the "Buzzer".

We will use the GPIO output to send a signal to the buzzer and close the circuit to generate a loud buzz. Then we will send another signal to turn it off.

The buzzer is located on the right side of the Joy-Pi-Board and is easily recognized by the loud noise that it makes when activated. When you use your Raspberry Pi for the first time, the buzzer may have a protective sticker on it. Make sure this sticker has been removed before using the Buzzer.

Just like in the previous example, we have prepared a special script with detailed comments that will explain how the whole buzzer process works, and how we can control the buzzer with the GPIOs.

First we import the **RPi.GPIO library** and the **time** library. Then we configure the buzzer. At **pin 12** we set the GPIO mode to **GPIO BOARD** and the pin as **OUTPUT**.

We output a signal for 0.5 seconds and then turn it off.

```
#!/usr/bin/python
#import the required libraries
import RPi.GPIO as GPIO
import time

#buzzer_pin is defined
buzzer_pin = 12
GPIO.setmode(GPIO.BOARD)
GPIO.setup(buzzer_pin, GPIO.OUT)

#emit noise
GPIO.output(buzzer_pin, GPIO.HIGH)

#Wait half a second.
time.sleep(0.5)

#Stop Noise output
GPIO.output(buzzer_pin, GPIO.LOW)

GPIO.cleanup()
```

Execute the following commands and try it yourself:

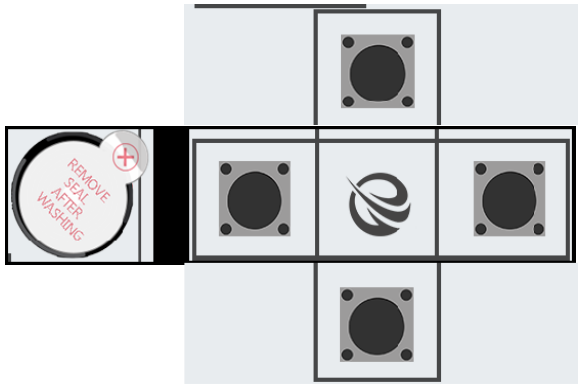
```
cd /home/pi/Desktop/Joy-Pi/
```

```
sudo python buzzer.py
```

5.2 LESSON 2: CONTROLLING THE BUZZER WITH KEY INPUTS

After successfully demonstrating how to turn the buzzer on and off, it's time to make things a little more exciting. In this lesson, we'll combine a button with the buzzer so that the buzzer is only turned on by pressing the button.

This time we will use 2 GPIO setups. One will be the **GPIO.INPUT**, which takes the button as an input, another will be the **GPIO.OUTPUT**, which sends a signal to the buzzer to output a sound.



Attention! For this example you have to switch between the modules.

In our example we use the upper of the 4 keys on the lower left side. Theoretically, however, any of the 4 keys can be used. If you still want to use another key, you have to change the pin assignment accordingly.

GPIO37	Upper button
GPIO27	Lower button
GPIO22	Left button
GPIO35	Right button

For this part of our tutorial we need to use 2 GPIO settings. One input and one output. The GPIO input is used to determine when a key was pressed and the GPIO output is used to activate the buzzer when that key is pressed.

As you can see in the example below, we have defined 2 pins called **buzzer_pin** and **button_pin**. The program runs until CTRL + C is pressed.

When you press the key on your Joy-Pi, the buzzer sounds! Release the key and the Buzzer stops.

```
#!/usr/bin/python

import RPi.GPIO as GPIO
import time

#define pins
button_pin = 37
buzzer_pin = 12

#set board mode to GPIO.BOARD
GPIO.setmode(GPIO.BOARD)

#set button_pin as input and buzzer_pin as output
GPIO.setup(button_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP
GPIO.setup(buzzer_pin, GPIO.OUT)

try:
      while True:
            #check if button is pressed
            if (GPIO.input(button_pin)):
                  #Buzzer on
                  GPIO.output(buzzer_pin, GPIO.HIGH)
            else:
                  #Buzzer off
                  GPIO.output(buzzer_pin, GPIO.LOW)

Except KeyboardInterrupt:
      GPIO.cleanup()
```

Execute the following commands and try it yourself:

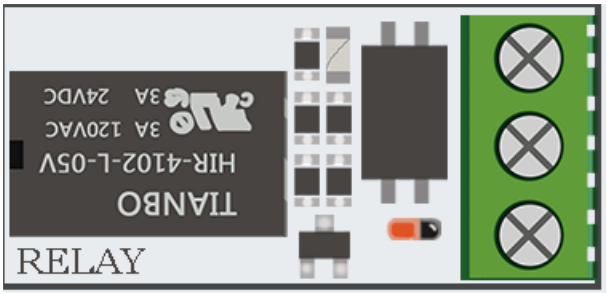
```
cd /home/pi/Desktop/Joy-Pi/
```

```
sudo python button_buzzer.py
```

5.3 LESSON 3: HOW A RELAY WORKS AND HOW TO CONTROL IT

Now that we know everything we need to know about the buzzer, it's time for the next lesson. Now we'll learn how to use the relay, what the function of the relay is and how to control it.

Relays are used to control a circuit by a separate low power signal, or when several circuits need to be controlled by one signal. In our example we show how to send a GPIO signal to open the relay to activate a custom circuit and how to send another signal to close the relay and deactivate the circuit.



The relay is located in the middle, lower part of the board, next to the key matrix. It has 3 inputs of which we will use 2 in this example (positive and negative). The positive is marked with **NC** and the negative with **NO**. The third one we do not use in this example is marked with COM.

Before we start with the code, it is important to understand how a relay works and what its task is.

If we look at the right side of the image, we can see that there are two black lines. The upper is the negative line and the lower is the positive line. By connecting positive and negative lines in the relay, we are able to create a custom electronic circuit to control, for example, an LED.

By opening the relay, we let the circuit "flow" and the LED should light up. By closing the relay, we block our circuit and our LED would no longer light up.

Attention! It is very important not to try to connect high voltage devices to the relay (e.g. table lamp, coffee machine etc.). This could result in electric shock and serious injury.

Now that we have understood what a relay is and how it works, we take a look at the code:

```
#!/usr/bin/python

import RPi.GPIO as GPIO
import time

#define relay_pin
relay_pin = 40

#Board Mode GPIO.BOARD
GPIO.setmode(GPIO.BOARD)
#relay_pin als Ausgang
GPIO.setup(relay_pin, GPIO.OUT)

#open Relay
GPIO.output(relay_pin, GPIO.LOW)
#wait half a second
time.sleep(0.5)
#close relay
GPIO.output(relay_pin, GPIO.HIGH)
GPIO.cleanup()
```

If you look closely, you may notice something unusual in the above code. Opening the relay is triggered by **GPIO.LOW** and closing is triggered by **GPIO.HIGH**.

In our previous example we used **GPIO.HIGH** to open something and **GPIO.LOW** to close something. On our JoyPi board we use a resistor for your safety. This resistor switches the directions of the relay, so GPIO.HIGH means to close the relay and GPIO.LOW means to open it.

Execute the following commands and try it yourself:

```
cd /home/pi/Desktop/Joy-Pi/
```

```
sudo python relay.py
```

5.4 LESSON 4: SENDING A VIBRATION SIGNAL

Have you always wondered how your phone vibrates when someone calls you or when you receive a message? We have exactly the same module built into our JoyPi and now we will learn how to use it.

The vibration module is located on the right side of the LED matrix and below the segment LED. When it is on, it is difficult to tell where the vibration is coming from because it feels like the whole JoyPi board is vibrating.

For this example you have to switch between the modules. Set switch number 1 of the right-hand switching unit to ON.

The vibration module uses a **GPIO.OUTPUT** signal, just like the buzzer and other modules before. By sending an output signal the vibration module vibrates, by stopping the signal with GPIO.LOW the vibration stops.

With different **time.sleep()** intervals you can control the speed of the vibration. Try it yourself and see what speed you reach.

```
#!/usr/bin/python

import RPi.GPIO as GPIO
import time

#define the vibration pin
Vibration_pin = 13

#set board mode to GPIO.BOARD
GPIO.setmode(GPIO.BOARD)

#set vibration pin as output
GPIO.setup(vibration_pin, GPIO.OUT)

#turn on vibration
GPIO.output(vibration_pin, GPIO.HIGH)

#Wait half a second
Time.sleep(0.5)

#turn off vibration
GPIO.output(vibration_pin, GPIO.LOW)

GPIO.cleanup()
```

Execute the following commands and try it yourself:

```
cd /home/pi/Desktop/Joy-Pi/
```

```
sudo python vibration.py
```

5.5 LESSON 5: DETECTING NOISES WITH THE SOUND SENSOR

In this lesson, we will learn how to use the sound sensor to make inputs, detect loud noises and react accordingly. So you can build your own alarm system that detects loud noises or turn on an LED by clapping!

The sound sensor consists of two parts: a blue potentiometer, which regulates the sensitivity, and the sensor itself, which detects the input of sounds. The sound sensor can be easily recognized by the blue potentiometer and the sensor itself is located on the right under the buzzer.

With the help of the potentiometer we can regulate the sensitivity of the sensor. For our script to work, we must first learn how to control the sensitivity. To adjust the sensitivity you have to turn the small screw on the potentiometer with a screwdriver to the left or right. The best way to test the sensitivity is to run the script. Clap your hands and see if the device is receiving a signal. If no signal is received this means that the sensitivity of the sensor is not set high enough. This can be easily corrected by turning the potentiometer.

```
#!/usr/bin/python

import RPi.GPIO as GPIO
import time

# sound_pin is defined
sound_pin = 18
# GPIO mode is set to GPIO.BOARD
GPIO.setmode(GPIO.BOARD)
# sound_pin is defined as input
GPIO.setup(sound_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)

try:
      while True:
            # check if a noise is detected
            if(GPIO.input(sound_pin)):
                  print('Sound detectet')
Except KeyboardInterrupt
      # Ctrl+c ends the program
      GPIO.cleanup()
```

Execute the following commands and try it yourself:

```
cd /home/pi/Desktop/Joy-Pi/
```

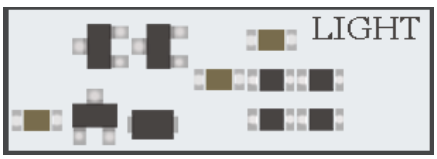
```
sudo python sound.py
```

First we define our pin, **GPIO18**. Then we set a while loop to run this script permanently. We check if we have received an input from the sound sensor indicating that loud noises have been detected and then we print "Sound Detected".

If **Ctrl + C** is pressed, the program is quit.

6.6 LESSON 6: DETECTING BRIGHTNESS WITH THE LIGHT SENSOR

The light sensor is one of our favorites. It is extremely useful in many projects and situations, e.g. with lamps that switch on automatically as soon as it gets dark. With the light sensor we can see how bright the module surface is. Since every case is different, we have to try to find out which configuration is best for us.



The light sensor is difficult to detect because it consists of very small parts. The sensor is to the left of the buzzer. If you cover it with your finger, the output of the light sensor should be close to zero, as no light can reach it.

After learning how to control sensor sensitivity, it's time to test it in real time and see how it works. However, the light sensor is a little different from other sensors because it works with I2C and not with the normal GPIOs as we learned in the lessons before.

In this script, we use binary data to control the light sensor, such as turning it on and off, high input and low input. Then we use these functions to "talk" to the light sensor and get the desired output in terms of light sensitivity.
Then we use the function "**sensor = LightSensor ()**" to measure the brightness, and "**sensor.readLight ()**" to convert the binary data into readable brightness numbers (the higher the number, the brighter).

```
#!/usr/bin/python

import time

def convertToNumber (self, data):


      #simple function to convert 2 byte data to decimal number
      return ((data[1] +(256 * data[0])) / 1.2)

def redLight(self):

      data = bus.read_i2c_block_data
(self.Device,self.ONE_TIME_HIGH_RES_MODE_1)
      Return self.convertToNumber(data)

def main():
      sensor = LightSensor()
      try:
            while True:
                  Print (“Light Level : “ + str(sensor.readLight()) + “ lx“
                  Time.sleep(0.5)
      Except KeyboardInterrupt:
            pass
if _name_ == “_main_“:
                  main()
```

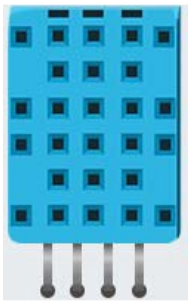
Execute the following commands and try it yourself:

```
cd /home/pi/Desktop/Joy-Pi/
```

```
sudo python light_sensor.py
```

5.7 LESSON 7: DETECTING THE TEMPERATURE AND THE HUMIDITY

The DH11 is a very interesting sensor, because it has not only one function, but two! It contains both a humidity sensor and a temperature sensor, both of which are very accurate. Ideal for any weather station project, or if you want to check the temperature and humidity in the room!



The DH11 sensor is very easy to recognize. A small blue sensor with many small holes. It is located to the right of the relay and above the touch sensor. Working with the DH11 sensor is very easy, thanks to the **Adafruit_DHT library**. The library is used to output temperature and humidity as values without having to perform complicated mathematical calculations.

```
import sys
import Adafruit_DHT

#Sensor Type (support variable)
Sensor = 11
# define Pin
Pin = 4


#Read sensor data
humiditiy, temperature = Adafruit_DHT.read_retry(sensor, pin)


#display sensor data
if humidity is not None and temperatur is not None:
      print('Temp={0:0.1f}* Humidity={1:0.1f}%' .format (temperature, humidi-
ty))
else:
      print('Failed to get reading.Try again!')
            sys.exit(1)
```

Execute the following commands and try it yourself:

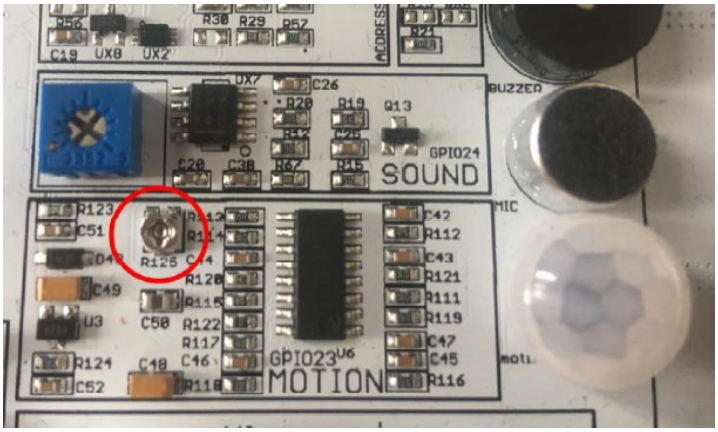
```
cd /home/pi/Desktop/Joy-Pi/
```

```
sudo python dh11.py
```

6.8 LESSON 8: DETECTING MOVEMENTS

The motion sensor is one of the most useful and frequently used sensors. It can be used, for example, to build an alarm system. When the sensor detects a movement, it can send a signal to the buzzer, which then sounds a loud alarm.

The motion sensor is located directly under the sound sensor and is covered by a small, transparent cap. The cap helps the sensor to detect more movements by refracting the infrared light of the environment. The sensitivity of the motion sensor, like that of the sound sensor, is controlled with a potentiometer. This is located below the potentiometer of the sound sensor, but is much smaller. Using a Phillips screwdriver, you can set the distances over which the motion sensor should jump.



The motion sensor is controlled by the GPIO pins. When motion is detected, the motion sensor will send a signal. This will stop for some time and then stop again until the sensor detects the next movement.

```
#!/usr/bin/python

import RPi.GPIO as GPIO
import time

#define pin
motion_pin = 16

#GPIO Mode as GPIO.BOARD
GPIO.setmode(GPIO.BOARD)
#motion_pin as input
GPIO.setup(motion_pin, GPIO.IN)

try:
      While True:
            if(GPIO.input(motion_pin) == 0):
                  print (“Nothing moves…“)
            elif(GPIO.input(motion_pin) == 1):
                  print (“Motion detected!“)
            time.sleep(0.1)
Except Keyboardinterrupt:
                  GPIO.cleanup
```

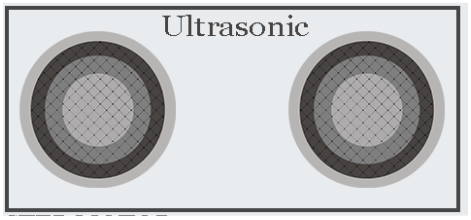
Execute the following commands and try it yourself:

```
cd /home/pi/Desktop/Joy-Pi/
```

```
sudo python motion.py
```

6.9 LESSON 9: MEASURING DISTANCES WITH THE ULTRASONIC SENSOR

Now we will learn how to use the ultrasonic sensor to measure distances and display them on the JoyPi screen. By the way, cars use the same method to measure distances.



The ultrasonic sensor is located at the bottom right of the JoyPi board, directly above the stepper motor and servo interfaces. It is easily recognizable by the two large circles. We will move our hands over the distance sensor to measure the distance between our hands and the JoyPi.

The distance sensor works with **GPIO INPUT**, but it is slightly different from what we used in our previous lessons. The sensor needs a certain interval to be able to detect the distance in an accurate way. It sends an ultrasonic signal and with a built-in sensor it receives the echo reflected by an obstacle. From the time difference between sending the signal and receiving the echo, the distance is calculated.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time
# Board mode GPIO.BOARD
GPIO.setmode(GPIO.BOARD)
#Define pins
TRIG = 36
ECHO = 32
#define inputs and outputs
GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(ECHO, GPIO.IN)

GPIO.output(TRIG, False)
print ("Waiting for Sensor to settle")
time.sleep(2)
#start measurement
GPIO.output(TRIG, True)
time.sleep(0.00001)
GPIO.output(TRIG, False)
while GPIO.input(ECHO)==0:
      pulse_start = time.time()
#calculated distance

pulse_duration = pulse_end - pulse_start
distance = pulse_duration * 17150
distance = round(distance, 2)
#output distance
print("Distance: ", distance, "cm")
           GPIO.cleanup()
```

Execute the following commands and try it yourself:

```
cd /home/pi/Desktop/Joy-Pi/
```

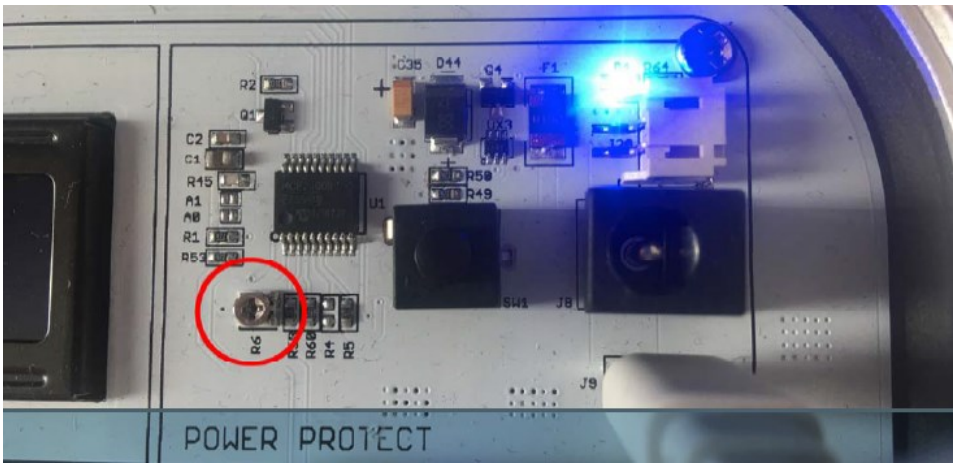
```
sudo python distance.py
```

6.10 LESSON 10: CONTROLLING THE LCD DISPLAY

With the Joy-Pi you can display the LCD data that you collect with your sensors and update it in real time depending on the changes that the modules go through. For example, in conjunction with the temperature sensor - always display the current temperature and humidity on the LCD.

The LCD screen takes up a large part of the JoyPi board - it is located at the top center of the JoyPi, to the right of the GPIO LED display. As soon as the demo script and the examples are executed, the display turns on. Thanks to the integrated backlight you can read data on the display even in complete darkness.

Like the sound and motion sensors, the LCD also has an associated potentiometer. With this potentiometer you can adjust the brightness of the backlight of the display.



The LCD and some other sensors do not work with GPIO technology. Therefore we use "**I2C**". We use the **address 21** for the LCD by establishing a connection to this I2C address. So we can send commands such as writing text, switching on the backlight of the LCD, activating the cursor, etc.

```
#!/usr/bin/python

import time
import Adafruit_CharLCD as LCD

#enter the columns and rows of the display
lcd_columns = 16
lcd_rows = 2

Lcd = LCD.Adafruit_CharLCDBackpack(adress=0x21)

#turn on the backlight
lcd-set_backlight(0)

#print a 2 line message
          lcd.message('Hello\nworld!')
```

To control the LCD we use the **Adafruit_CharLCDBackpack** library. This makes working with such a complicated product much easier.

Execute the following commands and try it yourself:

```
cd /home/pi/Desktop/Joy-Pi/
```

```
sudo python lcd.py
```

6.11 LESSON 11: READING AND WRITING RFID CARDS

In this lesson you will learn how to control the RFID module. The RFID module is a very interesting and useful module. It is used worldwide in a variety of solutions such as: Intelligent door locks, employee IDs, business cards and even dog collars.

The RFID module is located directly under the Raspberry Pi and looks like a small Wifi symbol. This symbol means wireless connectivity. To use it, we need to take the chip, or card, that comes with the JoyPi and hold it over the Joy-Pi RFID chip area. It must be close enough for our script to be recognized. 2-4cm should be close enough. Just try it out!

We have 4 functionalities: Authorize, Read, Write and Cancel Authorization.

First, the script will try to authorize the card. The chip uses the default password configuration. If the authorization was successful, it reads the data and displays it on the screen. After that it lifts the authorization to it and exits the script. In another script example, we can authorize, read, rewrite, and then unauthorize the data.

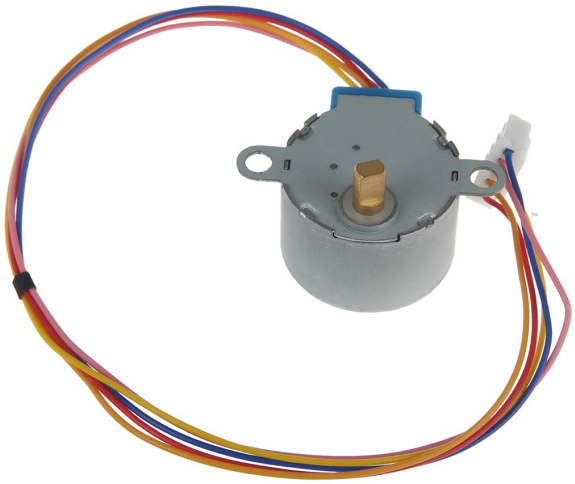
```
#!/usr/bin/python

import time

print("Starting")
while run:
      rdr.wait_for_tag()
      #read and check the card

      (error, data) = rdr.request()
      if not error:
            print ("Card read UID: "+str(uid[1])+","+str(uid[2])+","+str(uid[3]))

            print ("Setting tag")
            Util.set_tag(uid)
            print ("\nAuthorizing")
            #util.auth(rdr.auth_a, [0x12, 0x34, 0x56, 0x78, 0x96, 0x92])
            util.auth(rdr.auth_b, [0x74, 0x00, 0x52, 0x35, 0x00, 0xFF])
            print ("\nReading")
            Util.read_out(4)
            print ("\nDeauthorizing")
```

Execute the following commands and try it yourself:

```
cd /home/pi/Desktop/Joy-Pi/
```

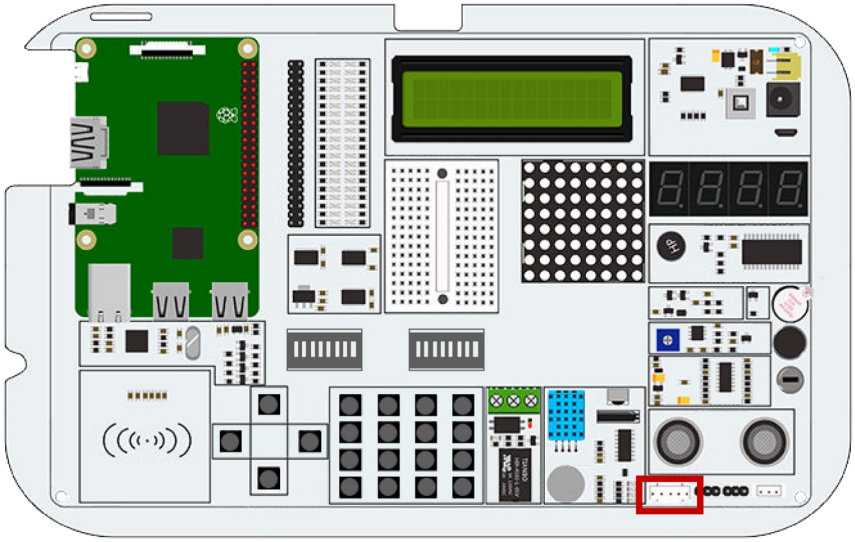
```
sudo python RFID_Read.py
```

6.12 LESSON 12: USING STEPPER MOTORS



The stepper motor is an independent module that you will have to connect to the board. We need to take the stepper motor that came with the kit and connect it to our Joy-Pi.

Simply connect the stepper motor to the following connector on the JoyPi board:



The module may heat up during use. This is due to technical reasons and is not unusual.

For this example you have to switch between the modules. Set switch numbers 3, 4, 5 and 6 on the right-hand switching unit to ON.

The stepper motor is connected to 4 GPIO pins, which are switched on quickly one after the other. This causes the stepper motor to "push" forward and take one step. Any number of steps can be executed with the **turnSteps** function. The t**urnDegrees** function rotates the motor by a certain angle.

```
def main():

print (“moving started“)
motor = Stepmotor()
# make 1 step
print (“One Step“)
motor.turnSteps(1)

sleep(0.5)
# make 20 steps
print (“20 Steps“)
motor.turnSteps(20)

sleep(0.5)
#make a quarter turn
print (“quarter turn“)
Motor.turnDegrees(90)
#stop
print (“movingstopped“)
Motor.close()

if _name_ == “_main_“:
                main()
```

Führen Sie die folgenden Befehle aus und versuchen Sie es selbst:

```
cd /home/pi/Desktop/Joy-Pi/
```

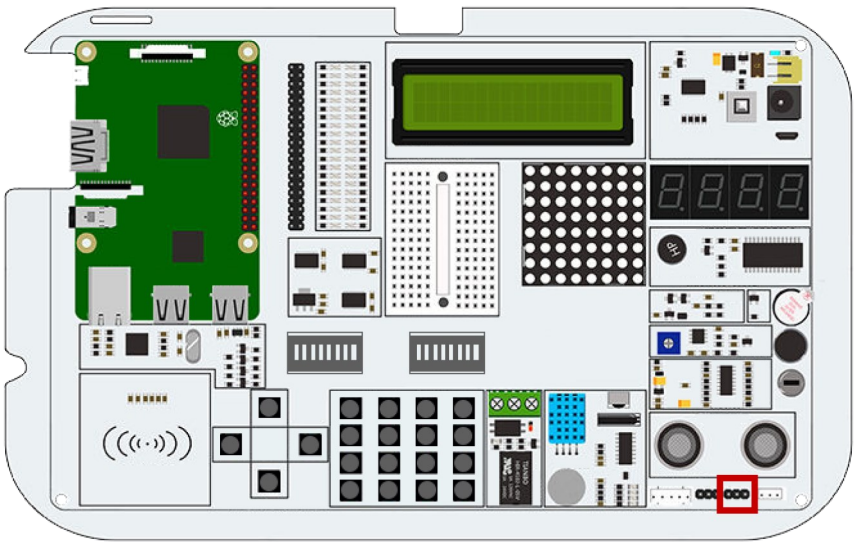
```
sudo python stepmotor.py
```

6.13 LESSON 13: CONTROLLING SERVO MOTORS

With the help of the servo motor, devices can be mechanically controlled and parts can be moved. For example, intelligent waste bins, a box of chocolates with an intelligent opening and closing door and many other interesting projects can be created.

The Joy-Pi has two servo interfaces, both of which can be used to control servo motors. In this tutorial we will use interface number one, which is marked as "Servo1". Of course you can also use the other servo interface, but you have to adapt the script to the correct GPIO's for this.

The servomotor needs three pins: positive, negative, and the data pin. The positive pin is the red cable, the negative pin is the black cable (also called ground) and the data cable is colored.



For this example you have to switch between the modules. Set switches number 7 and 8 on the right-hand switching unit to ON.

Cable	Pin
Red	Middle pin of Servo1
Black	Right pin of Servo1
Colored	Left pin of Servo1

Let's take a look at our example code to better understand it:
The servo uses the GPIO.board pin number 37. Each time the script will set the direction of the servo motor to rotate. We can use positive degrees to rotate left and negative degrees to rotate right. Just change the degrees and see how the rotation of the motor changes.

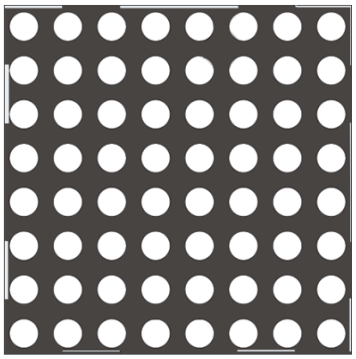
```
def main():

      #define servo_pin
      servo_pin = 37
      s = sg90(servo_pin, 0)

      Try:
            while True:
                  print(“Turn left ...“)
                  #turn counterclockwise
                  s.setdirection( 100, 10 )
                  #wait half a second
                  time.sleep(0.5)
                  print (“Turn right...“)
                  #turn clockwise
                  s.setdirection ( -100, -10 )
                  time.sleep(0.5)
            except KeyboartInterrupt:
                  s.cleanup()

if _name_ == “_main_“:
                  main()
```

Execute the following commands and try it yourself:

```
cd /home/pi/Desktop/Joy-Pi/
```

```
sudo python servo.py
```

6.14 LESSON 14: CONTROLLING THE 8X8 LED-MATRIX



The LED matrix plays an important role in many flashing LED projects. Even if you don't see it at first glance, the LED matrix can do much more than just blink red. It can be used to display information, text, emojis and even Chinese characters. Perfect for displaying information in fun and unique ways and maybe even a game like Snake or a countdown timer!

The LED matrix module is a large square module located on the left side of the segment LED and just below the LCD. It can easily be recognized by the small white dots that are the LEDs. Do not be fooled by the small size of the LEDs. This LED matrix can light up a dark place with ease!

In this example, we display a fast, long text and a slow, short text. In the script, we create a string with a message and use the **show_message()** function to display the message on the matrix display.

We can control properties, such as delays, that make the message faster or slower. For example, **scroll_delay 0** will be quite fast, while a delay of 0.1 will make the
message flow slows down a bit. The Matrix LED, unlike other modules, uses an SPI interface from which it can be controlled. Try several examples and change the code to see what happens.

```
def demo(n, block_orientation, rotate):

      serial = spi(port=0, device=1, gpio=noop())
      device = max7219(serial, cascaded=n or 1,
block_orientation=block_orientation, rotate=rotate or 0)
      print("Created device")

      #start demo
      msg = "MAX7219 LED Matrix Demo"
      print(msg)
      show_message(device, msg, fill="white", font=proportional(CP437_FONT))
      time.sleep(1)

      msg="Fast scrolling:Lorem ipsum dolor sit amet, consetetur sadipscing\
       elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna
aliquyam erat,\
       sed diam voluptua. At vero eos et accusam et justo duo dolores et ea
rebum.\
       Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum do-
lor sit amet.\
       Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam
nonumy eirmod tempor\
       invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At
vero eos et accusam\
       et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea ta-
kimata sanctus est Lorem\
       ipsum dolor sit amet"

      msg= re.sub(" +", " ",msg)
      print(msg)
      #Fast run
      show_message(device, msg, fill="white", font=proportional(LCD_FONT),
scroll_delay=0)
      Msg = "Slow scrolling: The quick brown fox jumps over the lazy dog"
      print(msg)
      #slow run
                  show_message(device, msg, fill="white", font=proportional
            (LCD_FONT), scroll_delay=0.1)
```

Execute the following commands and try it yourself:

```
cd /home/pi/Desktop/Joy-Pi/
```

```
sudo python matrix_demo.py
```

6.15 LESSON 15: CONTROLLING THE 7-SEGMENT DISPLAY

The segment LED is a very useful display when it comes to numbers and data. It can show us the time, count how many times we have done certain things and even be used to scare friends with a fake time bomb. The segment display is also used in many industrial solutions, such as elevators.

The segment display is located directly above the vibration sensor and next to the LED matrix. When it is off, 4 eights are visible. As soon as you have activated the segment display module the dark colour becomes a shiny, bright red.

In our example we demonstrate a clock. We will use the time and date modules to get the Raspberry Pi system time, which we display using the **segment.write_display()** function. The **set_digit()** function, in combination with the numbers 0,1,2 and 3, sets the position on the display where the number should be shown.

Since the current system time is retrieved in this example, it is necessary to configure the Raspberry Pi to the correct time zone first. Open a terminal window and enter the following command:

```
sudo dpkg-reconfigure tzdata
```

A window opens in which you can select your current time zone. After you have selected the correct time zone, confirm with the OK button and press Enter again to confirm.

```
#!/usr/bin/python
import time
import datetime
from Adafruit_LED_Backpack import SevenSegment

segment = SevenSegment.SevenSegment(adress=0x70)
#initialize the display
segment.begin()

print(“Press CRTL+Z to exit“)

#update the time continuously
while True:
      now = datetime.datetime.now()
      hour = now.hour
      minute = now.minute
      second = now.second

      segment.clear()
      #´Hours
      segment.set_digit(0, int(hour / 10))
      segment.set_digit(1, hour % 10)
      #Minutes
      segment.set_digit(2,int(minute / 10))
      segment.set_digit(3, minute % 10)
      #Seconds
      segment.set_colon(second % 2)
      segment.write_display()
                time.sleep(0.25)
```

Execute the following commands and try it yourself:

```
cd /home/pi/Desktop/Joy-Pi/
```

```
sudo python segment.py
```

6.16 LESSON 16: RECOGNIZE TOUCHES

The touch sensor is very useful when it comes to key functions. Many products on the market use touch instead of pressing a button, such as smartphones and tablets.
The touch sensor is located directly below the DH11 sensor and to the right of the relay.
The easily accessible positioning on the Joy-Pi allows easy operation.

The touch sensor works like any other key module. The only difference is that it only needs to be touched instead of pressed. By touching the touch sensor, the module closes a circuit because the computer detects that the sensor has been touched. The touch sensor uses GPIO Board Pin 11.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time

#definiere touch_pin
touch_pin = 11

#setze Board Modus zu GPIO.BOARD
GPIO.setmode(GPIO.BOARD)
#setze GPIO Pin als Eingang
GPIO.setup(touch_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)

try:
      while True:
            #überprüfe ob Berührung erkannt wurde
            if (GPIO.input(touch_pin)):
                  print("Touch detected!")
            time.sleep(0.1)
Except KeyoardInterrupt:
      #strg+c beendet das Programm
                  GPIO.cleanup()
```

Execute the following commands and try it yourself:

```
cd /home/pi/Desktop/Joy-Pi/
```

```
sudo python touch.py
```

6.17 LESSON 17: DETECTING TILTS WITH THE TILT SENSOR

The tilt sensor allows us to detect an inclination to the right or left. It is used in robotics and other industries to ensure that things are held straight. It's a small, elongated, black sensor that lies between the DH11 sensor and the ultrasonic sensor and can easily be detected by the sound it makes when you shake the board a little.
You could easily think that something inside the Joy-Pi-Board is damaged when you hear this noise. but this noise is completely normal. When the tilt sensor is tilted to the left, the circuit is activated and a GPIO HIGH signal is sent. If the tilt sensor is tilted to the right, the circuit is deactivated and a GPIO LOW signal is sent.

For this example you have to switch between the modules. Set switch number 2 of the right-hand switching unit to ON.

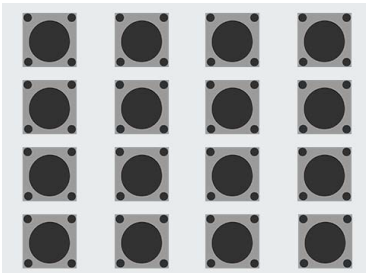
```
#!/usr/bin/python

import time
import RPi.GPIO as GPIO

#tilt_pin is defined
tilt_pin = 15

#GPIO mode is set to GPIO.BOARD
GPIO.setmode(GPIO.BOARD)

# Pin is defined as input
GPIO.setup(tilt_pin, GPIO.IN)

try:
      while True:
            #positive is tilted to the right, negative is tilted to the left
            if GPIO.input(tilt_pin):
                  print ("[-] Left Tilt")
            else:
                  print ("[-] Right Tilt")
            time.sleep(1)
except KeyboartInterrupt
      #Ctrl+c ends the program
                  GPIO.cleanup()
```

Execute the following commands and try it yourself:

```
cd /home/pi/Desktop/Joy-Pi/
```

```
sudo python tilt.py
```

6.18 LESSON18: USING THE BUTTON MATRIX



The button matrix is a module with 16 independent buttons that can be used for many projects such as a keyboard or a memory game. The great possibilities of the keys allow you to do almost anything.

The button matrix is located at the bottom center of the Joy Pi board, to the right of the relay. It is easily recognizable by the 16 individual buttons. The excellent positioning on the board allows easy operation of the keys while still providing a good overview of all other sensors.

The button matrix consists of four columns and rows. We configure the matrix rows and columns with their GPIO pins and initialize the **ButtonMatrix()** object as a button variable. Then we can press each button of the matrix and see if it has been pressed.

In our example, after recognizing a keystroke, we activate the function **activateButton()**, which displays the number of the pressed button. You can of course edit this module to do anything you can imagine.

For this example you have to switch between the modules. Set **ALL** switches of the left switching unit to **ON**.

```
def main():
      #initialize the button matrix
      buttons = ButtonMatrix()
      try:
            while True:
                  for j in range(len(buttons.columnPins)):
                        #set output pins to LOW
                        GPIO.output(buttons.columnPins[j], 0)
                        for i in range(len(buttons.rowPins)):
                              if GPIO.input(buttons.rowPins[i]) == 0:
                                    #button pressed, activate
                                    buttons.activateButton(i, j)
                                    #do nothing as long as button remains pressed
                                    While(buttons.buttonHeldDown(i)):
                                          pass
                        #set output pins to HIGH
                        GPIO.output(buttons.columnPins[j], 1)
      except KeyboardInterrupt:
            GPIO.cleanup()
if _name_ == "_main_":
                  main()
```

Execute the following commands and try it yourself:

```
cd /home/pi/Desktop/Joy-Pi/
```

```
sudo python button_matrix.py
```

6.19 LESSON 19: CONTROLLING AND USING THE IR-SENSOR

In this lesson, we will learn how to use the infrared receiver and how to get IR codes from a remote control receives. The use of this method is extremely useful because we can use different define actions for different buttons. With a remote control we can, for example, switch on different LEDs or control the servo motor each time the button is pressed.

The IR sensor is located to the right of the DH11 sensor and above the tilt sensor. It looks like a small LED with 3 pins. We also need the IR remote control, which is included in the Joy-Pi-Kit.

The IR receiver uses a library called **LIRC** and **Python-LIRC** to receive and understand the codes we send with the IR remote control. The Out variable contains the key we pressed. Using if queries, we can check whether certain keys have been pressed. This information allows us to execute the appropriate commands.

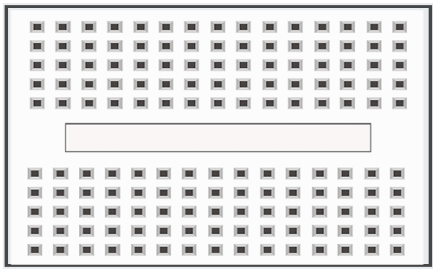
```
import socket,signal
import lirc, time, sys
import RPi.GPIO as GPIO
from array import array

GPIO.setmode(11)
GPIO.setup(17, 0)
GPIO.setup(18, 0)
PORT = 42001
HOST = “localhost“
Socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
Lirc = lirc.init(“keys“)
#lirc.set_blocking(False, Lirc)

def handler(signal.SigTSTP, handler):
      Socket.close
      GPIO.cleanup()
      exit(0)

signal.signal(signal.SIGTSTP, handler)

def sendCmd (cmd):
      n = len(cmd)
      a = array(‘c‘)
      a.append(chr((n >> 24) & 0xFF))
      a.append(chr((n >> 16) & 0xFF))
      a.append(chr((n >> 8) & 0xFF))
      a.append(chr(n & 0xFF))
      Socket.send(a.tostring() + cmd)
while True:
      Out = lirc.nextcode ()
                  print (Out[0])
```

Execute the following commands and try it yourself:

```
cd /home/pi/Desktop/Joy-Pi/
```

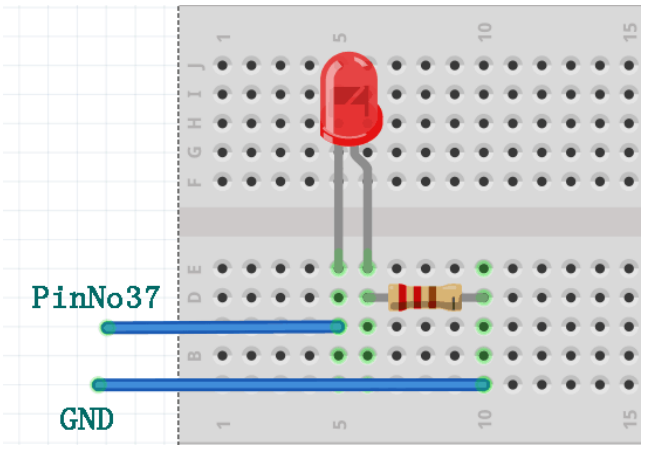
```
sudo python IR.py
```

6.20 LESSON 20: OWN CIRCUITS WITH THE BREADBOARD



The breadboard is an extremely useful part of the Joy-Pi that allows us to create our own circuits and functions. Now that we've learned how to use all the sensors, it's time to create our own. In this lesson you will create your first custom circuit using a flashing LED example. The breadboard is located in the middle of the Joy Pi board. It is a small, white, board with many small holes.

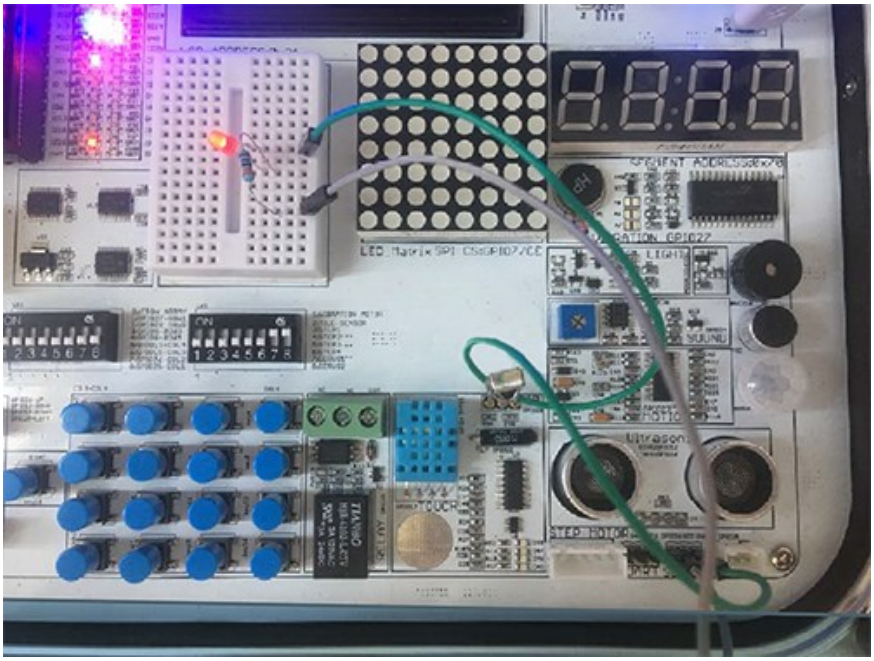
We will create a custom circuit with the function to make an LED blink. To do this, we need to use GPIO as output and GND, as we already did in earlier lessons. We will connect the **servo interface** (SERVO1 interface) to **GPIO 37**.



You can use this picture as a guide to create your circuit on the plug-in board.
Remember that pin number 37 is on the GPIO port and GND is on the GND port of the SERVO1 interface.

For this example you have to switch between the modules because the servo pins are used. Set the switches **7** and **8** of the right switching unit to **ON**.

Wir müssen einen Widerstand verwenden, der mit dem Joy-Pi - Paket geliefert wird, und ihn an die negative Seite der LED anschließen (die negative Seite der LED ist die mit dem kürzeren Bein). Die andere Seite des Widerstands werden wir direkt, mit dem Kabel, mit dem GND-Pin an der SERVO1-Schnittstelle verbinden. Die positive Seite der LED verbinden wir mit dem GPIO37 Pin der SERVO1-Schnittstelle.



We must use a resistor supplied with the Joy-Pi package and connect it to the negative side of the LED (the negative side of the LED is the one with the shorter leg). We will connect the other side of the resistor directly to the GND pin on the SERVO1 interface using the cable. Connect the positive side of the LED to the GPIO37 pin of the SERVO1 interface.

```
#!/usr/bin/python

import time
import RPi.GIPO as GPIO

#define LED Pin
led_pin = 37

#set GPIO mode to GPIO.BOARD
GPIO.setmode(GPIO.BOARD)
#set pin as output
GPIO.setup(led_pin, GPIO.OUT)

try:
      while True:
            #LED on
            GPIO.output(led_pin, GPIO.HIGH)
            #wait 0,2 seconds
            time.sleep(0.2)
            #LED off
            GPIO.output(led_pin, GPIO.LOW)
            #wait 0,2 seconds
            time.sleep(0.2)

except KeyboardInterrupt:
      #CTRL+C to exit the program
                  GPIO.cleanup()
```

Execute the following commands and try it yourself:

```
cd /home/pi/Desktop/Joy-Pi/
```

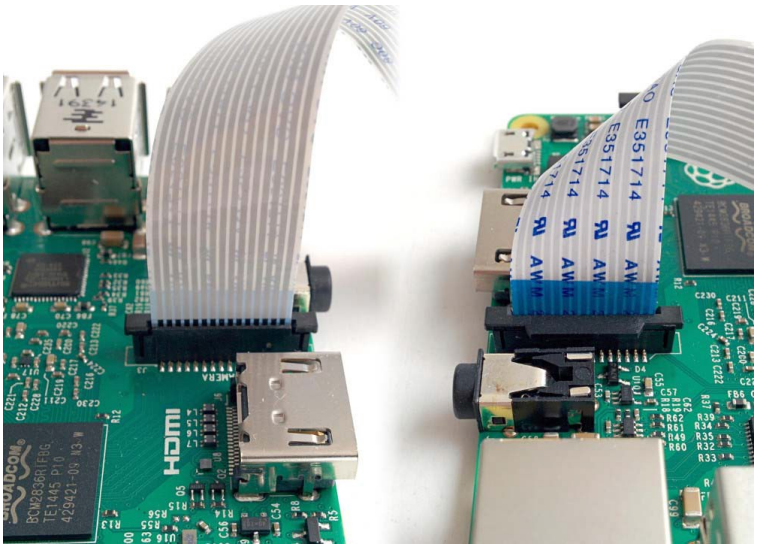
```
sudo python blinking_led.py
```

6.21 LESSON 21: PHOTOGRAPHING WITH THE RASPBERRY PI CAMERA

The Raspberry Pi camera is extremely useful and can be used for a variety of projects. For example for security cameras, face recognition and much more. In the following lesson we will introduce you to the basics of using the Raspberry Pi camera. This will teach you how to take a picture.

The camera is located centrally above the Joy-Pi's screen and is connected directly to the Raspberry Pi with a thin ribbon cable.

When connecting the cable, pay attention to the orientation of the cable. In the following picture you can see how to connect the camera correctly. By slowly and gently lifting the black connector, you can easily insert the cable. Close the black plug so that it can hold the Raspberry Pi camera cable.



Next, after making sure that the camera is connected, it's time to take a picture. We use a command called "**raspistill**" to take the picture. The -o parameter specifies the file name under which we want to save the image, and -t specifies how long the camera waits for the shot.

For -t we get a value of 5000, which corresponds to 5000 milliseconds. This results in 5 seconds to be able to prepare properly and take a nice picture. First we change to the Desktop directory to save the image to the desktop.

Try it and execute the following commands:

```
cd Desktop
```

```
raspistill –o picture.jpg –t 5000
```

After executing the commands, a video should appear and you should be able to see yourself in it. The video stays open for about 5 seconds and then closes. This means that the image has been captured and is ready to be viewed.

A file named "picture.jpg" should now appear on your desktop, which you can open.

6. INFORMATION AND TAKE-BACK OBLIGATIONS

Symbol on electrical and electronic equipment

This crossed-out dustbin means that electrical and electronic equipment does not belong in the household waste. You must return the old appliances to a collection point. Before handing over waste batteries and accumulators that are not enclosed by waste equipment must be separated from it.

Return options
As an end user, you can return your old appliance (which essentially fulfils the same function as the new appliance purchased from us) free of charge for disposal when you purchase a new appliance. Small appliances with no external dimensions greater than 25 cm can be disposed of in normal household quantities independently of the purchase of a new appliance.

Possibility of return at our company location during opening hours
Simac GmbH, Pascalstr. 8, D-47506 Neukirchen-Vluyn

Possibility of return in your area
We will send you a parcel stamp with which you can return the device to us free of charge. Please contact us by e-mail at Service@joy-it.net or by telephone.

Information on packaging
If you do not have suitable packaging material or do not wish to use your own, please contact us and we will send you suitable packaging.

7. SUPPORT

We also support you after your purchase. If there are any questions left or if you encounter any problems, please feel free to contact us by mail, phone or by our ticket-system on our website.

E-Mail: service@joy-it.net
Ticket-System: http://support.joy-it.net
Telefon: +49 (0)2845 98469 – 66 (11- 18 Uhr)

For further information please visit our website:

www.joy-it.net