Smart Home Heizungssteuerung – Open-Source Anleitung

Im folgenden Dokument findest du kompakt unsere Lösungsansätze und Musterlösungen zu den ersten drei Modulen deines Smart-Home-Workshops. Modul 1 führt durch die Einrichtung des Raspberry Pi, die Installation wichtiger Tools und den SSH-Zugriff. Modul 2 erklärt den SCD41-Sensor für CO_2, Temperatur und Luftfeuchtigkeit, inklusive Verkabelung, Beispielcode und häufigen Fehlerquellen. In Modul 3 zeigen wir, wie du LEDs und ein Relais anschließt, programmierst und mit Debugging-Tipps ausstattest.

Die drei Module des bauen inhaltlich aufeinander auf. Jede Einheit erweitert die bestehende Schaltung und den Code um neue Komponenten, vom grundlegenden Setup über die Sensorik bis hin zur Aktorsteuerung. Alle Bauteile, Programmabschnitte und Konfigurationen ergänzen sich Schritt für Schritt.

Inhaltsverzeichnis

1. Raspberry Pi als Entwicklungsplattform

Der Raspberry Pi dient als “Mini-Computer” und eine zentrale Steuereinheit für unser Dashboard. Er wird eingerichtet, mit dem Netzwerk verbunden und per SSH vom PC aus gesteuert, sodass wir bequem Programmcode darauf ausführen können. Außerdem installieren wir benötigte **Tools/Bibliotheken** und organisieren unser Projekt in einer sinnvollen **Verzeichnisstruktur**.
*(**Hinweis**: Wir setzen voraus, dass ein Raspberry Pi mit installiertem Raspberry Pi OS bereitliegt.)*

1.1 Raspberry Pi einrichten und SSH-Zugriff

Um den Raspberry Pi in Betrieb zu nehmen, installiere zunächst ein aktuelles Raspberry Pi OS auf einer microSD-Karte (z.B. mit dem Raspberry Pi Imager). Für einen Betrieb ohne Monitor/Tastatur kannst du SSH direkt aktivieren. Am einfachsten geht das über die erweiterten Optionen im Raspberry Pi Imager („Enable SSH“).
Hier findest du eine genaue Anleitung: Anleitung Einrichtung SSH Raspberry

Sobald der Pi hochfährt, ist SSH aktiviert. Nach dem Booten des Pi verbindest du ihn mit dem Netzwerk (LAN oder WLAN). Die IP-Adresse des Pi kannst du entweder im Router-Menü ablesen oder durch Eingabe von ***hostname -I*** im Terminal ermitteln. Notiere dir diese IP-Adresse, um später per SSH darauf zugreifen zu können. Nun kannst du dich von deinem PC aus per SSH mit deinem Raspberry Pi verbinden. Unter Windows verwendest du dazu z. B. PuTTY oder die PowerShell, unter Linux/macOS einfach das Terminal.

Der Befehl sieht typischerweise so aus: ***ssh pi@192.168.178.42***
Dabei ist Pi der Benutzername deines Raspberry Pi (sofern du ihn nicht anders festgelegt hast), und 192.168.178.42 ist dessen IP-Adresse.
(Hinweis: Standardbenutzer ist meist Pi mit Passwort Raspberry, sofern du keinen eigenen Benutzer gesetzt hast)
Wenn die Verbindung steht, hast du eine Shell auf dem Raspberry Pi und kannst Kommandos eingeben.

Tipp: *Aktiviere in der Raspberry Pi Konfiguration die I^2C-Schnittstelle, die wir für den Sensor brauchen (Details dazu in Abschnitt 2.4 Debugging). Dies machst du indem du den Befehl **sudo raspi-config** in das Terminal eingibst.*
Danach navigierst du im Menü zu „Interfacing Options“, dort findest du dann die „I2C“ Schnittstelle und kannst diese aktivieren/deaktivieren. Abschließend speicherst du die Einstellungen und startest deinen Raspberry Pi neu.

1.2 Installation wichtiger Tools und Bibliotheken

Für unser Projekt benötigen wir einige **Python-Bibliotheken**, um auf die Hardware-Pins und den I²C-Bus zuzugreifen:

Gpiozero

Eine einfach zu nutzende High-Level-Bibliothek, die auf RPi.GPIO aufbaut. Sie ist oft bereits in Raspberry Pi OS enthalten. Damit lassen sich LEDs, Taster, Sensoren etc. sehr leicht steuern, ohne viel Setup-Code.
Falls nötig, installieren mit Befehl: ***sudo apt install python3-gpiozero.***

Adafruit CircuitPython SCD4x

Die Bibliothek für den SCD40/SCD41-Sensor. Sie übernimmt im Hintergrund die I²C-Kommunikation, sodass kein direkter Einsatz von smbus/SMBus2 erforderlich ist.
(Installation: ***pip3 install adafruit-circuitpython-scd4x****)*

Tutorial Installation Adafruit Bibliothek: learn.adafruit.com

Zusätzlich sind Entwicklungstools wie ein Texteditor wichtig. Du kannst z.B. den Editor ***nano*** im Terminal nutzen, wie genau, wird weiter unten gezeigt, oder auch Dateien via Visual Studio Code (per Remote-SSH Extension) bearbeiten.

1.3 Projekt-Verzeichnisorganisation

Es ist gute Praxis, für dein Projekt einen eigenen Ordner anzulegen, sodass du alle Dateien übersichtlich geordnet hast und jederzeit weißt wo etwas liegt. Erstelle z.B. im Home-Verzeichnis des Pi einen Ordner.

mkdir /smart-home-dashboard
Erzeugt einen Ordner smart-home-dashboard

cd /smart-home-dashboard
Wechselt in den Ordner

In diesem Ordner kannst du dann deinen Code modular in verschiedenen Dateien strukturieren, beispielsweise:

sensor_scd41.py – Code zum Auslesen des SCD41-Sensors (Sensordaten erfassen).
led_control.py – Code zum Steuern der LED(s).
relay_control.py – Code zum Steuern des Relais (z.B. Heizmatte schalten).
main.py – Hauptprogramm, das alle Komponenten zusammenführt (optional, z.B. um periodisch Messwerte zu lesen und je nach Wert Aktoren zu schalten).

Tipp: *Beim Programmieren auf dem Raspberry Pi kannst du parallel mehrere Terminal-Tabs nutzen: z.B. einen, in dem ein Sensor-Skript läuft, und einen weiteren, um andere Dateien zu bearbeiten oder Debug-Commands auszuführen.*

2. Sensorik mit dem SCD41

Der SCD41 ist das “Auge” unseres Smart Home Dashboards für die Luftqualität. Er misst CO_2-Gehalt, Temperatur und relative Luftfeuchtigkeit der Umgebung
Es handelt sich um einen hochpräzisen NDIR-CO_2-Sensor der Firma, der Photoakustik-Technologie nutzt, um CO_2 direkt zu messen.
Hier findest du weitere Informationen zum genaueren Verständnis der Funktionsweise falls du Interessiert bist schaue hier mal vorbei: learn.adafruit.com

Der SCD41 kann CO_2 Konzentrationen von 400 bis 5000 ppm (Parts per Million) erfassen bei einer Genauigkeit von ± 5% vom Messwert. Die Betriebsspannung liegt zwischen 2.4–5V, Kommunikation erfolgt über den **I²C-Bus**.

2.1 Technische Grundlagen des SCD41

Der SCD41 Sensor eignet sich ideal, um das Raumklima zu überwachen. Denn CO_2-Werte sind ein guter Indikator für Luftqualität. So hat frische Außenluft hat einen ungefähren Wert von 400 ppm CO_2, innen gelten <800 ppm als optimal, >1000 ppm als kritisch.

Der Sensor hilft also z.B. festzustellen, wann gelüftet werden sollte. Temperatur und Feuchtigkeit ergänzen das Bild des Raumklimas (Komfort, Schimmelgefahr etc.
Der Sensor kommt meist als kleines Board daher, das den eigentlichen Sensirion-Chip samt notwendiger Schaltung enthält. Dieses Board hat beschriftete Anschlüsse (Pins) zum Verbinden mit dem Raspberry Pi.

Pinbelegung:
VIN: Spannungsversorgung
3Vo: 3.3V-Ausgang von on-board Regler
GND: Masse
SCL: I²C Clock
SDA: I²C Data

Sensor-Modul Abbildung

Du erkennst die Pins **VIN, 3Vo, GND, SCL, SDA** auf der Platine. Für unseren Anschluss an dem Pi sind vor allem VIN, GND, SCL und SDA relevant – sie verbinden den Sensor mit Strom und I²C-Datenleitung des Raspberry Pi. (Das Metallgehäuse in der Mitte ist die Sensorkammer für CO_2.)

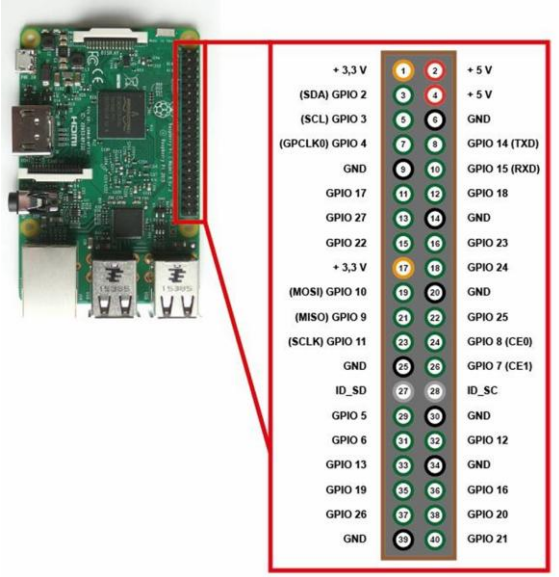
2.2 Verkabelung des SCD41 an den Raspberry Pi (I²C)

Der Raspberry Pi hat mehrere GPIO-Pins, von denen bestimmte für I²C reserviert sind. Standardmäßig nutzt I²C den Bus **I2C-1** mit den Pins:

GPIO 2 (Pin 3) – SDA (Datenleitung)
GPIO (Pin 5) – SCL (Clock-Leitung)
5V für **VIN** und **GND (Pin 6)** für Masse

Verbinde den SCD41 folgendermaßen mit dem Raspberry Pi:
I: VIN des Sensors an **5V** des Pi
II: GND des Sensors an **GND** des Pi (Pin 6).
III: SCL des Sensors an **GPIO3 (SCL)** des Pi (Pin 5).
IV: SDA des Sensors an **GPIO2 (SDA)** des Pi (Pin 3).
Achte darauf, dass die Verbindungen korrekt und fest sitzen. Vertausche nicht SDA und SCL, und schließe VIN nicht an einen falschen Pin an. Im Zweifelsfall konsultiere das Raspberry Pi Pinout.

Abbildung der Pinbelegung eines Raspberry:



Tipp:
I²C-Kabel sollten möglichst kurz sein, um Störeinflüsse zu minimieren.

Warum I²C statt „normaler“ GPIO?
I²C ist ein Bus-System um mehrere Geräte gleichzeitig mit dem Raspberry Pi zu verbinden. Jedes Gerät bekommt eine einzigartige Adresse, sodass das Pi die Sensoren oder Aktoren eindeutig ansprechen kann. Das ist besonders praktisch, wenn du später noch zusätzliche Sensoren hinzufügen möchtest – du klemmst sie einfach auf die gleichen Leitungen.
Im Gegensatz dazu sind „normale“ GPIO-Pins meist für einfache Signale (z. B. LED an/aus, Taster gedrückt/nicht gedrückt) gedacht. Der SCD41-Sensor braucht hingegen eine digitale Datenübertragung, um seine komplexen Messwerte (CO_2, Temperatur, Luftfeuchtigkeit) zu senden. Genau dafür wurde I²C entwickelt: Es ermöglicht schnellen Datenaustausch mit nur minimalem Verdrahtungsaufwand.

Hinweis: Bevor der Sensor kommunizieren kann, muss die I²C-Schnittstelle des Raspberry Pi aktiviert sein. Falls noch nicht geschehen, führe den Befehl sudo raspi-config aus. Wähle danach “Interfacing Options”, dort findest du “I2C” und aktiviere diese. Wie du das machst, kannst du in Kapital 1.1 Nachlesen. Nach einem Neustart ist dann I2C aktiviert.

2.3 Messwerte mit dem SCD41 erfassen – ein Beispiel

Nun schreiben wir ein kleines Python-Programm, um Messwerte vom SCD41 zu lesen. Wir nutzen die Adafruit Bibliothek, da sie uns viel Arbeit abnimmt. Stelle sicher, dass **Adafruit Blinka** und **Adafruit SCD4x** Library installiert sind (siehe Abschnitt 1.2).
Erstelle die Datei **sensor_scd41.py** und füge folgenden Code ein:

Link zum Code:
https://github.com/DIYSensorikWorkshop/DIY-Smart-Sensorik-Workshop/blob/main/Python%20Code%20Sensorik/Part1_scd41.py

```
# Hier fügen wir alle nötigen Bibliotheken ein
import time
import board
import busio
from adafruit_scd4x import SCD4X

# Mit diesem Code wird die I²C-Schnittstelle des Raspberry Pi initialisiert
i2c = busio.I2C(board.SCL, board.SDA)

# Hier starten wir den SCD41-Sensor
scd41 = SCD4X(i2c)
scd41.start_periodic_measurement()
print("SCD41-Sensor gestartet. Warte auf erste Messwerte...")

# Der Sensor liefert alle 5 Sekunden neue Werte
time.sleep(5)

try:
    while True: # Solange der Prozess nicht abgebrochen wird führt der Raspberry folgenden Schleife immer wieder aus.
        if scd41.data_ready:  # Hier wird geprüft, ob neue Messwerte verfügbar sind
            co2 = scd41.CO2  # CO₂ in PPM
            temperature = scd41.temperature  # Temperatur in °C
            humidity = scd41.relative_humidity  # Luftfeuchtigkeit in %

            # Messwerte im Terminal ausgeben
            print(f"CO₂: {co2} ppm | Temperatur: {temperature:.1f} °C | Luftfeuchtigkeit: {humidity:.1f} %")
        else:
            print("Warte auf neue Sensordaten...")

        time.sleep(5)  # Alle 5 Sekunden neue Messwerte abrufen
# Falls du eine Taste drückst wird der Prozess abgebrochen.
    print("\nProgramm beendet. Sensor gestoppt.")
```

Was macht der Code? Zuerst importieren wir die nötigen Module. Dann initialisieren wir die I²C-Schnittstelle mit board.I2C (). Die Adafruit-Bibliothek kennt die Standardpins und erstellt so ein I²C-Bus-Objekt. Anschließend erstellen wir ein SCD4X-Objekt namens scd, das den Sensor repräsentiert. Mit ***scd.start_periodic_measurement()*** starten wir die Messung. Nun nimmt der Sensor alle 5 Sekunden einen neuen Messwert auf. In der Schleife überprüfen wir mit ***scd.data_ready,*** ob schon neue Daten bereitstehen. Wenn ja, dann lesen wir Werte und gebe diese auf der Konsole aus. Danach warten wir 5 Sekunden, bevor wir erneut prüfen (der Sensor liefert ca. alle 5 Sek. neue Werte, häufiger abfragen ist nicht nötig). Den Wert kann natürlich nach Belieben anpassen.

Hast du alles angeschlossen und den Code eingefügt dann kannst du nun das das Skript auf dem Raspberry Pi mit python3 sensor_scd41.py starten. Du solltest nun im Terminal bzw. CMD fortlaufend Messwerte sehen, z.B.:
CO2: 420 ppm | Temp: 22.5 °C | Feuchte: 45.0 %
CO2: 421 ppm | Temp: 22.5 °C | Feuchte: 45.0 %
CO2: 424 ppm | Temp: 22.4°C | Feuchte: 42.0 %

Wie führen wir das Skript aus?

Kopiere den Beispielcode Zwischenablage. Öffne anschließend auf deinem Computer ein Terminal und verbinde dich per SSH mit dem Raspberry Pi.

Nach der Anmeldung manövrierst du auf den Desktop und legst einen Projektordner an, zum *Beispiel so:*
***cd** Desktop und dann **mkdir** SCD41, gefolgt von **cd** SCD41*.

Danach erstellen wir mit dem Editor mit nano eine leere Python Datei namens ***SCD41_test.py***. In nano fügst du anschließend den kopierten Code mit SRG-V ein.
Speichere die Datei mit Strg+S, bestätige mit Enter und beende den Editor mit Strg+X.

Jetzt startest du das Skript mit dem Befehl: ***python SCD41_test.py***. Der Sensor wird initialisiert, wartet ein paar Sekunden und gibt anschließend in regelmäßigen Abständen die Werte für CO_2, Temperatur und Luftfeuchtigkeit im Terminal aus. Beenden kannst du das Skript jederzeit mit Strg+C.

2.4 Debugging-Tipps für den SCD41 (häufige Fehlerquellen)

Trotz korrekter Installation kann es vorkommen, dass keine Messwerte kommen oder der Sensor nicht gefunden wird.
Wurde das **I²C-Gerät erkannt?** Führe den Befehl **i2cdetect -y 1** aus, um alle Geräte auf Bus 1 aufzulisten. In der Ausgabe sollte an Adresse 0x62 ein Gerät angezeigt werden. Die Ausgabe auf der Konsole sollte dann so aussehen. Hier siehst du die 127 möglichen Adressen der Schnittstelle:

```
   0 1 2 3 4 5 6 7 8 9 A B C D E F
00:  -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- --
60: -- -- -- -- -- 62 -- --
70: -- -- -- -- -- -- -- –
```

Hier wird der Sensor bei 0x62 gefunden. Wenn nicht, überprüfe als erstes die Verkabelung (SDA/SCL vertauscht? Wackelkontakt?). Stelle auch sicher, dass I²C aktiviert ist (siehe oben) und der Sensor Strom bekommt.

Was kann alles Falsch laufen?

I²C aktiv/geladen? Falls i2cdetect gar nicht existiert oder kein /dev/i2c-1 vorhanden ist, wurde I²C evtl. nicht aktiviert. Aktiviere es mit raspi-config wie oben beschrieben und starte den Raspberry erneut.

Berechtigungen: Um i2cdetect oder I²C in Python zu nutzen, braucht dein Benutzer die richtigen Rechte. Wenn nicht, füge ihn hinzu (***sudo adduser pi i2c***). Bei Verwendung von RPi.GPIO direkt benötigt man häufig sudo zum Ausführen. Im Zweifel teste, ob sudo python3 sensor_scd41.py einen Unterschied macht.
Bibliotheken installiert? Falls Fehler wie "Module not found" aufkommen, prüfe, ob Adafruit Blinka und adafruit-circuitpython-scd4x wirklich installiert sind.
Adresse/Bus korrekt? Der SCD41 hat fest Adresse 0x62. Solltest du mehrere I²C-Geräte anschließen, achte auf Adresskonflikte. Am Raspberry Pi nutzen fast alle Sensoren Bus 1 (GPIO 2/3).
Fehlerhafte Messwerte: Wenn CO_2 konstant 0 oder sehr hohe Werte zeigt, könnte der Sensor ein Problem haben. Achte auch darauf, dass der Sensor nicht in einer geschlossenen Box ohne Luftaustausch misst, sonst steigen CO_2-Werte immer weiter. Für realistische Werte müsstest du gelegentlich lüften oder Sensor freier positionieren.
Hast du weiterhin Probleme, hilft dir oft ein Blick ins das offizielle Datenblatt. Meisten gibt es auch nützliche Foreneinträge in denen jemand ein ähnliches Problem / Verhalten beobachtet hat.

3. Aktoren: LEDs und Relais

Neben dem Sensor, der Daten liefert, wollen wir auch **Aktoren** ansteuern, hier exemplarisch **LEDs** (als einfache Signalausgabe) und ein **Relais**, um z.B. eine Heizmatte zu schalten. Aktoren ermöglichen dem Smart Home Dashboard, auf die gemessenen Umweltbedingungen zu reagieren (z.B. Warn-LED bei schlechter Luft, oder Heizung einschalten bei niedriger Temperatur).

3.1 Grundlagen – Funktionsweise von LEDs und Relais

LED

Eine LED ist ein Halbleiter-Bauteil, das aufleuchtet, wenn Strom in Durchlassrichtung fließt. Wichtig dabei ist die Polarität, die **Anode** (+, längeres Beinchen) muss positiv sein, die **Kathode** – (kürzeres Bein) negativ. Ansonsten kann der Strom nicht fließen da die Led eben nur in jene Richtung Strom leitet. Man schließt die Anode an einen GPIO-Ausgang und die Kathode an Masse an. **Wichtig:** Eine LED sollte nicht ohne Vorwiderstand angeschlossen werden.

Widerstand

Ein Widerstand (220 Ω) begrenzt den Strom durch die LED, damit sie nicht durchbrennt den GPIO nicht überlastet. Die LED leuchtet je nach Widerstand heller oder dunkler – zu groß gewählt bleibt sie aus, zu klein gewählt wird sie zerstört. Probiere ruhig verschiedene Widerstände aus?
Faustregel: 20Ω bis 220 Ω sind für erste Versuche gut.

Relais

Ein Relais ist ein elektrisch gesteuerter Schalter. Es besteht im Endeffekt aus einer Spule (Elektromagnet) und einem Schaltkontakt. Wenn Strom durch die Spule fließt, zieht sie einen „Anker“ an und schließt oder öffnet damit mechanisch einen Stromkreis. Je nach dem was für ein Relais und ei implementiert. Das Tolle ist, dass man mit einem kleinen Steuerstrom (vom Raspberry Pi, 3.3V) einen viel größeren Strom/Spannung schalten kann.

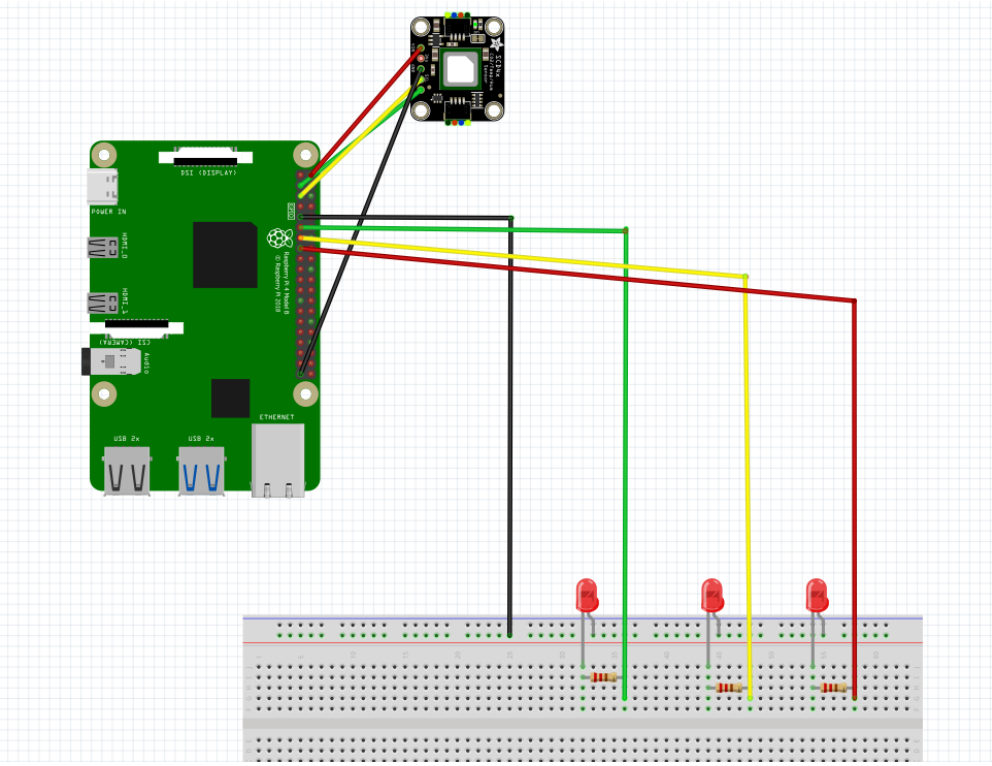
Wichtiger Sicherheitshinweis: Wenn du mit dem Relais Netzspannung (230V) schaltest (z.B. eine echte Heizmatte für Steckdosen), ist extreme Vorsicht geboten!!! Tue dies Niemals alleine. Lasse einen Fachmann die Verbindung herstellen. 230V sind lebensgefährlich
Berühre keine blanken Drähte und sorge für Isolierung!

Was brauchen wir?

Raspberry Pi mit freien GPIO-Pins und angeschlossener SCD41 Sensor (Part 1)

- 3xLED (Farbe nach Wahl)
- 3x Vorwiderstand (zwischen 20 Ω und 220 Ω)
- 5x Jumper-Kabel (z. B. rot für GPIO, schwarz für GND)
- 1 Breadboard oder direkt Steckverbindung zum Raspberry Pi

3.2 Schaltplan und Verkabelung mit dem Raspberry Pi



Wir verbinden die Anode der LEDs über den Wiederstand mit dem GPIO des PI's.
mit einem GPIO-in des Raspberry Pi.
Die LED Kathoden haben wir seriell an einen GND verbunden. Merke: LED-Kathode immer an Masse, LED-Anode an den GPIO-Ausgang, dazwischen ein Widerstand.

LED	GPIO-Pin	Pin am Raspberry Pi	Zusatz
Rot	GPIO 22	Pin 15	Über 220Ω-Widerstand mit GND verbinden
Gelb	GPIO 27	Pin 13	Über 220Ω-Widerstand mit GND verbinden
Grün	GPIO 17	Pin 11	Über 220Ω-Widerstand mit GND verbinden

3.3 Beispielcode zur Steuerung_LED

Link zum Code:
https://github.com/DIYSensorikWorkshop/DIY-Smart-Sensorik-Workshop/blob/main/Python%20Code%20Sensorik/Part2_led_control.py

A) LED blinken lassen

Erstelle die Datei `led_control.py` mit folgendem Inhalt:

```
# Pins für die drei LEDs (nach BCM-Nummern)
LED_ROT   = 22
LED_GRUEN = 27
LED_BLAU  = 17

# Grenzwerte, die ihr bei Bedarf ändern könnt
TEMP_UNTERGRENZE = 25.0  # unterhalb davon: kalt
TEMP_OBERGRENZE  = 26.0    # oberhalb davon: warm
CO2_SCHWELLENWERT_GUT = 1000  #ppm CO₂ Schwellenwert

# Benötigte Bausteine laden: Zeitfunktionen, Hardware-Pins, I²C-Bus, Sensor-Treiber
import time, board, busio, RPi.GPIO as GPIO
from adafruit_scd4x import SCD4X

# Pi-GPIOs vorbereiten und LED-Pins als Ausgänge setzen (starten aus = LOW)
GPIO.setmode(GPIO.BCM)
for p in (LED_ROT, LED_GRUEN, LED_BLAU):
    GPIO.setup(p, GPIO.OUT, initial=GPIO.LOW)

# Hilfsfunktion: LEDs gezielt ein- oder ausschalten
def leds(rot=0, gruen=0, blau=0):
    GPIO.output(LED_ROT,   rot)
```

```
    GPIO.output(LED_GRUEN, gruen)
    GPIO.output(LED_BLAU,  blau)

# Alle drei LEDs gemeinsam blinken lassen
# zyklen = wie oft, an/aus = Sekunden pro Phase
def blink_alle(zyklen=8, an=0.3, aus=0.3):
    for _ in range(zyklen):
        leds(1, 1, 1); time.sleep(an)
        leds(0, 0, 0); time.sleep(aus)

# Verbindung zum Sensor aufbauen
i2c = busio.I2C(board.SCL, board.SDA)
scd = SCD4X(i2c)
scd.start_periodic_measurement()         # Messung starten
print("SCD41 gestartet. Warte auf Messwerte...")
time.sleep(5)                             # Sensor braucht ein paar Sekunden

try:
    while True:                           # Endlosschleife, läuft bis ihr
stoppt
        if scd.data_ready:                # Nur lesen, wenn neue Daten
vorliegen
            co2 = scd.CO2
            temperatur = scd.temperature
            luftfeuchte = scd.relative_humidity

            # Zahlen im Terminal anzeigen
            print(f"CO₂: {co2} ppm | Temperatur: {temperatur:.1f} °C |
Luftfeuchte: {luftfeuchte:.1f} %")

            # 1) CO₂-Prüfung hat Priorität: zu hoch -> alle LEDs blinken
            if co2 is not None and co2 > CO2_SCHWELLENWERT_GUT:
                blink_alle()
                continue  # danach nächste Runde, Temperatur-LEDs werden
übersprungen

            # 2) Sonst Temperatur-Lampe setzen:
            if temperatur is not None and temperatur > TEMP_OBERGRENZE:
                leds(1, 0, 0)   # rot = zu warm
            elif temperatur is not None and temperatur >= TEMP_UNTERGRENZE:
                leds(0, 1, 0)   # grün = im Zielbereich
            elif temperatur is not None:
                leds(0, 0, 1)   # blau = zu kalt
            else:
                leds(0, 0, 0)   # keine Daten

        time.sleep(5)             # alle 5 Sekunden wieder prüfen
except KeyboardInterrupt:         # Strg+C gedrückt
    pass
finally:
```

```
    leds(0, 0, 0)                # LEDs aus
    try:
        scd.stop_periodic_measurement()  # Messung sauber stoppen
    except Exception:
        pass
    GPIO.cleanup()               # Pins freigeben
    print("Programm beendet.")
if __name__ == "__main__":
    main()
```

Was macht der Code?

Dieses Skript liest regelmäßig die Temperatur, die CO_2-Konzentration und die Luftfeuchtigkeit aus, indem es einen SCD41-Sensor verwendet. Die Messergebnisse werden dann über drei farbige LEDs visualisiert, die an die GPIO-Pins des Raspberry Pi angeschlossen sind. Ganz oben im Skript legst du fest, welche Pins für die LEDs verwendet werden und ab welchen Werten sie reagieren sollen. Du kannst also einstellen, ab wann es als zu kalt, angenehm oder zu warm gilt – sowie den CO_2-Schwellenwert, ab dem die Luftqualität als schlecht betrachtet wird.

Bevor es losgeht, richtet das Skript die nötigen GPIO-Pins ein. Die LEDs werden dabei auf einen sicheren Ausgangszustand gesetzt, also ausgeschaltet. Danach wird die Verbindung zum Sensor über den I2C-Bus aufgebaut und die Messung gestartet. Der Sensor braucht ein paar Sekunden, bevor er verlässliche Daten liefert.

Sobald alles läuft, beginnt eine Endlosschleife. Das Skript prüft alle fünf Sekunden, ob neue Sensordaten verfügbar sind. Wenn der CO_2-Wert zu hoch ist, blinken alle LEDs kurz – so bekommst du ein deutliches visuelles Signal, dass die Luftqualität nicht gut ist. In diesem Fall werden die Temperatur-LEDs übersprungen, weil die CO_2-Warnung Vorrang hat.

Wenn der CO_2-Wert im grünen Bereich liegt, wird anhand der Temperatur eine der drei LEDs eingeschaltet: Rot bedeutet zu warm, grün ist optimal, blau steht für kühlere Bedingungen. Gibt es keine brauchbaren Temperaturdaten, bleiben alle LEDs aus.

Beenden kannst du das Skript jederzeit mit Strg+C. Danach werden die LEDs ausgeschaltet, der Sensor sauber gestoppt und alle verwendeten Pins wieder freigegeben. Damit lässt sich das System unkompliziert herunterfahren, ohne dass etwas hängen bleibt.

Steuerung einer Heizmatte mit einem Relaismodul und dem Raspberry Pi

Für die Heizungssteuerung erweitern wir den bisherigen Aufbau um ein Relais und die Lastseite. Du verkabelst erst Pi↔Relais, dann Relais↔Heizmatte. Du kannst das Relais mit dem kleinen Skript kurz testen oder gleich den Code für die fertige Heizungssteuerung nutzen.

Was brauchen wir?

- Relaismodul, 1-Kanal (VCC 5 V, Eingang 3,3 V-tauglich)
- Heizmatte passend zur Betriebsspannung
- Netzgerät für die Heizmatte (z. B. 12 V, ausreichend Ampere)
- Klingeldraht
- Jumper-Kabel Pi↔Relais
- Breadboard oder Schraubklemmen
- Optional: Sicherung in Serie, Schrumpfschlauch/Isoliermaterial

Anschluss des Relaismoduls an den Raspberry Pi

IN: Verbinde den Pin mit einem freien GPIO-Pin des Raspberry Pi, beispielsweise GPIO 23
VCC: Schließen diesen Pin an die 3V Stromversorgung des Raspberry Pi an.
GND: Verbinden den Pin mit einem GND-Pin des Raspberry Pi. (Du kannst auch mehre GND Anschlüsse auf dem Breadboard zusammenlegen um Platz zu sparen)

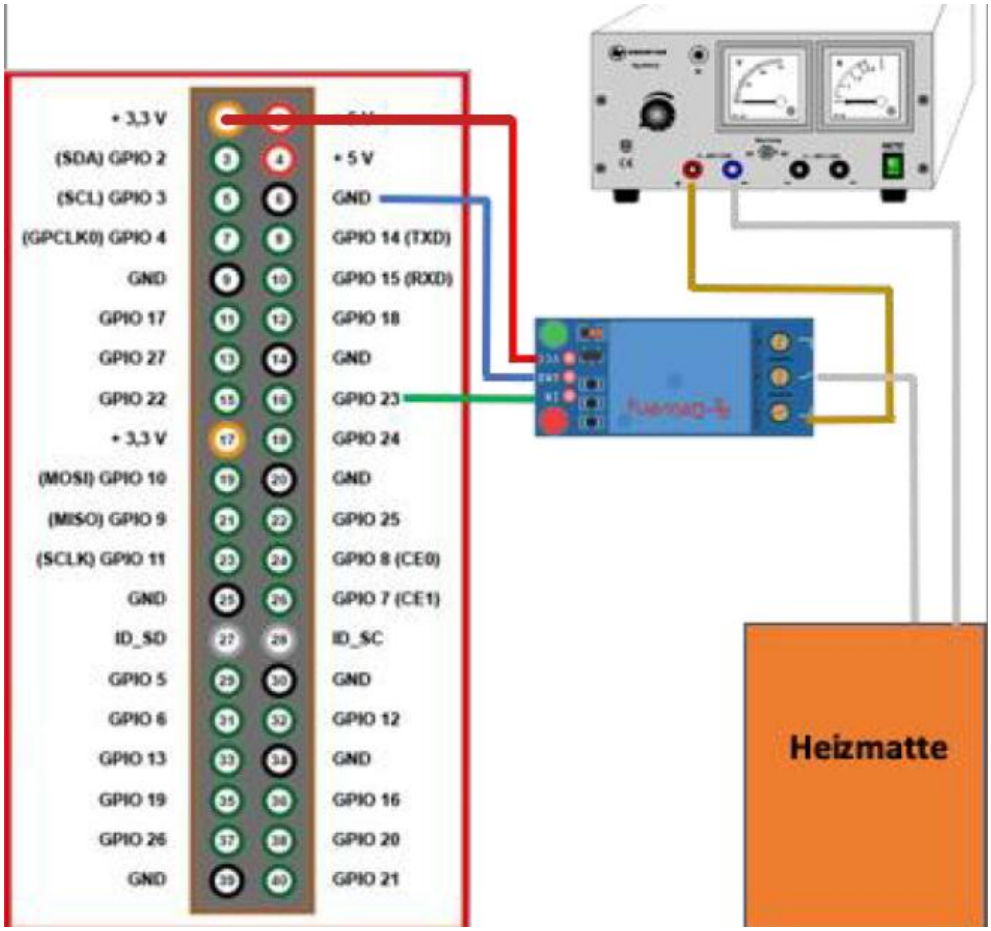
Hinweis: Viele Relaismodule benötigen 3,3V als Versorgungsspannung. Achten Sie darauf, den VCC-Pin entsprechend anzuschließen, da einige Module auch mit 5V laufen, aber dann nicht zuverlässig schalten.

Anschluss der Heizmatte an das Relaismodul

COM (Common): Verbinden den Anschluss mit einem der beiden Anschlüsse der Heizmatte.
NO (Normally Open): Schließen den Anschluss an den Minuspol der 12V-Stromquelle an.
Pluspol der 12V-Stromquelle: Verbinden mit dem freien Anschluss der Heizmatte

In diesem Aufbau fließt im Ruhezustand kein Strom durch die Heizmatte, da der Stromkreis am Relais unterbrochen ist. Erst wenn das Relais aktiviert ist, schließt sich der Stromkreis zwischen COM und NO, und die Heizmatte wird mit Strom versorgt.

Schaltplan:



So wird das Relais geschaltet:

```python
def set_heater_control(temp):
    """
    Steuert das Relais für die Heizung:
    - Wenn die Temperatur über 25°C liegt, wird die Heizung ausgeschaltet.
    - Wenn die Temperatur 25°C oder darunter liegt, wird die Heizung eingeschaltet.
    """
    if temp > TEMP_MEDIUM_MAX:
        GPIO.output(RELAY_PIN, GPIO.LOW)  # Relais aus → Heizung aus
        print("Relais AUS (Heizung aus)")
    else:
        GPIO.output(RELAY_PIN, GPIO.HIGH)  # Relais an → Heizung an
        print("Relais AN (Heizung an)")

def main():
    """Simuliert Temperaturwerte und steuert LEDs & Relais entsprechend"""
    setup_gpio()

    try:
        while True:
            temp = float(input("Gib eine Temperatur ein: "))  # Temperatur manuell eingeben
            set_temperature_leds(temp)  # LEDs basierend auf Temperatur aktualisieren
            set_heater_control(temp)  # Relais basierend auf Temperatur steuern
    except KeyboardInterrupt:
        print("\nProgramm beendet.")
    finally:
        GPIO.cleanup()  # Setzt alle GPIO-Pins zurück

if __name__ == "__main__":
    main()
```

Was passiert hier?
Im obigen Code gehen wir von active-high aus (HIGH bedeutet Relais ist an). Sollte sich nichts tun, versuche das Gegenteil (LOW für an, HIGH für aus). Du kannst zum Test auch einfach Passe den Code entsprechend an oder kommentiere die Prints um (z.B. „Heizmatte AN" beim passenden Signal).
Nun kann das Programm starten. Das Relais sollte nun alle 5 Sekunden schalten und du solltest ein leises Klicken hören. Wenn an der Ausgangsseite eine Heizmatte angeschlossen ist, würde diese im selben Rhythmus ein- und ausgehen. In echter Anwendung würde man natürlich nicht stumpf im Intervall schalten, sondern z.B. aufgrund eines Messwertes. Du könntest hier anstelle des starren Zeitintervalls z.B. den CO_2-Wert aus dem Sensor lesen und abhängig davon schalten (z.B. Lüfter an, wenn CO_2 > 1000 ppm). Dies kann man im main.py dann kombinieren. Unser Ziel war es aber zunächst, das manuelle Ansteuern *zu demonstrieren.*
Hinweis: *Vergiss nicht, am Ende GPIO.cleanup() aufzurufen oder das Relais explizit wieder auszuschalten, damit es nicht in ungewolltem Zustand stehen bleibt, wenn das Programm endet.*

Code Heizungssteuerung Komplett

Link zum Code:
https://github.com/DIYSensorikWorkshop/DIY-Smart-Sensorik-Workshop/blob/main/Python%20Code%20Sensorik/Part3_heat_control.py

```
import time
import board
import busio
import RPi.GPIO as GPIO
from adafruit_

# GPIO-Nummern (BCM-Layout)
LED_COOL = 17   # Kühl -> grüne LED
LED_MEDIUM = 27 # Mittel -> gelbe LED
LED_WARM = 22   # Warm -> rote LED
RELAY_PIN = 23  # Steuert das Relais (geändert!)


# Schwellwerte in °C
TEMP_COOL_MAX = 20.0   # Bis 20°C: kühl
TEMP_MEDIUM_MAX = 25.0 # Bis 25°C: mittel
# Ab 25°C: warm

def setup_gpio():
    """GPIOs initialisieren."""
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(LED_COOL, GPIO.OUT)
    GPIO.setup(LED_MEDIUM, GPIO.OUT)
    GPIO.setup(LED_WARM, GPIO.OUT)
    GPIO.setup(RELAY_PIN, GPIO.OUT)

    # Zu Beginn alles ausschalten
    GPIO.output(LED_COOL, GPIO.LOW)
    GPIO.output(LED_MEDIUM, GPIO.LOW)
    GPIO.output(LED_WARM, GPIO.LOW)
    GPIO.output(RELAY_PIN, GPIO.LOW)  # Relais aus
```

```python
def set_temperature_leds(temp):
    """Setzt die LEDs je nach Temperatur."""
    print(f"Aktuelle Temperatur: {temp} °C")  # Debug-Ausgabe
    if temp <= TEMP_COOL_MAX:
        # Kühl
        GPIO.output(LED_COOL, GPIO.HIGH)
        GPIO.output(LED_MEDIUM, GPIO.LOW)
        GPIO.output(LED_WARM, GPIO.LOW)
    elif temp <= TEMP_MEDIUM_MAX:
        # Mittel
        GPIO.output(LED_COOL, GPIO.LOW)
        GPIO.output(LED_MEDIUM, GPIO.HIGH)
        GPIO.output(LED_WARM, GPIO.LOW)
    else:
        # Warm
        GPIO.output(LED_COOL, GPIO.LOW)
        GPIO.output(LED_MEDIUM, GPIO.LOW)
        GPIO.output(LED_WARM, GPIO.HIGH)

def set_heater_control(temp):
    """
    Steuert das Relais für die Heizung:
    - Bei warm (Temp > TEMP_MEDIUM_MAX) -> Relais aus (Heizung aus).
    - Sonst Relais an (Heizung an).
    """
    if temp > TEMP_MEDIUM_MAX:
        # Warm -> Relais aus
        GPIO.output(RELAY_PIN, GPIO.LOW)
        print("Relais AUS (Heizung aus)")
    else:
        # Kühl oder Mittel -> Relais an
        GPIO.output(RELAY_PIN, GPIO.HIGH)
        print("Relais AN (Heizung an)")

def main():
    setup_gpio()


    # I2C-Initiierung
    i2c = busio.I2C(board.SCL, board.SDA, frequency=100000)
    scd4x = SCD4X(i2c)
    scd4x.start_periodic_measurement()
    print("SCD4x-Sensor gestartet. Warte auf erste Messwerte...")

    try:
        while True:
            if scd4x.data_ready:
                co2 = scd4x.CO2
                temperature = scd4x.temperature
                humidity = scd4x.relative_humidity

                # Ausgabe der Sensordaten im Terminal
                print(f"CO2: {co2:.1f} ppm, "
                      f"Temp: {temperature:.2f} °C, "
                      f"Feuchte: {humidity:.2f} %")

                # LEDs aktualisieren
                set_temperature_leds(temperature)

                # Relais steuern
                set_heater_control(temperature)
```

```
        # Alle 5 Sekunden erneut messen
        time.sleep(5)

    except KeyboardInterrupt:
        print("Beende Programm...")
    finally:
        GPIO.cleanup()
        client.close()

if __name__ == "__main__":
    main()
```

Was macht der Code?

Der Code liest regelmäßig die Temperatur und den CO_2-Wert vom SCD41-Sensor aus und steuert damit drei LEDs sowie ein Relais. Ganz oben im Skript legst du die wichtigsten Einstellungen fest: Welche GPIO-Pins für die roten, grünen und blauen LEDs verwendet werden, welcher Pin das Relais schaltet, ab welchen Temperaturwerten es als "kalt", "okay" oder "warm" gilt und ab welchem CO_2-Wert alle LEDs kurz blinken sollen. Dort bestimmst du auch, ob das Relais bei einem LOW- oder HIGH-Signal einschalten soll.

Anschließend richtet die Funktion gpio_setup() die Pins ein und stellt sicher, dass zu Beginn alles ausgeschaltet ist. Mit scd41_setup() wird der Sensor gestartet. In der Hauptschleife holt sich das Programm regelmäßig neue Messwerte. Ist der CO_2-Wert zu hoch, blinken alle LEDs kurz als Hinweis. Andernfalls zeigt die Farbe der LEDs die Temperatur an: Blau bedeutet kühl, Grün steht für einen mittleren Bereich und Rot signalisiert Wärme.

Gleichzeitig läuft eine einfache Heizungslogik mit: Ist es wärmer als der eingestellte Grenzwert, bleibt die Heizung aus. Wird es kühler, schaltet sie sich ein. Wenn bei deiner Hardware eine LED andersherum reagiert oder das Relais genau entgegengesetzt arbeitet, musst du nur die Werte oben im Skript anpassen und das Programm neu starten. So erhältst du auch ohne tiefere Python-Kenntnisse schnell ein sichtbares Feedback und kannst die Grenzwerte sowie die verwendeten Pins unkompliziert an deine Hardware anpassen.

Starte wie in den vorherigen Teilen:

Öffne auf dem Raspberry Pi ein Terminal, wechsle in deinen Projektordner und leg die Datei an. Mit nano heating_control.py öffnest du den Editor, fügst den Code mit Strg+Shift+V ein, speicherst mit Strg+O und Enter und schließt nano mit Strg+X. Stelle sicher, dass die Abhängigkeiten installiert sind und I^2C aktiv ist. Dann startest du das Programm mit python3 heating_control.py. Beenden kannst du es jederzeit mit Strg+C. Beim Beenden setzt das Programm die GPIO Pins zurück und schaltet das Relais aus.

Nach dem Start siehst du im Terminal laufend Messwerte für CO_2, Temperatur und Luftfeuchte sowie Meldungen zum Schaltzustand. Das Relais klickt hörbar, wenn es umschaltet. Die Heizmatte geht an, wenn die Temperatur unter dem Grenzwert liegt, und aus, wenn sie darüber liegt. Die LEDs zeigen den jeweiligen Temperaturbereich an, sodass du den Zustand ohne Blick ins Terminal einschätzen kannst.

Wenn das Relais genau umgekehrt reagiert, passt du die Einstellung für den Logikpegel an und startest erneut. Achte bei Änderungen an der Verdrahtung darauf, das Netzteil der Heizmatte vorher zu trennen. Für einen ersten Test genügt auch der Blick auf die LED am Relaismodul, die das Anziehen signalisiert.

3.4 Debugging und Fehlervermeidung bei LEDs und Relais

1. ***LED leuchtet nicht:*** *Überprüfe die Polung der LED. Das lange Bein (Anode) muss zum GPIO (über Widerstand), das kurze Bein zur Masse*

   Ohne richtigen Anschluss bleibt sie aus. Prüfe auch den Widerstandswert – ist er viel zu hoch (z.B. 1 MΩ statt 220 Ω), fließt kaum Strom. Ist er zu niedrig (z.B. 0 Ω), überlastest du den GPIO. Standard-LEDs zeigen bei 220 Ω einen gut sichtbaren Glanz.

2. ***GPIO-Nummerierung verwechselt:*** *Raspberry Pi hat zwei gebräuchliche Pin-Nummerierungen –* ***physikalische Pin-Nummern*** *(BOARD) und* ***GPIO BCM-Nummern****. Achte darauf die Richtigen zu verwenden*

3. ***Relais klickt nicht:*** *Prüfe die Verkabelung des Relaismoduls. Hängt* ***GND des Moduls an GND des Pi****? Ohne gemeinsamen Massebezug funktioniert das Steuersignal nicht. Ist VCC an 3,3V? (Manche Module laufen auch an 5V, dann entsprechend anschließen.)*

4. ***Relais-Logikpegel:*** *Wie oben erwähnt, viele 3V-Relaismodule sind active-low, da sie optisch entkoppelt sind: Das bedeutet, du musst GPIO.LOW ausgeben, um das Relais* ***einzuschalten****, und GPIO.HIGH für aus. Wenn dein Relais genau invers zu deinem Code schaltet, invertiere die Ausgaben oder initialisiere den PIN anders (manche ziehen den Eingang intern hoch). Beobachte die LED auf dem Relaismodul (falls vorhanden): leuchtet sie, wenn du den Pin auf LOW setzt? Dann ist es active-low (bei manchen steht auch INVERTED in der Beschreibung). Unser Code oben kann leicht angepasst werden – wichtig ist nur, es zu verstehen.*

5. ***Heizmatte wird nicht warm:*** *Wenn das Relais zwar klickt, aber die Heizmatte nicht reagiert, liegt das Problem auf der Lastseite. Prüfe die Verkabelung der Heizmatte mit COM/NO am Relais. Ist der Stromkreis wirklich geschlossen, wenn das Relais anzieht? Messe die Spannung an der Matte. Ggf. wurde NC statt NO verwendet – dann würde die Matte genau andersherum arbeiten (aus, wenn Relais an). Schließe im Zweifel die Matte testweise direkt an die Versorgung an, um zu sehen ob sie grundsätzlich funktioniert.*

6. ***Sauber abschalten:*** *Bei Programmen, die Endlosschleifen mit GPIO nutzen, immer eine Abbruchbedingung (try/except KeyboardInterrupt) einbauen und am Ende GPIO.cleanup() aufrufen. Sonst kann es passieren, dass der GPIO-Pin in undefiniertem Zustand bleibt, was beim Relais z.B. bedeutet, dass es angezogen bleibt, obwohl das Programm schon beendet ist.*

4.Lösungen zu Wissen- und Diskussions/-aufgaben

Aufgabe 1: Lückentext – trage die richtigen Begriffe ein (SDA, SCL, 3.3V, GND)

1. *Pin 3 am Raspberry Pi ist SDA*
2. *Pin 5 am Raspberry Pi ist SCL.*
3. *Pin 1 am Raspberry Pi liefert 3.3V.*
4. *Pin 6 am Raspberry Pi ist GND.*

*Hinweis: Schüler*innen sollten ein Pinout zur Hilfe nutzen dürfen. Ziel ist das Verstehen von I²C und der Stromversorgung.*

Aufgabe 2: Welche Sicherheitsaspekte sind beim SSH-Zugriff wichtig?

- *Standard-Passwort sollte direkt geändert werden.*
- *SSH-Zugriffe können durch Firewalls oder Fail2Ban abgesichert werden.*

Nur bekannte Geräte/IP-Adressen zulassen (z. B. durch Key-basierte Authentifizierung).

Aufgabe 3: Welche Vorteile hat der „Headless“-Betrieb (also ohne Monitor und Tastatur) über SSH?

Vorteile

- *Kein Monitor/Tastatur nötig → spart Platz & Kosten.*
- *Fernzugriff von überall im Netzwerk.*

Ideal für Automatisierungen/Projekte.

Aufgabe 4: Welche Nachteile oder Risiken können dadurch entstehen?

Nachteile/Risiken

- *Kein direkter Zugriff bei Netzwerkfehlern.*
- *Höhere Hürde bei der Erstinstallation.*
- *Sicherheitsrisiken bei offenem SSH.*

Aufgabe 5: Erkläre in drei Sätzen, warum ein CO_2-Sensor in einem Klassenzimmer oder Jugendzimmer sinnvoll sein kann. Überlege, welche Folgen es hat, wenn der CO_2-Wert zu hoch ist.

Ein CO_2-Sensor im Klassenraum ist sinnvoll, da hohe CO_2-Werte zu Konzentrationsproblemen führen. Wenn der CO_2-Wert steigt, ist Lüften nötig. Das hilft, frische Luft hereinzulassen und das Wohlbefinden zu verbessern.

Aufgabe 6 Nenne mindestens zwei typische Fehlerquellen, wenn der Sensor keine Daten liefert, obwohl das Skript korrekt ist. Was kannst du in diesen Fällen überprüfen oder verändern?

Typische Fehlerquellen

- *Falsche Verkabelung (SDA/SCL vertauscht).*
- *I^2C-Schnittstelle nicht aktiviert.*
- *Sensor nicht korrekt erkannt (falsche Adresse).*
- *Fehlende/fehlerhafte Bibliotheken.*

Aufgabe 7: Diskussionsaufgabe

Weitere Luftwerte messen?

- *CO_2 ist ein guter Indikator für „verbrauchte“ Luft.*

Aber: VOC (flüchtige organische Verbindungen) oder Feinstaub geben genauere Infos über Luftqualität.

Technik als Hilfe

- *Sensorik schafft Bewusstsein für Luftqualität.*
- *Unterstützt gesundes Verhalten (z. B. Lüften).*

Technik ersetzt nicht Eigenverantwortung

Grenzen

- *Kosten, Wartungsaufwand.*
- *Datenschutz bei Überwachungssystemen.*

Aufgabe 8

- *D Begrenzt den Strom durch eine LED, damit sie nicht durchbrennt.*
- *B Die Anschlussseite einer LED, die positiv gepolt werden muss.*
- *C Begriff für eine Schaltung, bei der ein niedriger Pegel (0 V) ein Signal aktiviert.*
- *E Der gemeinsame Anschluss eines Relais, an den die Last angeschlossen wird.*
- *A Der Kontakt am Relais, der normalerweise offen ist und sich erst beim Einschalten schließt.*
-

Aufgabe 9 -Wissensfrage Modul 3: Erkläre den Unterschied zwischen „active-high“ und „active-low“ Relaismodulen. Warum kann es sein, dass dein Code umgekehrt reagieren muss?

- *Active-high → Relais schaltet bei HIGH-Signal (3.3V).*
- *Active-low → Relais schaltet bei LOW-Signal (0V).*

*Aufgabe 10 - Diskutiere mit deinen Mitschüler*innen:*

- *Wo siehst du Vorteile darin, alltägliche Geräte (Heizung, Licht, Lüfter) automatisiert zu schalten?*
- *Gibt es auch ethische oder praktische Bedenken, wenn Maschinen für uns Entscheidungen treffen (z.B.„Heizung an/aus“)?*
- *Sollte man manche Dinge lieber manuell regeln?*

Vorteile der Automatisierung

- *Energieeffizienz (z. B. Heizung bei Fenster offen → aus).*
- *Komfortsteigerung.*
- *Zeitersparnis.*

Ethik und Bedenken

- *Kontrollverlust („Black Box“-Entscheidungen).*
- *Technik kann falsche Entscheidungen treffen.*

Glossar – Abkürzungen & Begriffe

Abkürzung	***Erklärung***
GPIO	General Purpose Input/Output – Universelle Ein-/Ausgangspins am Raspberry Pi zur Ansteuerung von Sensoren und Aktoren.
I^2C	Inter-Integrated Circuit – Ein serielles Busprotokoll, mit dem mehrere Geräte über zwei Leitungen (SDA, SCL) angeschlossen werden können.
SDA	Serial Data Line Datenleitung des I^2C-Busses.
SCL	Serial Clock Line Taktleitung des I^2C-Busses.
CO_2	Kohlenstoffdioxid Ein farb- und geruchloses Gas, das als Luftqualitätsindikator verwendet wird.
ppm	Parts per Million Maßeinheit für Konzentrationen, z. B. CO_2 in der Luft. 1000 ppm = 0,1 % Volumenanteil.
SSH	Secure Shell Ein sicheres Netzwerkprotokoll zur Fernsteuerung von Computern über ein Terminal.
BCM	Broadcom Chip Model Bezeichnung der internen Pin-Nummerierung des Raspberry Pi-Chips.
VCC	Versorgungsspannung
GND	Ground Masseanschluss, Bezugspunkt für elektrische Spannungen.
Relais	Elektronischer Schalter, der mit einem kleinen Steuerstrom welcher einen größeren Stromkreis schalten kann.

Quellenverzeichniss

1. ***Adafruit Industries, Inc.***
   Adafruit CircuitPython SCD4x Library
   GitHub Repo https://github.com/adafruit/Adafruit_CircuitPython_SCD4x
2. ***Adafruit Industries, Inc.***
   CircuitPython on Raspberry Pi – *Adafruit Learn Guide*
   https://learn.adafruit.com/circuitpython-on-raspberrypi-linux
3. ***Raspberry Pi Foundation.***
   Raspberry Pi Documentation
   https://www.raspberrypi.org/documentation/

4. ***Elektronik-Kompendium.***
   Online-Ressource zu elektronischen Grundlagen und Schaltungen
   https://www.elektronik-kompendium.de