

LED-Matrix Workshop



<https://github.com/InES-HPMM/LED-Matrix-Workshop>

Inhaltsverzeichnis

1. Hardware	2
1.1. Aufbau	2
1.2. Leiterplatte (PCB)	3
1.3. Microcontroller: RaspberryPi Pico	4
1.4. Verbindungen zwischen RaspberryPi Pico und PCB	5
2. Programmieren	6
2.1. MicroPython auf RaspberryPi Pico installieren	6
2.2. Thonny installieren	7
2.3. Erstes Programm	8
2.4. Programm auf RaspberryPi Pico speichern	9
2.5. Python Grundlagen	10
2.6. Python Library von ZHAW	16
A. Anhang	19
A.1. PCB Schematic	19

1. Hardware

1.1. Aufbau

Die LED-Matrix besteht aus einer Reihe von Komponenten. Diese werden aufeinander gelegt und mit vier Schrauben befestigt. Folgende Komponenten werden benötigt:

- Diffusorplatte (aus Plexiglas)
- Ziffernblatt (scharzer Karton mit gelaserten Buchstaben)
- Gitter (3D gedruckt)
- Leiterplatte
- RaspberryPi Pico Microcontroller-Board

Das Ziffernblatt wird benutzt, um die LED-Matrix in eine **Word Clock** zu verwandeln. Im LED-Matrix Modus wird dieses weggelassen.

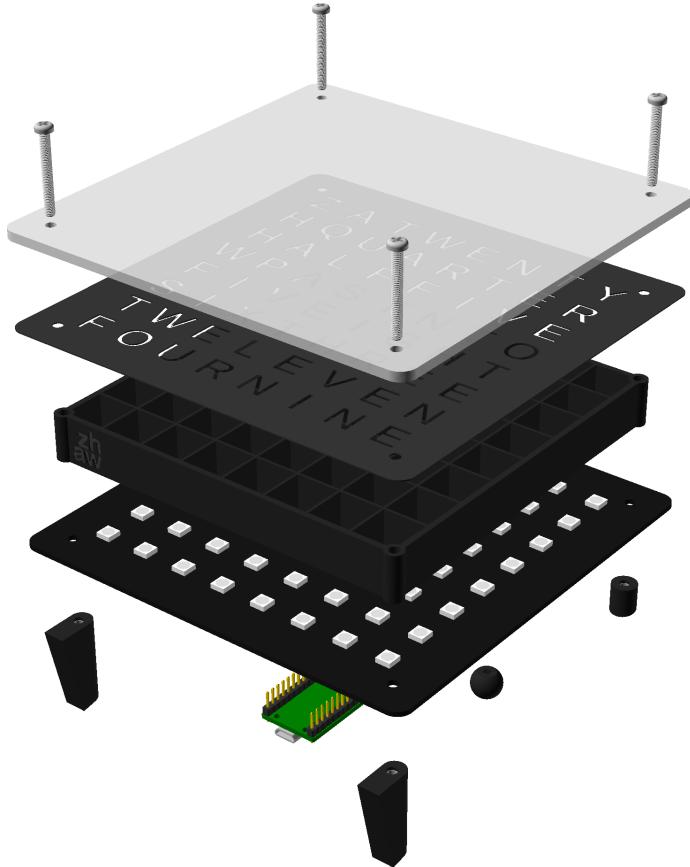


Abbildung 1.1.: Aufbau

1.2. Leiterplatte (PCB)

Die Leiterplatte wird im Englischen **Printed Circuit Board** (PCB) genannt und ist ein Träger für elektronische Bauteile. Auf der Vorderseite des LED-Matrix PCB befinden sich 64 RGB LED's. Auf der Rückseite hat es einen Joystick-Knopf, einen Schiebeschalter sowie einen Verbindungsstecker für das Microcontroller-Board.

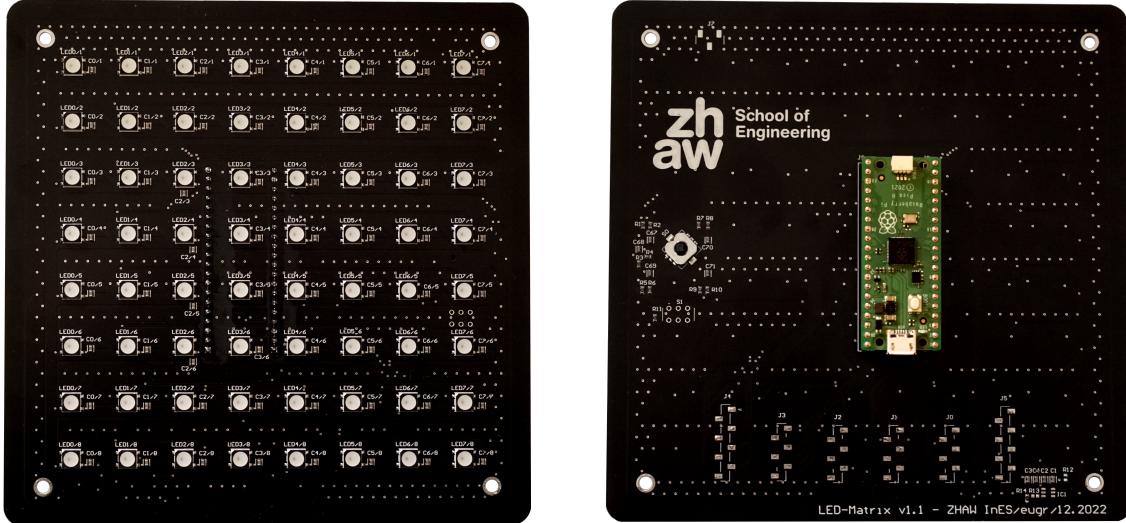


Abbildung 1.2.: PCB Vorder- und Rückseite mit RaspberryPi Pico

Das verwendete Microcontroller-Board ist der **Raspberry Pi Pico**, welcher bei den Steckern J8/J9 eingesetzt wird. Es ist wichtig zu beachten, dass man den Pico beim Einsticken so dreht, dass der Micro-USB Port nach unten zeigt. Um mit dem Pico die LEDs zum Leuchten zu bringen und die Schalterpositionen einlesen zu können, muss man wissen wie die Komponenten auf dem PCB an das Microcontroller-Board angeschlossen sind. Diese Information bekommt man aus dem Schaltplan (Schematics). Die Schematics des PCBs sind im Anhang A.1 ab Seite 19 zu finden.

1.2.1. Neopixel - RGB LED

LED steht für **light-emitting diode** und heisst auf Deutsch **Leuchtdiode**. Neopixel sind addressierbare rot-grün-blau (RGB) LEDs. Dies bedeutet, dass man die rote, grüne sowie blaue LED jedes Neopixel-LED einzeln ansprechen bzw. ansteuern kann. Dafür hat es im Inneren des Neopixel einen eigenen Controller. Durch unterschiedlich helles leuchten dieser drei Grundfarben können fast alle anderen Farben gemischt werden.

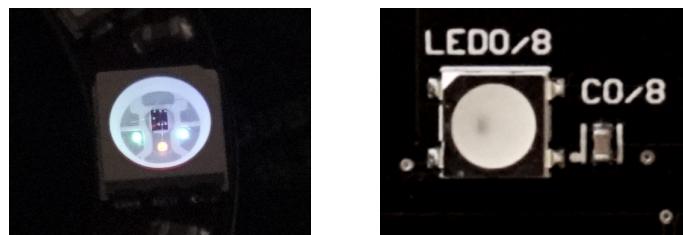


Abbildung 1.3.: Neopixel - Addressierbare RGB LEDs

Die Neopixel sind alle hintereinander gehängt. Die Ansteuerung läuft über ein Steuersignal. Das erste Neopixel liest die ersten RGB Werte aus dem Signal und leitet das restliche Steuersignal weiter an das nächste Neopixel. Somit kann für jede der 64 RGB LEDs ein individueller Farbwert eingestellt werden.

1.2.2. Joystick und Schiebeschalter

Beim Joystick lassen sich fünf verschiedene Bewegungen erkennen. Der Joystick kann nach oben, unten, links, rechts bewegt oder in die Mitte gedrückt werden. Auf dem PCB hat es ebenfalls einen Schiebeschalter mit zwei möglichen Einstellungen. Beide Komponenten sind im Schema auf Seite 22 zu finden.

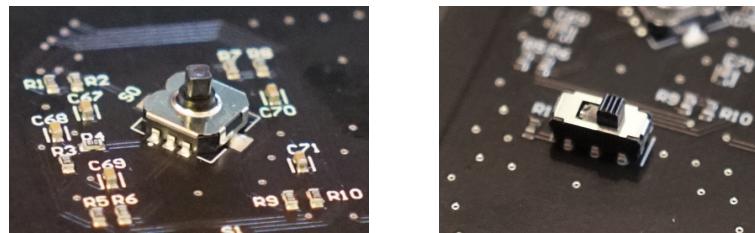


Abbildung 1.4.: Joystick (links) und Schiebeschalter (rechts)

1.3. Microcontroller: RaspberryPi Pico

Der RaspberryPi Pico ist ein kostengünstiges Mikrocontroller-Board mit 26 digitalen Schnittstellen. Des Weiteren verfügt der Pico über SPI-, UART-, I2C-Schnittstellen und einen Analog-Digital Wandler. Für den Betrieb der LED Matrix werden diese weiteren Features jedoch nicht benötigt. Die folgende Abbildung zeigt das Pin-Layout.

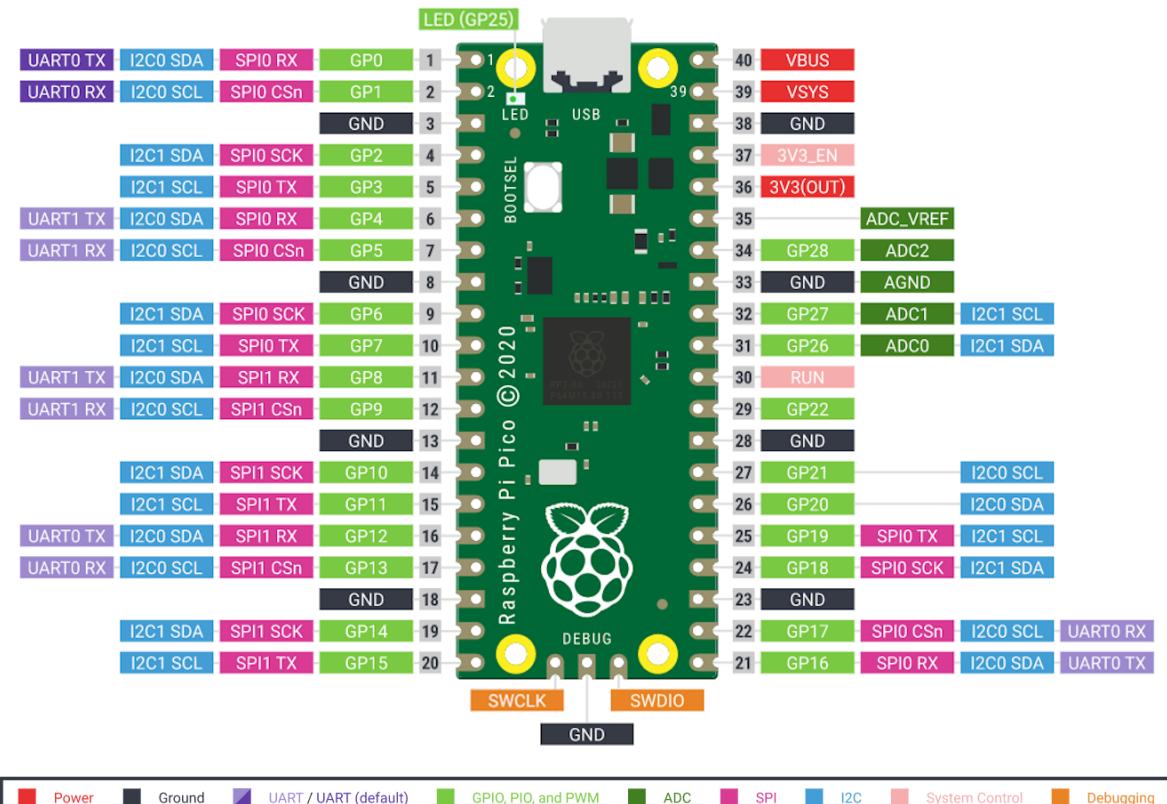


Abbildung 1.5.: RaspberryPi Pico - Pin Layout (Quelle: projects.raspberrypi.org)

Die hellgrünen Label mit der Bezeichnung GP0-GP28 sind GPIOs (**general-purpose input/output**). Übersetzt ins Deutsche heisst das **Allzweckeingabe/-ausgabe**. Bei jedem GPIO kann man wählen, ob es als Input oder Output verwendet werden soll. GPIOs welche mit einem Schalter verbunden sind, benutzt man als Input. Denn man möchte den Zustand des Schalters einlesen. Für das

Ansteuern einer LED benutzt man den GPIO als Output, weil man möchte die LED steuern und nicht einlesen.

1.4. Verbindungen zwischen RaspberryPi Pico und PCB

Aus dem Schema im Anhang und der Abbildung 1.5 auf Seite 4 kann man herausfinden, wie die Hardwarekomponenten am Pico angeschlossen sind.

- Schiebeschalter: (slide switch) SL_SW ==> GP____
- Joystick: (4-way switch) 4WSW_A ==> GP____
- 4WSW_B ==> GP____
- 4WSW_C ==> GP____
- 4WSW_D ==> GP____
- 4WSW_CENTER ==> GP____
- Neopixel LED SPI0_TX_LED ==> GP____

2. Programmieren

2.1. MicroPython auf RaspberryPi Pico installieren

MicroPython ist eine vollständige Implementation der Programmiersprache Python 3, welche direkt auf dem RaspberryPi Pico läuft. Um MicroPython auf dem Pico zu installieren, muss die passende MicroPython-Installationsdatei (Dateiformat: UF2) auf den Pico kopiert werden. Die Datei kann unter folgendem Link heruntergeladen werden:

<https://www.raspberrypi.com/documentation/microcontrollers/micropython.html>

Es gibt zwei verschiedene Dateien zum Herunterladen, weil es zwei verschiedene Versionen des RaspberryPi Pico gibt. Man muss daher wissen, welchen Pico man verwendet.

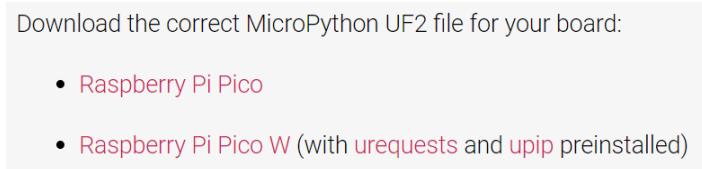


Abbildung 2.1.: MicroPython-Installationsdatei herunterladen

Der *Raspberry Pi Pico W* ist die Version des Picos mit einem WLAN Modul. Die beiden Versionen lassen sich einfach unterscheiden. Auf dem Pico W befindet sich ein silbernes Wifi-Modul:

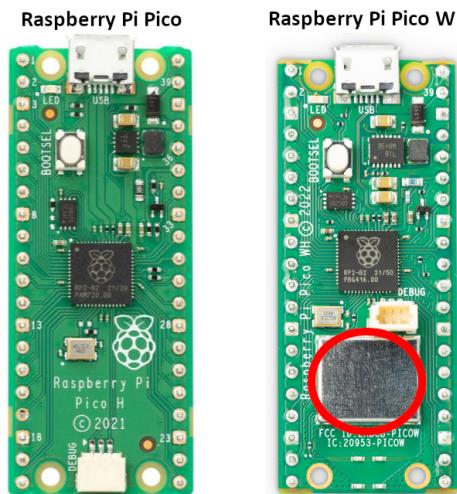


Abbildung 2.2.: Raspberry Pi Pico Versionen (Quelle: <https://github.com/raspberrypi/>)

(Beide Versionen des Picos können mit oder ohne gelötete Steckleisten bestellt werden. Die Ausführung mit Steckleiste hat noch den Buchstaben H angehängt.)

Nach dem Herunterladen der Installations-Datei muss man diese auf den Pico kopieren. Verbinde dazu den RaspberryPi Pico über ein Micro-USB Kabel mit dem Computer. **Während dem Einstecken muss der weisse Knopf `BOOTSEL` gedrückt werden.** Nach erfolgreichem Verbinden erscheint auf dem Computer ein Laufwerk mit dem Namen **RPI-RP2**. Kopiere die heruntergeladene MicroPython-Installationsdatei (Dateiformat: UF2) auf dieses Laufwerk. MicroPython wird anschliessend automatisch auf dem Pico installiert und das Laufwerk verschwindet wieder. Der RaspberryPi Pico ist somit bereit um Programmiert zu werden.

2.2. Thonny installieren

Thonny ist ein Python Editor und eignet sich sehr gut für das Programmieren des RaspberryPi Pico. Unter dem folgenden Link kann Thonny heruntergeladen werden:

<https://thonny.org/>

Starte das Programm nach der Installation. Die Oberfläche von Thonny ist aufgeteilt in zwei Bereiche. Der obere Bereich ist der Texteditor für den Programmiercode. Im unteren Bereich hat es eine *Shell*. Darin kann man direkt Python-Kommandos ausführen sowie mit dem Pico Kommunizieren. Die Shell kann Kommandos an den Pico übermitteln sowie Nachrichten vom Pico für uns anzeigen.

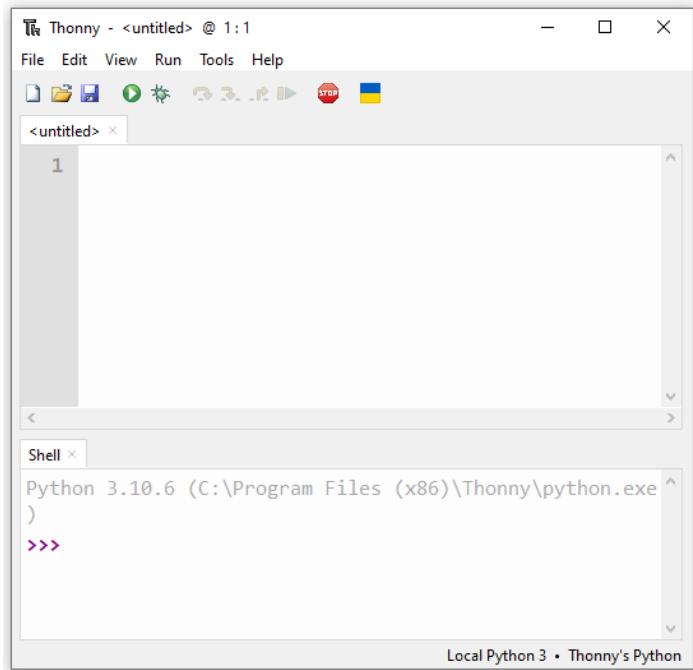


Abbildung 2.3.: Thonny Oberfläche

Bevor mit dem Programmieren gestartet werden kann, muss noch in der rechten unteren Ecke das MicroPython vom RaspberryPi Pico ausgewählt werden:

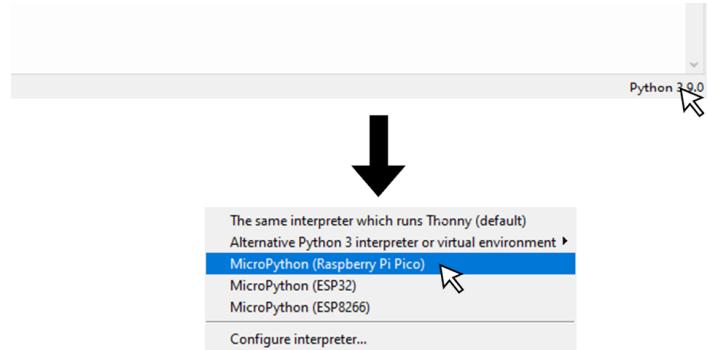


Abbildung 2.4.: MicroPython (Raspberry Pi Pico) (Quelle: projects.raspberrypi.org)

Falls **MicroPython (Raspberry Pi Pico)** nicht zur Auswahl steht, stecke denn Pico nochmals neu ein. Dieses Mal BOOTSEL **nicht** gedrückt halten. Wenn das Problem weiter besteht, wiederhole die Schritte aus dem Kapitel 2.1.

2.3. Erstes Programm

Stecke den RaspberryPi Pico auf den PCB wie beschrieben in Kapitel 1.2. Schreibe danach das folgende Programm in den Texteditor von Thonny:

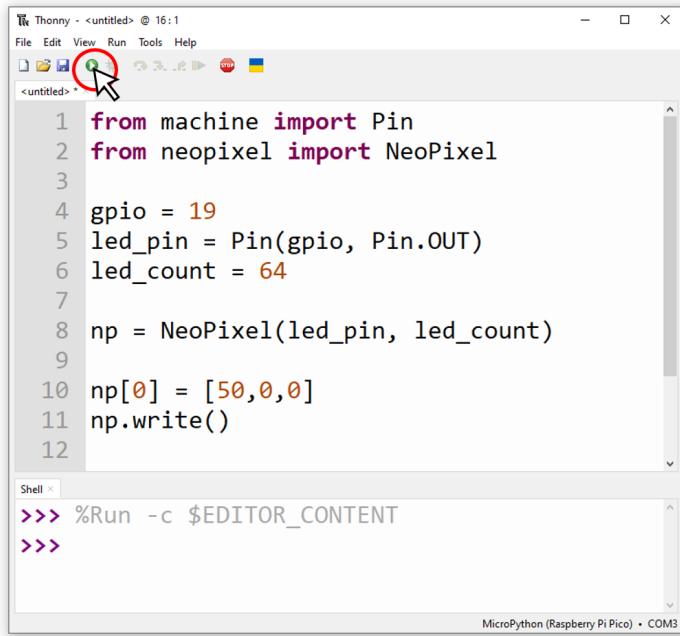
```
from machine import Pin
from neopixel import NeoPixel

gpio = 19
led_pin = Pin(gpio, Pin.OUT)
led_count = 64

np = NeoPixel(led_pin, led_count)

np[0] = [50,0,0]
np.write()
```

Drücke auf den grünen Play Knopf um dein erstes Programm auszuführen:



```

1 from machine import Pin
2 from neopixel import NeoPixel
3
4 gpio = 19
5 led_pin = Pin(gpio, Pin.OUT)
6 led_count = 64
7
8 np = NeoPixel(led_pin, led_count)
9
10 np[0] = [50,0,0]
11 np.write()
12

```

Shell <>> %Run -c \$EDITOR_CONTENT
>>>

MicroPython (Raspberry Pi Pico) • COM3

Abbildung 2.5.: Erstes Programm ausführen

Auf der Leiterplatte sollte nun die erste LED leuchten.

2.4. Programm auf RaspberryPi Pico speichern

Python-Programme können auf dem RaspberryPi Pico gespeichert werden. Wenn es ein Programm mit dem Namen *main.py* auf dem Pico gibt, wird dieses immer gestartet, wenn der Pico eingesteckt wird. Nenne dein Programm daher *main.py* falls es nach dem Einsticken laufen soll. Das Programm kann jederzeit geändert oder überschrieben werden.

Zum Speichern drücke im Menu auf *File > Save as ...*, wähle Raspberry Pi Pico aus, schreibe den Dateinamen und drücke OK:

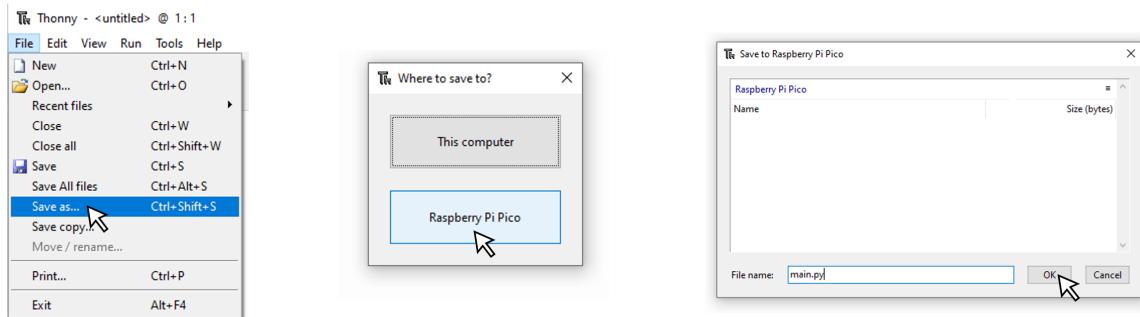


Abbildung 2.6.: Programm auf Pico speichern

2.5. Python Grundlagen

In diesem Teil lernen wir einige Programmiergrundlagen mit der Sprache Python. Schreibe jede der folgenden Aufgaben selber in Thonny und versuche mit kleinen Änderungen, ein anderes Verhalten des Programmes zu erreichen.

2.5.1. Konsolenausgabe

In der Programmierumgebung können im Konsolenfenster Nachrichten ausgegeben werden. Im Programm fügen wir diese Funktion mit dem Befehl print() ein.

```
print("Willkommen im WordClock Workshop!")
# Ausgabe in der Konsole: Willkommen im WordClock Workshop!
```

2.5.2. Datentypen

In einem Programm werden Variablen verwendet. Diesen wird ein Wert zugewiesen, der einem vordefinierten Datentypen entspricht. Jeder Datentyp hat bestimmte Eigenschaften, die folgend erklärt werden.

Number

Eine Number ist eine beliebige Zahl. Sie darf positiv, negativ oder eine Dezimalzahl sein.

```
a = 33
b = -12
c = 2.85
```

String

Ein String ist eine Kette von beliebigen Zeichen. Wir verwenden ihn für Text.

```
d = "Hallo!"
e = "Die WordClock hat 64 LEDs."
```

Boolean

Ein Boolean kann nur zwei Werte annehmen: Wahr oder Falsch. Er kann beispielsweise zur Speicherung eines Status verwendet werden.

```
f = True
g = False
h = (2 > a) # False
i = not g # True
```

Lists

In einer Liste können mehrere Werte gespeichert werden. Auf die Werte wird über einen Index von 0 bis Länge-1 zugegriffen.

```
farben = ["Rot", "Orange", "Blau", "Violett"] # Liste hat 4 Werte
farbe1 = farben[0] # Rot
print(farben[3]) # Ausgabe: Violett
farben[2] = "Weiss" # Blau wurde durch Weiss ersetzt
farben[4] = "Gelb" # FEHLER: Diese Liste hat nur 4 Eintraege
```

2.5.3. GPIO Ausgang

Um auf unserer Matrix eine LED zu steuern, benötigen wir einen GPIO-Pin. Mit diesem können wir die LEDs ein- und wieder ausschalten. Schreibe ein Programm, dass eine LED auf der Matrix ein- und nach einer Sekunde wieder ausschaltet.

```
from machine import Pin
from neopixel import NeoPixel
from utime import sleep_ms

# GPIO-Pin fuer Neopixel
pin_leds = 19

# Anzahl der LEDs
anzahl_leds = 64

pin = Pin(pin_leds, Pin.OUT)
np = NeoPixel(pin, anzahl_leds)

np[0] = [0, 0, 25]
np.write()

sleep_ms(2000)
np[0] = [0, 0, 0]
np.write()
```

2.5.4. Endlos Schleife

Das vorherige Programm war nach einem Ein- und Ausschaltvorgang beendet. Dies ist unpraktisch, das Programm soll nicht enden. Für diese Anwendung gibt es Schleifen. Eine Schleife wiederholt alle Befehle in ihrem Block, solange die Abbruchbedingung True ist.

Einrücken in Python

Damit der Pico den Code verstehen kann, müssen Regeln befolgt werden. Die Gruppe von Befehlen innerhalb einer Schleife muss um mindestens einen Leerschlag oder einen Tab eingerückt werden. So wird erkannt, welche Befehle sich innerhalb der Schleife befinden.

Schreibe ein Programm, mit dem eine LED auf der Matrix jede Sekunde ihren Zustand ändert.

```
from machine import Pin
from neopixel import NeoPixel
from utime import sleep_ms

# GPIO-Pin fuer Neopixel
pin_leds = 19

# Anzahl der LEDs
anzahl_leds = 64

pin = Pin(pin_leds, Pin.OUT)
np = NeoPixel(pin, anzahl_leds)

while True:
    np[0] = [0, 0, 25]
    np.write()
    sleep_ms(1000)
    np[0] = [0, 0, 0]
    np.write()
    sleep_ms(1000)

# Programmende: Hier kommen wir nie hin.
```

2.5.5. GPIO Eingang und bedingte Verzweigungen

So wie wir bei den LEDs einen Zustand setzen können, ist es auch möglich den Zustand von Switch und Joystick zu lesen. Schreibe ein Programm, mit dem eine LED mit dem Switch ein- und ausgeschaltet werden kann.

```
from machine import Pin
from neopixel import NeoPixel
from utime import sleep_ms

# GPIO-Pin fuer Neopixel
pin_num_leds = 19

# GPIO-Pin fuer Switch
pin_num_switch = 9

# Anzahl der LEDs
anzahl_leds = 64

pin_leds = Pin(pin_num_leds, Pin.OUT)
np = NeoPixel(pin_leds, anzahl_leds)
pin_switch = Pin(pin_num_switch, Pin.IN)

while True:
    value = pin_switch.value()

    if value == 1:
        np[0] = [50, 50, 50]
    else:
        np[0] = [0, 0, 0]

    np.write()
    sleep_ms(100)
```

2.5.6. LEDs seriell ein- und ausschalten

Mit einer for-Schleife können wir über die gesamte NeoPixel Liste iterieren. Dadurch ist es möglich mit wenig Code alle LEDs zu steuern. Schreibe ein Programm, das alle LEDs der Reihe nach einschaltet. Sobald alle LEDs leuchten sollen sie wieder abgestellt werden.

```
from machine import Pin
from neopixel import NeoPixel
from utime import sleep_ms

pin = Pin(19, Pin.OUT)
np = NeoPixel(pin, 64)

sleepTime = 35
brightness = 35

while True:
    for i in range(len(np)):
        np[i] = [brightness, 0, 0]
        np.write()
        sleep_ms(sleepTime)

    for i in range(len(np)):
        np[i] = [0, brightness, 0]
        np.write()
        sleep_ms(sleepTime)

    for i in range(len(np)):
        np[i] = [0, 0, brightness]
        np.write()
        sleep_ms(sleepTime)

    for i in range(len(np)):
        np[i] = [brightness, brightness, brightness]
        np.write()
        sleep_ms(sleepTime)

    for i in range(len(np)):
        np[i] = [0, 0, 0]
        np.write()
        sleep_ms(sleepTime)
```

2.5.7. Matrix Navigation

56	57	58	59	60	61	62	63
48	49	50	51	52	53	54	55
40	41	42	43	44	45	46	47
32	33	34	35	36	37	38	39
24	25	26	27	28	29	30	31
16	17	18	19	20	21	22	23
8	9	10	11	12	13	14	15
0	1	2	3	4	5	6	7

Abbildung 2.7.: Liste Indizes auf der LED Matrix

Die Abbildung 2.7 zeigt die Liste für die LEDs der Matrix. Über diese Indizes kann auf der Neopixel-Liste eine Farbe definiert werden. Schreibe ein Programm, mit dem die Joystick Eingabe gelesen- und die leuchtende LED verschoben wird. Wird mit dem Joystick nach links gezeigt, soll sich die leuchtende LED nach links bewegen. Analog sollen die Bewegungen in die anderen Richtungen funktionieren. Wenn die leuchtende LED am Rand der Matrix angekommen ist wird das erkannt und sie soll an derselben Stelle stehenbleiben.

```
from machine import Pin
from neopixel import NeoPixel
from utime import sleep_ms

pin = Pin(19, Pin.OUT)
np = NeoPixel(pin, 64)

pin_js_r = Pin(2, Pin.IN) #rechts
pin_js_l = Pin(7, Pin.IN) #links
pin_js_u = Pin(3, Pin.IN) #hoch
pin_js_d = Pin(6, Pin.IN) #runter

currentPosition = 0
while True:
    np[currentPosition] = [10, 10, 30]
    np.write()
    sleep_ms(200)

    if pin_js_r.value() == 0:
        if (currentPosition + 1) % 8 != 0:
            np[currentPosition] = [0, 0, 0]
            currentPosition += 1
    elif pin_js_l.value() == 0:
        if currentPosition % 8 != 0:
            np[currentPosition] = [0, 0, 0]
            currentPosition -= 1
    elif pin_js_u.value() == 0:
        if currentPosition <= 55:
            np[currentPosition] = [0, 0, 0]
            currentPosition += 8
    elif pin_js_d.value() == 0:
```

```

if currentPosition >= 8:
    np[currentPosition] = [0, 0, 0]
    currentPosition -= 8

```

2.6. Python Library von ZHAW

Auf dem folgenden Link stellt die ZHAW verschiedene Module mit Beispielprogrammen zur Verfügung, welche für die LED-Matrix designed wurden:

<https://github.com/InES-HPMM/LED-Matrix-Workshop>

Um ein Modul auf der LED-Matrix zu verwenden, muss folgendes gemacht werden:

- Lade das gewünschte Python Modul herunter (.py Datei)
- Öffne die Python Datei mit Thonny
- Wähle im Menu: File > Save as ... (oder Ctrl+Shift+S)
- Wähle als Speicherort den RaspberryPi Pico
- Speichere die Datei unter dem gleichen Namen
- Ab jetzt können alle Funktionen des Moduls mit **from modulname import x,y,z** importiert und benutzt werden.

2.6.1. Modul: ledmatrix

Dieses Modul beinhaltet Funktionen/Klassen um die Ausgabe auf der LED-Matrix zu vereinfachen. Im folgenden Codeabschnitt werden die Funktionen vorgestellt. Zusätzlich werden Beispiele für die Verwendung aufgeführt:

```

from ledmatrix import (
    ColorTable,
    LedMatrix,
    PixelColor,
)
from utime import sleep_ms

# Kreiere eigene Farbe
my_color = PixelColor(2, 211, 13)

# LED Matrix Objekt
matrix = LedMatrix(8, 8)

# Setze Helligkeit
matrix.set_brightness(20)

# Setze Pixel x=0,y=1
# ColorTable beinhaltet 16 Farben
matrix[0, 1] = ColorTable.YELLOW

# Sende Farbwerte an LED-Matrix
matrix.apply()
sleep_ms(1000)

# Alle Neopixel auf blau setzen
matrix.fill(ColorTable.BLUE)
matrix.apply()
sleep_ms(1000)

```

```

# Alle Farben auf Null setzen
matrix.clear()
matrix.apply()
sleep_ms(1000)

# Liste von Pixel auf einen Farbwert setzen
pixel_list = [(5, 1), (5, 2), (6, 1)]
matrix.draw_list(pixel_list, ColorTable.GREEN)
matrix.apply()
sleep_ms(1000)

# Linien zeichnen:
# Gerade Linie
matrix.draw_line(
    (0, 0), (0, 4), ColorTable.ORANGE
)
# Diagonale Linie
matrix.draw_line(
    (0, 7), (5, 5), ColorTable.PINK
)
matrix.apply()

```

2.6.2. Modul button

In diesem Modul sind Schiebeschalter sowie Joystick den GPIOs zugeordnet und als Inputs definiert. Nach dem initialisieren von *Button()* können direkt die Werte gelesen werden:

```

from button import Button

# Button initialisieren
btn = Button()

# Verhalten bei einem button Ereignis festlegen
def left(_):
    print(".")
btn.set_left_handler(left)

# Werte vom Joystick & Schiebeschalter einlesen
btn.up.value()
btn.down.value()
btn.left.value()
btn.right.value()
btn.center.value()
btn.switch.value()

```

2.6.3. Modul characters

Auf der LED-Matrix kann man Buchstaben und Zahlen als Darstellung auf der 8x8 Matrix ausgeben. Wenn das Ziffernblatt eingelegt ist, hat man die Zahlen der Minuten- & Stundenanzeige zur Verfügung. Das Modul *characters* beinhaltet Funktionen um auf diese beiden Arten Zahlen/Buchstaben leuchten zu lassen.

Das folgende Codebeispiel zeigt wie sie benutzt werden. Für die Ansteuerung werden ebenfalls Funktionen aus dem Modul *ledmatrix* benutzt, welche im vorherigen Absatz erklärt werden.

```

from characters import (
    ClockTable,

```

```

        CharacterTable ,
)
from ledmatrix import ColorTable, LedMatrix
from utime import sleep_ms

# LED-Matrix initialisieren
matrix = LedMatrix(8, 8)
matrix.set_brightness(20)

# Die 10 von der Minutenanzeige in blau
# leuchten lassen
matrix.draw_list(
    ClockTable.MIN_TEN, ColorTable.BLUE
)
matrix.apply()
sleep_ms(1000)

# Die 10 von der Stundenanzeige in gruen
# leuchten lassen
matrix.clear()
matrix.draw_list(
    ClockTable.HOUR_TEN, ColorTable.GREEN
)
matrix.apply()
sleep_ms(1000)

# Buchstabe A auf LED-Matrix ausgeben in rot
matrix.clear()
matrix.draw_list(
    CharacterTable.A, ColorTable.RED
)

```

A. Anhang

A.1. PCB Schematic

Version	1	2	3	4	5	6	7	8
Date	22.12.22	eigr	Route joystick by 45°, add level shifter					
Sign	1.1	1.2	30.01.23	sczr	Beauty schematics, add missing pin 1 marker			
Release Notes								
A								

LED-Matrix Rows								
ROW8	LED Matrix_Row_SchDoc							
ROW7	LED Matrix_Row_SchDoc							
ROW6	LED Matrix_Row_SchDoc							
ROWS	LED Matrix_Row_SchDoc							

Raspberry Pi Pico								
U_RaspPi_Pico_SchDoc								
U_RaspPi_Pico_SchDoc								
U_RaspPi_Pico_SchDoc								
U_RaspPi_Pico_SchDoc								

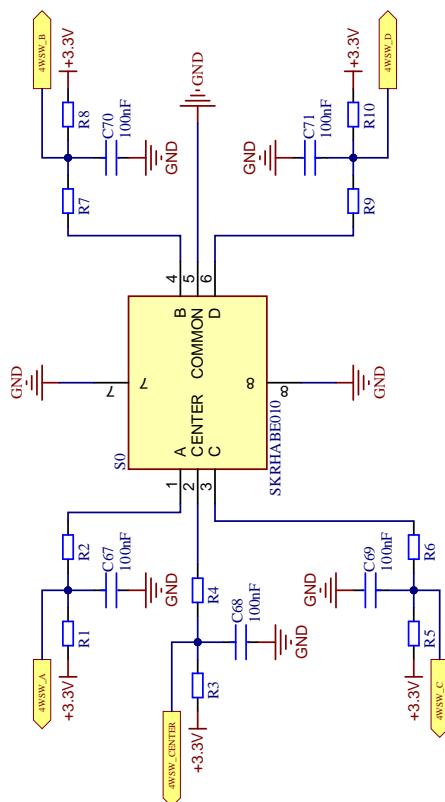
Joystick and Switch								
U_Switch_Joystick_SchDoc								
U_Switch_Joystick_SchDoc								
U_Switch_Joystick_SchDoc								
U_Switch_Joystick_SchDoc								

Jumpers								
U_Jumper_Connectors_SchDoc								
U_Jumper_Connectors_SchDoc								
U_Jumper_Connectors_SchDoc								
U_Jumper_Connectors_SchDoc								

D								
Zhaw School of Engineering								
Zhaw School of Engineering								
Zhaw School of Engineering								
Zhaw School of Engineering								

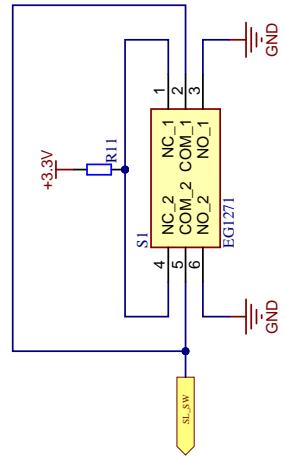
1 2 3 4

Joystick



^A Button push is sensed by a pull to GND (logical low)
Debounce is already done in hardware with the RC filter (1k
Resistor, 0.1uF Capacitor).
Together with the external 10k pullup Resistor, the 1k Resistor
creates a voltage divider. Therefore, logical low wont be 0V. The
MCU interprets voltages between -0.3 and 0.8V as logical low /
logical 0.
 $((11\text{Ohm})/(1\text{kOhm}+10\text{kOhm})) \cdot 3.3V = 0.3V$
This is ok for a logical low.

Switch



D		C		B		A	
Sheet 2 of 6	Print date: 31.01.2023	Drawn by: Lukas Eugster					
File: E:\git\LED-Matrix-Workshop\hardware\05_pcb_dev\eug\PCB_Project_santSwitch_Joystick_Sch							
1	2	3	4				
Size	Document Number	Version					
A4	2	1.2					

Zurich University of Applied Sciences
InES - Institute of Embedded Systems
PO Box 805
8401 Winterthur
Switzerland



© InES This document is property of InES - Institute of Embedded Systems
No copying or distribution without prior written permission of InES

Title Joystick and Switch

Size Document Number

A4 2

Version 1.2

Sheet 2 of 6

Print date: 31.01.2023

Drawn by: Lukas Eugster

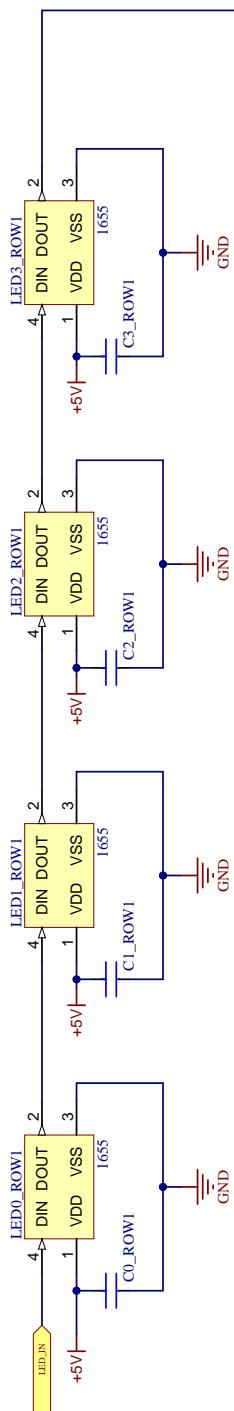
File: E:\git\LED-Matrix-Workshop\hardware\05_pcb_dev\eug\PCB_Project_santSwitch_Joystick_Sch

4

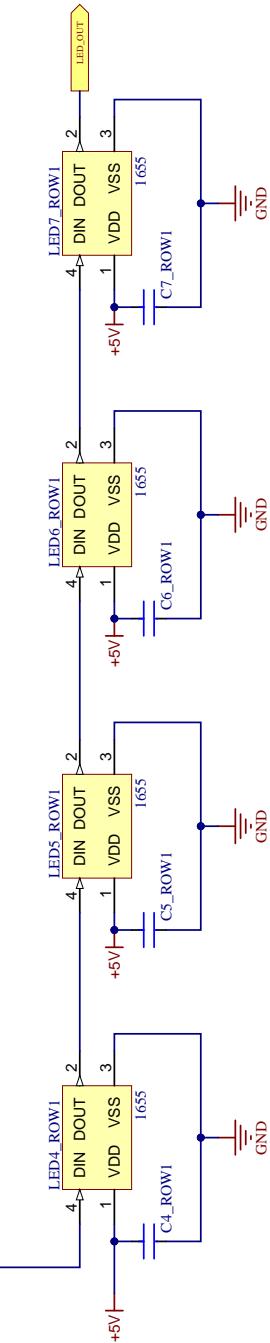
Row of 8 LED's

1 2 3 4

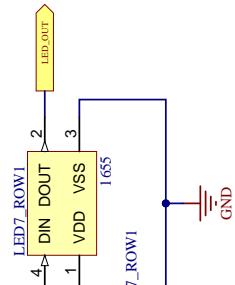
A



B



C



D

LED-Matrix Row	
Size	Document Number
A4	3

Version
1.2

Sheet 3 of 6	Date: 31.01.2023	Drawn by: Lukas Eugster
File: E:\git\LED-Matrix-Workshop\hardware\05_pcb_dev\ugitPCB_Project_santLED_Matrix		4

Zurich University of Applied Sciences
InES - Institute of Embedded Systems
PO Box 805
8401 Winterthur
Switzerland

© InES This document is property of InES - Institute of Embedded Systems
No copying or distribution without prior written permission of InES

