Umsetzung eines eigenen GPIO-Schalter

CODE

```
#gpio_high_low_button_1.py GPIO-Schalter
#! /usr/bin/env python
# -*- coding: utf-8

try:
    import Tkinter as tk
except ImportError:
    import tkinter as tk

import RPi.GPIO as GPIO #Bibliothek laden
GPIO.setmode(GPIO.BOARD) #Verdrahtung nach der Pin-Nummer
GPIO.setwarnings(False) #Warnungen abschalten

PORT = (12)

class GPIO_on_off(tk.LabelFrame):
    def __init__(self, root, port):
        tk.LabelFrame.__init__(self, root, text = "{0} {1} {2}".format("GPIO",
            port, "LOW"), highlightthickness = 11, font = "Arial 9")
        self.root = root
        self.port = port
        GPIO.setup(port, GPIO.OUT) #Pin als Ausgang
        GPIO.output(self.port, False) #Pin anschalten
        self.button_state = False
        self.button = tk.Button(self, width=4, bd=2, height=1,
            highlightthickness=11, activebackground = "lightgreen",
            bg="lightgreen", font="Arial 15", command=self.change_state)
        self.button.pack()

    def change_state(self):
        if self.button_state:
            GPIO.output(self.port, False) #Pin auschalten
            self.button.config(bg="lightgreen", relief="raised",
                activebackground="lightgreen")
            self.config(text="{0} {1} {2}".format("GPIO", self.port,
                "LOW"))
            self.button_state = False
        else:
            GPIO.output(self.port, True) #Pin anschalten
            self.button.config(bg="red", relief="sunken",
                activebackground="red")
            self.config(text="{0} {1} {2}".format("GPIO", self.port,
                "HIGH"))
            self.button_state = True

    def on_exit(self):
        GPIO.cleanup() #Grundzustand herstellen
        self.root.destroy()

if __name__ == '__main__':
    root = tk.Tk()
    gpio_button = GPIO_on_off(root, PORT)
    gpio_button.pack()
    root.protocol("WM_DELETE_WINDOW", gpio_button.on_exit)
    root.resizable(0, 0)
    root.mainloop()
```

CODE

Zeilen 2-3
Dies ist der Shebang und nur für unixartige Betriebssysteme relevant.

Zeilen 5-8
Das Script importiert Tkinter und testet dabei auf die verwendetet Python-Version (2 o. 3) mit einem Errorhandling. Das **(„as tk“)** bedeutet den Import von Tkinter mit der Abkürzung tk **–** könnte auch anderes heissen, doch dies würde den Code für andere Nutzer unleserlich machen.

Zeile 14
Die Konstante **PORT** ist nur für den Test des Scriptes gewählt. Es können hier alle verwendbaren GPIO's eingetragen werden.

Zeilen 30-64

In diesem Abschnitt ensteht der eigentliche Schalter. Wer mehr über Tkinter erfahren will, der sollte die Zeile 42 einkommentieren und **(„help(self.button“)** gibt einem alle Einzelheiten zur Klasse-Button heraus. Geht natürlich auch mit allen anderen Klassen (Frame, Label, etc. oder mit tk selbst). Die Zeile 38 **(GPIO.output(self.port, False)** ist aus mir nicht erklärbaren Gründen nötig, da auch nach eine GPIO.cleanup() die eingeschalteten Pin's an bleiben ?!

Zeilen 16-48
Hier ensteht das eigentliche Hauptfenster **(„root“)**, der GPIO-Schalter wird erzeugt **(„gpio_button = GPIO_on_off()“)** und in das Hauptfenster gepackt. Die Zeile 70 verhindert das Schließen des Hauptfensters und leitet es auf (**root.protocol(ROOT_PROTOCOL, gpio_button.on_exit)**die an den GPIO-Schalter gebundene Funktion**(def on_exit(self):)**um. Sie wird aufgerufen, versetzt den Port in Grundzustand und „zerstört“ das Hauptfenster **(self.root.destroy()).**

Darstellung des Schalters

Auf dem raspberry mit RDP

Auf Lubuntu

Wiederverwendung des Codes

CODE

```
#gpio_high_low_button_4.py GPIO-Schalterleiste mit 4 Schaltern
#! /usr/bin/env python
# -*- coding: utf-8

try:
    import Tkinter as tk
except ImportError:
    import tkinter as tk

import RPi.GPIO as GPIO #Bibliothek laden
GPIO.setmode(GPIO.BOARD) #Verdrahtung nach der Pin-Nummer
GPIO.setwarnings(False) #Warnungen abschalten

PORTS = (12, 16, 18, 22)

class GPIO_on_off(tk.LabelFrame):
    def __init__(self, root, port):
        tk.LabelFrame.__init__(self, root, text = "{0} {1} {2}".format("GPIO",
            port, "LOW"), highlightthickness = 11, font = "Arial 9")
        self.root = root
        self.port = port
        GPIO.setup(port, GPIO.OUT) #Pin als Ausgang
        GPIO.output(self.port, False) #Pin anschalten
        self.button_state = False
        self.button = tk.Button(self, width=4, bd=2, height=1,
```

```
            highlightthickness=11, activebackground = "lightgreen",
            bg="lightgreen", font="Arial 15", command=self.change_state)
        self.button.pack()

    def change_state(self):
        if self.button_state:
            GPIO.output(self.port, False) #Pin auschalten
            self.button.config(bg="lightgreen", relief="raised",
                activebackground="lightgreen")
            self.config(text="{0} {1} {2}".format("GPIO", self.port,
                "LOW"))
            self.button_state = False
        else:
            GPIO.output(self.port, True) #Pin anschalten
            self.button.config(bg="red", relief="sunken",
                activebackground="red")
            self.config(text="{0} {1} {2}".format("GPIO", self.port,
                "HIGH"))
            self.button_state = True

    def on_exit(self):
        GPIO.cleanup() #Grundzustand herstellen
        self.root.destroy()

if __name__ == '__main__':
    root = tk.Tk()
    tk.Label(root, font="Arial 15", bd=5,
    text="-------------- >>> Meine Schalterleiste <<< -------------",
    ).pack()
    for port in PORTS:
        gpio_button = GPIO_on_off(root, port)
        gpio_button.pack(side="left")
    root.protocol("WM_DELETE_WINDOW", gpio_button.on_exit)
    root.title("GPIO SCHALTERLEISTE")
    root.resizable(0, 0)
    root.mainloop()
```

CODE

Der Sinn solch eines eigenen GPIO-Schalter ist die Wiederverwendbarkeit des Codes. Möchte man mehreren Schalter gleichzeitig darstellen und schalten oder den Schalter in einem anderen Programm verwenden, kann dieser nun importiert werden.

Zeilen 55-57
Die gewünschten Ports sind in einer Liste. Über diese wird iteriert und es wird für jeden Port ein eigener Schalter erzeugt.

Darstellung der Schaltleiste

Auf Lubuntu

Auf dem raspberry mit RDP

Der Import des eigenen Schalters funktioniert in dem man sein eigenes Script importiert.

```
from Name_meines_Scriptes import GPIO_on_off
```

Man sollte zum Programmieren nicht die Python-IDLE nehmen, da sie nicht wirklich gut sein soll und eigentlich unbrauchbar ist. Kann ich nicht beurteilen, da mir da die nötige Sachkenntnis fehlt. Probleme macht sie, wenn man Tkinter-GUI's entwickelt, denn die IDLE ist auch damit programmiert und so entstehen Verhaltensweisen und Fehler in der eigenen GUI, die man dann unnötigerweise sucht. Diese Probleme hatte ich auch. Ich verwende Scite auf dem Desktop oder nano im Terminal. Beide unter GPL und einfach in der Bedienung. Die Scripte teste ich über das Remote Desktop Protokoll mit xrdp auf dem raspberry und remmina auf LXDE-Lubuntu (Distributor ID:Ubuntu Description: Ubuntu 15.04 Release:15.04 Codename: vivid). Mit **(„sudo apt-get install xrdp“)** kann man sich xrdp auf den raspberry holen.