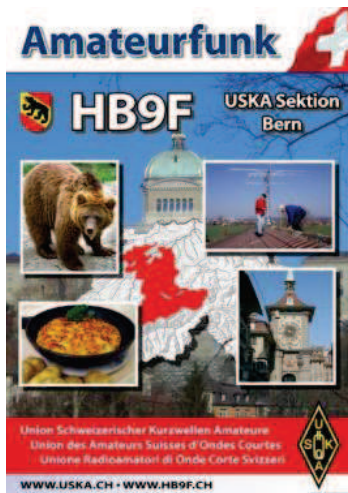


Weiterbildungskurs: Python mit dem Raspberry Pi



Egid Plüss, HB9ABH

unter Mitarbeit von Jarka Arnold, PHBern

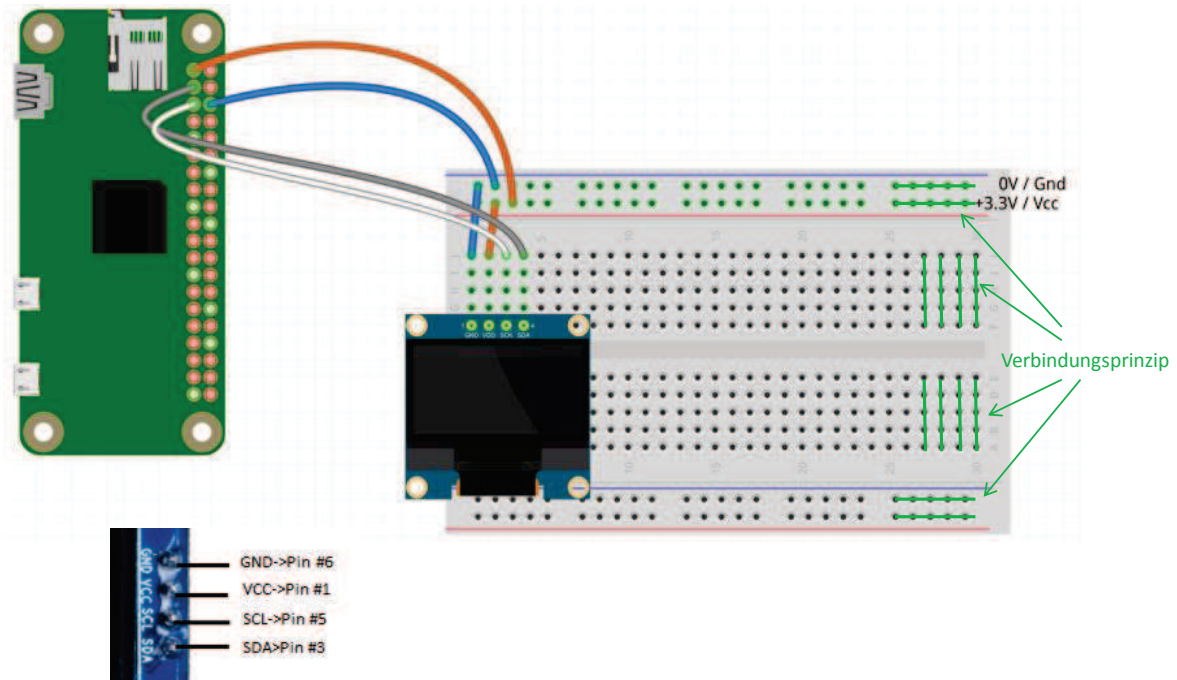
Kurs-Website:

www.python-exemplarisch.ch/uskabern

Kursprogramm:

09.00 - 09.15	Begrüssung durch den Präsidenten der Sektion Bern, HB9GAA Abgabe des Kursmaterials
09.15 - 09.45	Entwicklung von Microcontroller-Systemen
09.45 - 10.30	Vorstellung des Experimentiermaterials Erste Schritte mit dem Raspberry Pi
10.30 - 10.45	Pause
10.45 - 11.00	GPIO am Raspberry Pi
11.00 - 11.30	Erste Schritte mit Python: Blinken einer LED
11.30 - 12.00	Programmstrukturen: while, if
12.00 - 13.00	Mittagessen
13.00 - 13.30	Leistungstreiber am DigitalOut
13.30 - 14.00	Taste an DigitalIn
14.00 - 14.30	Siebensegment-Display TM1637
14.30 - 15.00	OLED-Display (SSD1306) am I2C-Bus
15.00 - 15.15	Pause
15.15 - 15.45	Analog-Digital Wandler (ADS1115)
15.45 - 16.45	Remote messen und steuern
16.45 - 17.00	Schlussdiskussion

Anschluss des Displays auf dem Breadboard



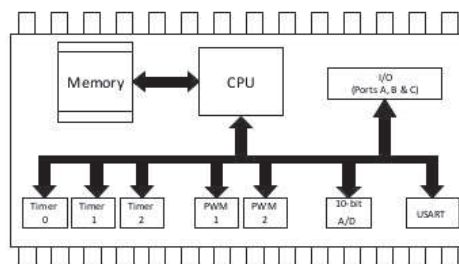
Übersicht Microcontroller-Architekturen

- 8051/8032 Entwickelt von Intel -Architektur in anderen Fabrikaten übernommen
- PIC ("Pick") Entwickelt von Microchip, sehr beliebt in Kleinsystemen (> 10 Milliarden)
- AVR, ATmega Entwickelt von Atmel, im Hobbybereich verbreitet durch Arduino
- ARM Entwickelt von ARM (England). Vorherrschende Architektur für 32-bit Microcontroller. Entwicklung unter mbed. Im Raspberry Pi verbaut
- ESP8266 32-bit Microcontroller von Espressif mit dem Tensilica Xtensa DSP (Digital Signal Processor)

8051, PIC and AVR haben (8-bit) **Harvard-Architektur**, d.h. separaten Speicher für Programm und Daten
 ARM hat (16 oder 32-bit) **von Neumann-Architektur**, d.h. gemeinsamen Speicher für Programm und Daten

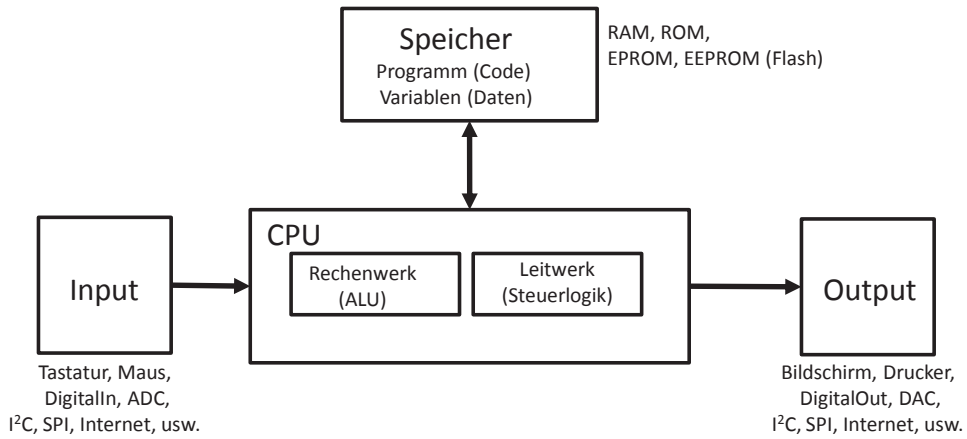
8051 und PIC benötigen mehrere Clockzyklen pro Instruktion
 AVR und ARM benötigen meist nur 1 Clockzyklus pro Instruktion (RISC)

Typisch für Microcontroller: Einfacher Instruktionssatz, internes RAM, eingebaute Peripherie.
 Beispiel PIC:



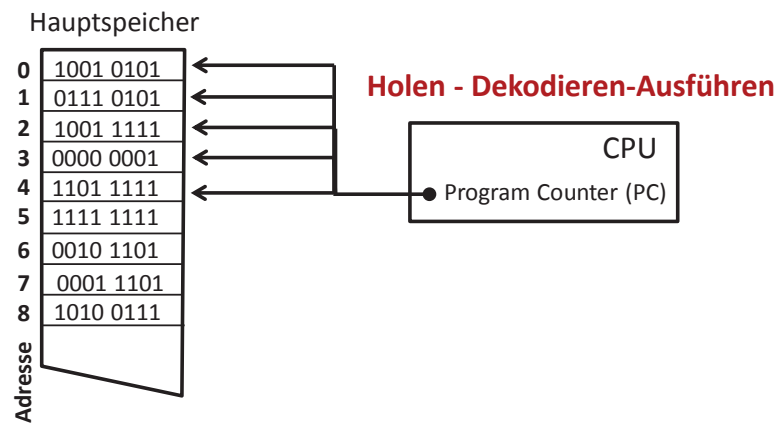
A computer on a chip

Prinzipieller Aufbau eines Computersystems



Bei Microcontroller werden gewisse Teile von Input/Output/Speicher im CPU-Chip integriert

**Prinzipieller Programmablauf:
Das Programm besteht aus Instruktionen (Maschinenbefehlen)**



8 bits = 1 Byte (8 Schalter aus/ein)

Byte-Darstellung:

1	1	0	1	0	1	0	1
---	---	---	---	---	---	---	---

Wertigkeiten: 128 64 32 16 8 4 2 1 = 210 (dec)

Hexadezimal: 8 4 2 1 8 4 2 1 = C5 (hex)

Halbytes: C 5

Legend: 10 = A, 11 = B, 12 = C, 13 = D, 14 = E, 15 = F

Entwicklung von Microcontroller-Systemen

1. Maschinenprogramm auf einem Entwicklungssystem editieren
2. Maschinenprogramm auf Targetsystem downloaden
3. Maschinenprogramm auf Targetsystem ausführen

Problem: Maschinenprogramm schreiben ist sehr, sehr mühsam!

Lösung 1 : Symbolische Abkürzungen für Maschinenbefehle erfinden

Assemblersprache ("Assembler")

```
MOV 10, #2
MOV 11, #3
MOV acc, 10
ADD acc, 11
MOV 12, acc
```

Lösung 2 : Höhere Programmiersprache erfinden (maschinennah)

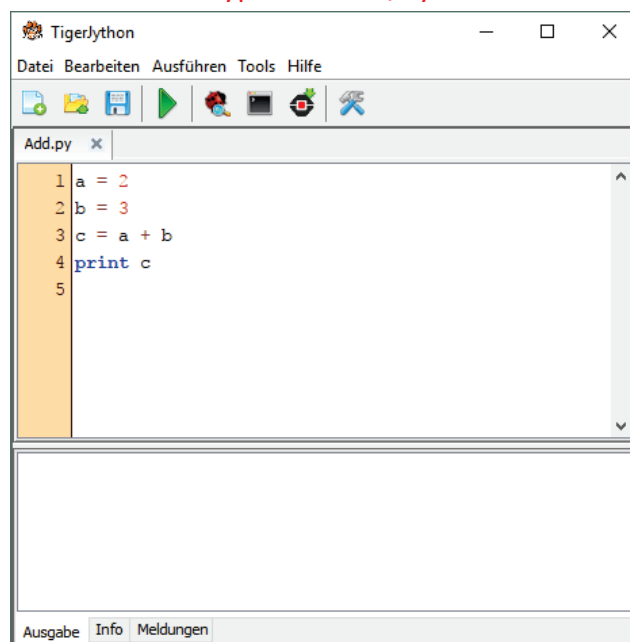
```
int a = 2
int b = 3
int c = a + b
```

typisch: C/C++

Lösung 3: Einfache höhere Programmiersprache erfinden

```
a = 2
b = 3
c = a + b
```

typisch: Basic/Python



Und los geht's!

Alle schalten ein!

Der RPi ist ein Linux Computer mit Bildschirm/Tastatur/Maus

Nur im Notfall und für Inbetriebnahme mit Original SD-Karte empfehlenswert

Demo: booten

Materialliste:

Raspberry Pi: Typ 2, 3 oder Zero, ZeroW
 Typ 2 und Zero mit WLAN Dongle

Netzgerät 5V/2A MicroUSB

USB-Kabel: MicroUSB auf USB (Anschluss an PC)

Experimentiermaterial:

Anzahl	Beschreibung
1	Breadboard (klein, 400pin)
20	Jumperkabel m-m (10cm)
20	Jumperkabel w-w (10cm)
20	Jumperkabel m-w (20cm)
1	Potentiometer 10k
1	Impulse-Taste 6x6mm 2pin
1	Low-Power LED (rot)
1	Widerstand 330 Ohm (CFR-25JT-52-330R)
1	3V Buzzer (2 polig)
1	ACP 1115 16 Bit ADC
1	OLED Display Modul 128x64 SSD1306
1	Display TM 1637
1	micro SD-Card (16GB)

Erste Schritte mit dem Raspberry Pi**Startfrust vermeiden!**

Know-How von anderen beziehen!

Bitte eigene ergänzende Notizen machen!

Der Raspi startet mit einem Betriebssystem (Linux). Dieses befindet sich auf der SD-Karte

Die bereits vorkonfigurierte SD-Karte verwenden!

Szenario 1:
 Anschluss eines HD-Bildschirms
 Tastatur und Maus

HDMI-Monitor: direkt
 DVI-Monitor: über HDMI-DVI-Adapter

Nur Demo

**Szenario 1:**

**Anschluss eines HD-Bildschirms,
 Tastatur und Maus**

Einloggen:

username: **pi**

password: **raspberrypi**

(Tastatur Tasten y/z beachten!)

Nachteile: Man ist isoliert vom Rest der Welt. Man muss sich mit Linux auskennen

Vorteil: Man "fühlt" den Raspi wie einen eigenen kleinen Computer

Szenario 2: Direkte Verbindung mit Ethernet-Kabel

Voraussetzung: Notebook mit Ethernet-Anschluss
(event. Ethernet-USB-Adapter verwenden)

Der PC wird so eingerichtet, dass er sozusagen sein WLAN mit einem Benutzer teilt, der am Ethernet-Port hängt (ICS: Internet Connection Sharing)

(eine tatsächliche Verbindung zum Internet ist nicht notwendig)

Achtung: Der Raspi darf keinen aktiven WLAN-Port haben!

Szenario 2:
Direkte Verbindung
mit Ethernet-Kabel



Nur Demo

Gute Variante für in Geräte eingebaute Raspberry Pi!

Genaue Anleitung siehe:

http://www.python-exemplary.com/index_en.php?inhalt_links=navigation_en.inc.php&inhalt_mitte=raspi/en/development.inc.php

Szenario 3: Verbindung über Access Point (WLAN Router)

- Router (DHCP) vergibt IP-Adresse an Host und an Raspi (meist im Bereich 192.168.0.nnn)
- Damit der Host den Raspi ansprechen kann, muss er dessen IP-Adresse kennen
- Zwei übliche Arten:
 - Mit dem Router-Manager (Client-List)
 - Mit einem Scanner-Programm
 - Angry IP Scanner (für Windows/Mac/Linux) <http://angryip.org>
 - Advanced IP Scanner (nur Windows) <http://advanced-ip-scanner.com>
 - Fing App
App Store oder Google Play
- Damit der Raspi über WLAN an den Router kommt, muss er dessen SSID/Passwort kennen
- Default für Raspibrick-Firmware:
SSID: **raspilink**, Passwort: **aabbaabbaabb**
- Einstellen mit **RaspiBrickConfig.jar** auf der Windows-Partition der SD-Karte

Bewährter Router:
TP-LINK TL-WR841N

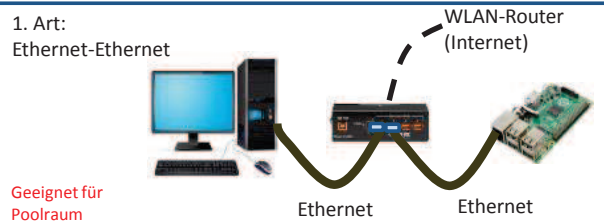
siroop.ch Fr. 18.-

Konfigurationsanleitung auf :

http://www.iython.ch/index.php?inhalt_links=navigation.inc.php&inhalt_mitte=pi2go/configRouter.inc.php

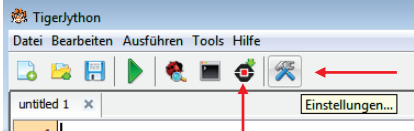
Es kann auch ein Access-Point eines Handys verwendet werden!

Host Router Target

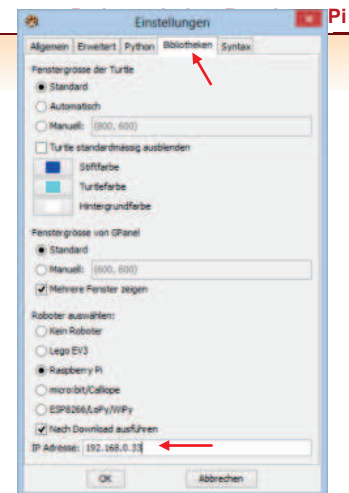


- Verbindung testen:
Windows: Eingabeaufforderung, Mac: Terminal öffnen
Eintippen: **ping 192.168.0.nnn**, wo nnn Ihre Zahl ist
Immer, wenn Verbindung nicht klappt, ping machen!
- Remote Desktop
Auf Raspi ist ein VNC-Server gestartet. Darauf zugreifen mit VNC-Client (VNC Viewer)
Anmelden mit IP-Adresse und Port: **<IP-Adresse>:5901**. Passwort: **123456**
TightVNC: www.tightvnc.com (Achtung: nur Viewer installieren)
VNC Viewer : www.realvnc.com/download/viewer
Der Raspi als vollwertiger Linux-Computer (alles vorinstalliert, z.B. Mathematica, TigerJython, usw.)
- Etwas Linux lernen:
Siehe Linux Spickzettel <http://www.python-exemplarisch.ch/dpkkurs/download/linux-cheat-sheet.pdf>
Wichtig: Falls Admin-Rechte nötig, den Befehlen immer **sudo** voranstellen
Software installieren:
`sudo apt-get update`
`sudo apt-get install <software>`

Remote Development mit TigerJython

- Einstellungen vornehmen:

- Weitere Tools im Menü:

Tools	Hilfe
Remote Terminal	← Remote Shell (über SSH)
Hinunterladen/Ausführen	← Programm herunterladen und ausführen *)
Modul herunterladen	← Name bleibt gleich, keine Ausführung (für Module, Bibliotheken)
Python auf dem Target beenden	← Notbremse, fast nie nötig!
Target herunterfahren	← Meist überflüssig, ohne Hemmungen Power weg!
Target herunterfahren/neu starten	



*) Alle heruntergeladenen Programm heissen gleich: myApp.py und befinden sich in /home/pi/scripts. Können auch dort ausgeführt mit: python myApp.py (im Terminal)

- Test: Programm downloaden und auf Target ausführen
Datei öffnen: Examples/Oled1.py
Downloadbutton (oder im Menu: Hinunterladen/Ausführen) klicken

Wichtig: Das Programm darf "hängen" z.B. while True:
Ausgaben (Fehler und print) werden in TigerJython angezeigt

Anmerkung: Falls das Programm automatisch beim Booten starten soll,
nennt man es autostart.py und lädt es runter (Modul hinterladen)

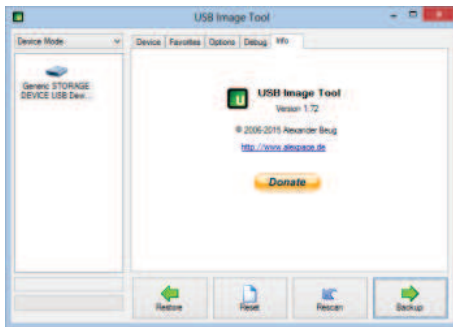
Management Tips

- Backup/Restore der SD-Karte (micro SD mind. 16 GB, Type 10)

USB Image Tool : www.alexpage.de/

SD-Karte muss gleich gross oder grösser sein

Falls was nicht geht, Original oder Backup kopieren (www.raspibrick.com)

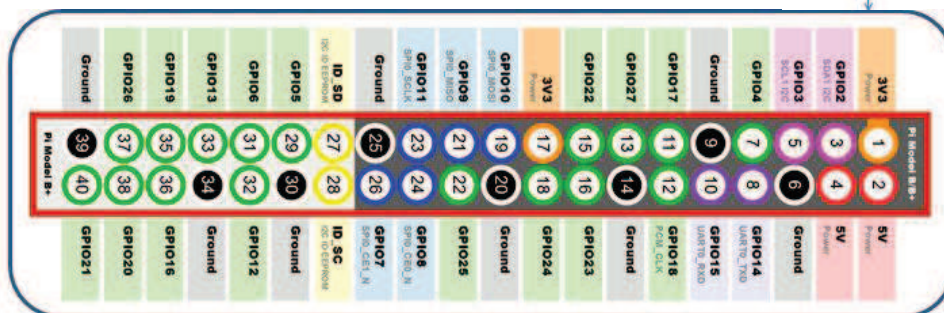


- Manchmal muss man SD-Karte formatieren

SDFormatter: www.sdcard.org/downloads/formatter_4

Pinbelegung, Spannungen/Ströme

- Pinbelegung GPIO 02..27 = 26 Pins



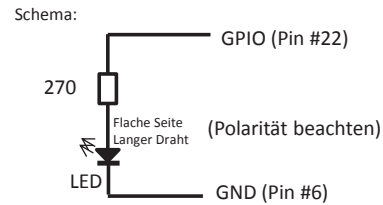
Grenzwerte:

- GPIO:
 - Ausgangsspannung 0, 3.3V (max 16 mA nach oben oder unten, total kleiner als 50 mA, also typisch 2mA / pin)
 - Eingangsspannung: 3.3V (kein Überspannungsschutz!!)
- Versorgungsspannung für externe Geräte:
 - 5V Versorgung: ca. 200-300 mA geteilt mit USB-Versorgung (also sehr wenig, zu wenig für Motoren)
 - 3.3 V Versorgung (von Model B+ an): ca 500 mA
- Spannungsversorgung: Micro-USB Netzgerät **mit 2A**
 - Variante: Externe Spannungsversorgung 5 V (Pin2): **NICHT AUF ÜBERSPANNUNG GESCHÜTZT!** (Micro-USB Eingang bleibt leer)

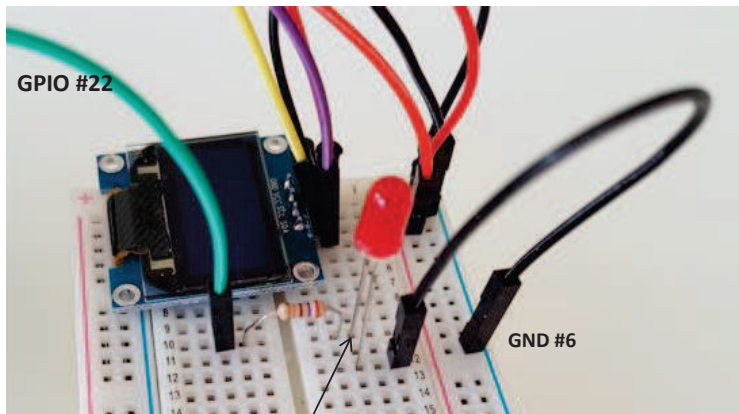
Digital Out: LED

- Experiment: Blinkende LED
 - LED langer Draht bzw. flache Seite ist +
 - Verbindungen mit Steckplatine (Bread board)
 - Strombegrenzender Seriewiderstand (typisch 270 Ohm, Strom max 10 mA)

Eventuell als Test ohne Software zuerst einfach statt bei Pin #22 bei Pin #1 (3.3V) einstecken!



Aufbau auf Steckplatine (Bread Board):



langer Draht beim Widerstand

Mein erstes Python-Programm -> ALLER ANFANG IST SCHWER**Digital Out: LED**

```
# DigitalOut1.py           ← Kommentar
import RPi.GPIO as GPIO   ← GPIO Modul importieren.
from time import sleep     ← RPI Tutorial (www.python-examplarisch.ch/rpi unter GPIO Wiki)
                           ← von Modul time die Funktion sleep importieren

GPIO.setmode(GPIO.BOARD) ← Funktion setmode() aus dem Modul GPIO aufrufen (mit Punktoperator)
                           ← Die Konstante GPIO.BOARD verwenden (Pin-Numerierung)
GPIO.setup(22, GPIO.OUT)  ← Pin #22 auf Output setzen (Digital out) GPIO.BOARD oder GPIO. BCM
GPIO.output(22, GPIO.HIGH) ← Output-Wert auf hoch (3.3V, High, H) setzen
sleep(5)                  ← 5 s warten
GPIO.output(22, GPIO.LOW) ← Output-Wert auf tief (0V, Low, L) setzen
GPIO.cleanup()           ← Ressourcen freigeben (GPIO rücksetzen, da es Programmende überlebt)
```

Programmstruktur: Sequenz
mache dies, dann das, dann jenes...

Programmstruktur: Repetition (Wiederholung mit while)

Solange Bedingung erfüllt, wiederhole den Block

gleiche Hardware

```
# DigitalOut2.py

import RPi.GPIO as GPIO
from time import sleep

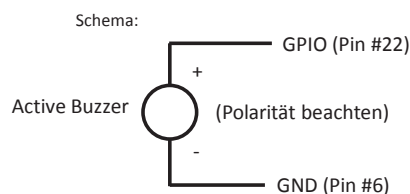
GPIO.setmode(GPIO.BOARD)
GPIO.setup(22, GPIO.OUT)
n = 0
while n < 10:
    GPIO.output(22, GPIO.HIGH)
    sleep(0.5)
    GPIO.output(22, GPIO.LOW)
    sleep(0.5)
    n = n + 1
GPIO.cleanup()
```

← Variable definieren und initialisieren
 ← Solange n kleiner als 10 ist, folgenden Block ausführen
 ← LED ein
 ← warten
 ← LED aus
 ← warten
 ← den alten Wert von n durch n + 1 ersetzen
 (keine Gleichung, sondern eine Zuweisung)

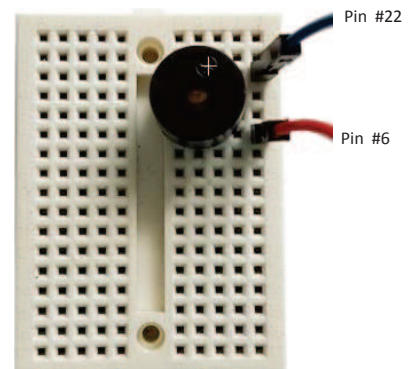
← Block einrücken (meinst 4 Leerschläge)

Digital Out: Buzzer

- Active Buzzer versus Lautsprecher
- Active Buzzer mit 2 pins: + - (Polarität beachten)
- Active Buzzer mit 3 pins: Signal - + (manchmal S und + vertauscht)
- Gleiches Programm wie vorhin (DigitalOut2.py)



Demo: Morsecode: Morse.py



Aktoren: Relais, Magnete, Ventile, Leistungselektronik

Treiber nötig (hoher Strom, hohe Spannung)

- Bipolare Transistoren
- Mosfets
- Darlington-Arrays: ULN2003

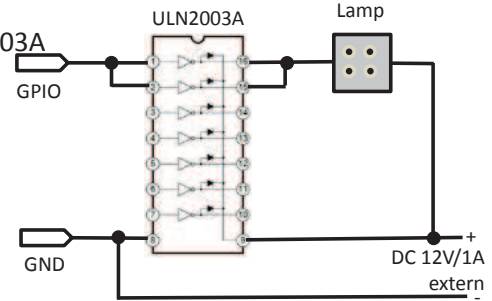
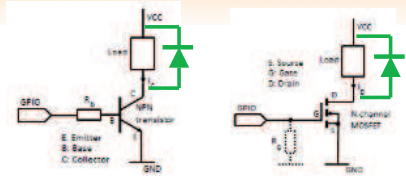
Achtung bei Spulen: Wegen Abschaltspitzen Schutzdioden nötig (Clamp Dioden)

Demo: Leistungs-LED 10V/1A (10W) über ULN2003A

- pro Ausgang 500 mA, Parallelschaltung erlaubt, hier 2 Outputs verwenden !)
- Entspricht der Lichtleistung einer 60W-Lampe
- Auf Kühlung achten

Andere typischen Anwendungen

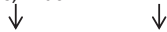
- Relaisplatine (mit Treiber oder Hat)
- Motortreiber (H-Brücke, typisch LM293D, auch mit Hat, siehe:)
- 230V Triac (siehe: Tutorial: <http://www.python-exemplary.ch/misc>)
- Funkschalter (siehe nachfolgend)



Achtung: Aufpassen mit den hohen Spannungen!

Verwendung von 433MHz-Funkschaltern

das Schalten von 230V Netzspannung ist für Mensch und Raspberry Pi gefährlich.
Klassisch: Mechanisches Relais, Triac

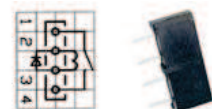
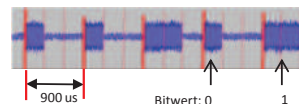
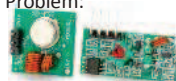


http://www.python-exemplarisch.ch/index_de.php?inhalt_links=navigation_de.inc.php&inhalt_mitte=raspi/de/miscellaneous.inc.php

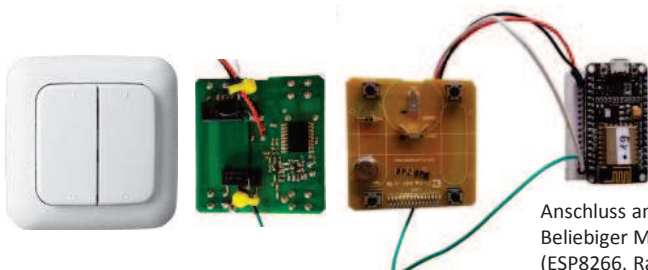
Um die Gefahr zu bannen, ist die Verwendung von Funkschalter zu empfehlen.
Dabei werden Niederspannung und Hochspannung räumlich getrennt.

Die üblichen Funkschalter arbeiten in einem der ISM-Bänder (Industrial, Scientific, Medical), oft auf 433.82 (od. 433.85, 433.90) MHz (max 2.5W, konzessionsfrei)
Modulation: Meist OOK (On-Off-Keying, CW)

Idee: Kleiner eigener TX (Sendemodul) mit Raspberry Pi tasten. Problem:
Mit Python und RPi schwierig, exaktes Timing einzuhalten
(da man die Interrupts nicht abschalten kann)



HB9ABH-Lösung: Verwendung von 5V-Reed-Relais in einem Wandschalter



Eingebaute Schutzdiode, Spule 500 Ohm
Anzug bei 2.5 V (5 mA), Abfall bei 2.0 V
Conrad: Bestell-Nr.: 1205899

Anschluss an GPIO, DigitalOut 3.3V
Beliebiger Microcontroller
(ESP8266, Raspberry Pi, Arduino, Mbed)

Digital In: Taste (Button)

Programmstruktur: Selektion (Auswahl mit if)

*Falls Bedingung erfüllt, dann führe den folgenden Block aus
sonst führe den folgenden Block aus*

```
# Button1.py

import RPi.GPIO as GPIO
from time import sleep

P_BUTTON = 16

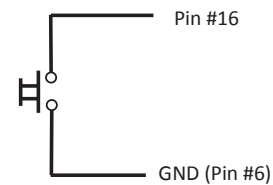
def setup():
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(P_BUTTON, GPIO.IN, GPIO.PUD_UP)

setup()
while True:
    v = GPIO.input(P_BUTTON)
    if v == GPIO.LOW:
        print LOW"
    else:
        print "HIGH"
        sleep(0.1)
```

Annotations:

- Internen Pullup verwenden oder GPIO.PUP_DOWN, GPIO.PUP_OFF
- "Input floating"
- Schalter **pollen!** (wichtiger Begriff)
- Polling Intervall sehr wichtig->anpassen

Schema:



Mit Buzzer morsen

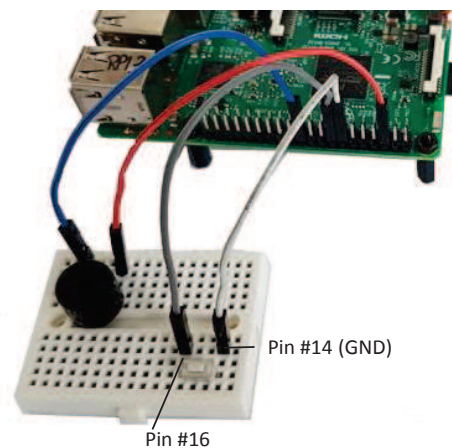
```
# Button2.py

import RPi.GPIO as GPIO
from time import sleep

P_BUTTON = 16
P_BUZZER = 22

def setup():
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(P_BUTTON, GPIO.IN,
    GPIO.PUD_UP)
    GPIO.setup(P_BUZZER, GPIO.OUT)

print "Starting"
setup()
dt = 0.1
while True:
    v = GPIO.input(P_BUTTON)
    if v == GPIO.LOW:
        GPIO.output(P_BUZZER, GPIO.HIGH)
    else:
        GPIO.output(P_BUZZER, GPIO.LOW)
    sleep(dt)
```



7-Segment Display TM1637

Ein Microcontroller-System ohne Rückmeldung sehr unprofessionell
Im einfachsten Fall genügen einige LEDs

- **Immer noch weit verbreitet, da gut lesbar und billig**

Problem: 7 LED bei gegebener Zeichen richtig ansteuern->Decoder, z.B. Hex-to-Decimal (seit 1960)
Direkt mit Microcontroller schlechte Idee, benötigt 7 GPIO/digit!
Besser: 7 Segmente + Decimal Point = 1 Byte genügt

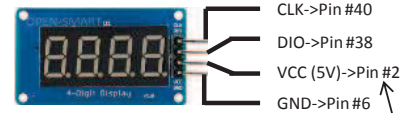
- **Man verwendet ein serielles Protokoll**

d.h. überträgt für eine vierstellige Anzeige die 4 Bytes zeitlich hintereinander. Der TM1637 verwendet ein proprietäres Protokoll mit 4 Leitungen: data (DIO) und clock (CLK), sowie zur Spannungsversorgung 3.3V/GND)

- **Kleine Library verwenden:**

API Doc: <http://www.aplu.ch/classdoc/tm1637/index.html>

Anzeigbare Zeichen: <http://www.python-exemplary.com/download/display-mapping.pdf>



Anschluss an Raspi

Achtung: Mit 5V speisen

Beispiel: Hochzählen

```
# SevenSegment1.py
from TM1637 import FourDigit

d = FourDigit(dio = 38, clk = 40)
while True:
    for n in range(10000):
        d.show(n)
```

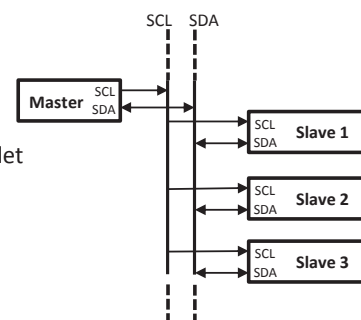
Beispiel: Text und Lauftext

```
# SevenSegment2.py
from TM1637 import FourDigit
from time import sleep

d = FourDigit(dio = 38, clk = 40)
d.show("HELO")
sleep(4)
d.scroll("cq de hb9abh")
```

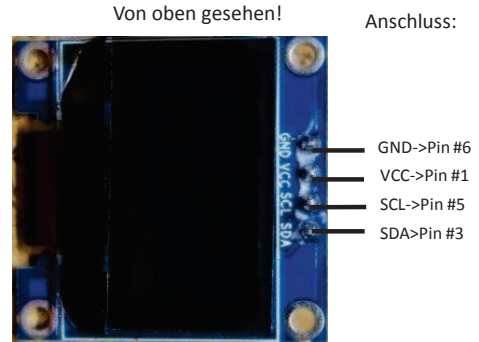
Kurze Theorie I²C-Bus (Inter-Integrated Circuit)

- Bytes werden **seriell** (eines nach dem anderen) übertragen
- Es wird eine Datenleitung (**SDA**) und eine Clockleitung (**SCL**) verwendet
SCL gibt den Takt vor, mit dem die einzelnen Bits übertragen werden
- Es gibt einen **Master** und einen oder mehrere **Slaves**
- Die Datenübertragung erfolgt über einen gemeinsamen **Bus** zwischen Master und einem Slave
- Idle-Zustand: Slaves "hören" auf ihre spezifische **Adresse**
Der Master sendet Adresse und ist dann mit diesem Slave verbunden, die anderen schweigen gefälligst
- Bei I²C werden die Adressen gewöhnlich als **7-bit Hex** angegeben, z.B. 0x4A = 010'1010.
Bei der Übertragung wird noch ein tiefstwertiges Bit dazugenommen: 0: Slave read; 1: Slave write
- Auf Raspberry Pi wird gewöhnlich die Library **smbus** verwendet



Alphanumerische/graphische Displays: OLED 128x64 pixels

- **Relative hoher Kontrast, aber doch kleines Font**
- **Relative hoher Rechenaufwand: Zeichen -> Pixels**
- **Übertragung mit I²C**
- **Verwendung einer Library `SSD1306.py` + `OLED1306.py`**
API Doc: <http://www.aplu.ch/classdoc/oled1306/index.html>



Achtung: 3.3V Versorgung wählen (wegen Pullups auf SDA/SCL)!

Es muss `SSD1306.py` und `OLED1306.py` auf den Raspberry Pi heruntergeladen werden!

```
# Oled1.py
from OLED1306 import OLED1306

disp = OLED1306()
disp.setText("Hello Python")
print "done"
```

```
# Oled2.py
from OLED1306 import OLED1306

disp = OLED1306()
disp.setText("The IP address is", 0, 10)
disp.setText("192.168.0.1", 2, 20)
```

Zeilennummer
Fontgröße

```
# Oled3.py
from OLED1306 import OLED1306
from time import sleep

disp = OLED1306()
count = 0
print "starting"
while True:
    disp.println("count: " + str(count))
    count += 1
    sleep(1)
```

Autoscroll

Analog-Digital-Wandler (ADC)

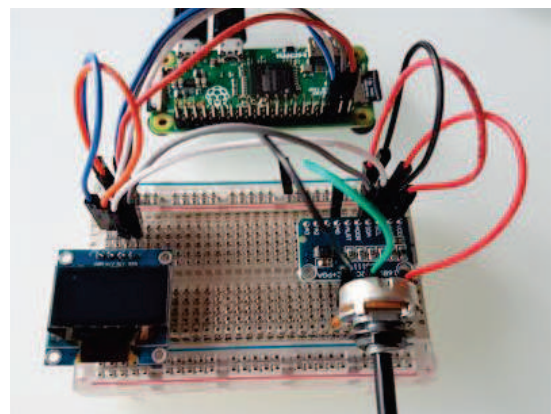
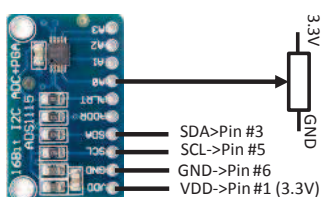
Designentscheid bei Raspberry Pi: ADC nach eigener Wahl, daher extern

- Sehr wichtiges Bauelement in allen Microcontroller-Anwendungen, oft in μC integriert
- Wichtigste Eigenschaften:
 - Bitbreite (typisch 8, 10, 12 bit, d.h. bei 16 bit ist die Auflösung unipolar 65536 Werte von 0 bis 65535, d.h. der Spannungsschritt bei 3.3V ca. 50 μV , bipolar 0..+32367, Spannungsschritt 100 μV)
 - Umwandlungsgeschwindigkeit/bzw. Taktfrequenz: typisch 100 Hz (langsam), 100 kHz (schnell), man muss die Daten aber auch noch übertragen/verarbeiten!

- Aufgabe: Spannung mit ADS1115 messen und anzeigen

Zur Sicherheit eher Potentiometer statt externes Netzgerät verwenden!

- ADS1115 features:
Auflösung 16 bit (0..+32767)
Single supply 2 - 5.5V
2 Differential or 4 Single Ended
Up to 860 samples per seconds



Ausschreiben in Console:

Modul downloaden!

```
# ADC1.py

from ADS1x15 import ADS1115
from time import sleep

adc = ADS1115()
channel = 0
gain = 1
t = 0
while True:
    v = adc.read_adc(channel,
gain)
    print t, "s:", v
    t += 0.1
    sleep(0.1)
```

Ausschreiben auf OLED:

2 Module downloaden!

```
# ADC2.py

from OLED1306 import OLED1306
from ADS1x15 import ADS1115
from time import sleep

disp = OLED1306()
disp.setNumberOfLines(5)

adc = ADS1115()
channel = 0
gain = 1
t = 0
while True:
    v = adc.read_adc(channel, gain)
    s = "t = %3.1fs - v:%2d" %(t, v)
    disp.println(s)
    t += 0.1
    sleep(0.1)
```

Test, welche I²C-Devices vorhanden sind:

```
i2cdetect -y 1
```

```
pi@raspberrypi:~$ i2cdetect -y 1
00: 0 1 2 3 4 5 6 7 8 9 a b c d e f
10: ---
20: ---
30: ---
40: --- 48 --- 3c ---
50: ---
60: ---
70: ---
ADC ADS1115 OLED SSD1306
```

Automatisches Scrollen sehr praktisch!

Messperiode grösser als 0.1 s, da auch andere Anweisungen Zeit brauchen

Appendix: Remote messen und steuern

Bluetooth

siehe Tutorial: <http://www.python-exemplarisch.ch/bluetooth>

LoRa (Funk, mit dem LoPy in Bearbeitung)

siehe <http://pycom.io/product/lopy/> und im Laufe des Jahres www.python-exemplarisch.ch

TCP

Anbindung über Ethernet/WLAN oder GSM Modul

(siehe Tutorial: <http://www.python-exemplarisch.ch/gsm>)

HTTP (Hypertext Transmission Protocol)

auf WebServer/Browser-Technologie aufgebaut.

Beispiel: Auf dem Raspberry Pi läuft eine WebServer, der eine dynamisch Webseite zeigt, beispielsweise um ein Gerät ein/auszuschalten (Heizung, usw.)

Code: WebController1.py, WebController2.py, WebController3.py

SMTP (Simple Mail Transfer Protocol)

Beispiel: Der Raspberry Pi sendet Informationen als Email, beispielsweise ein Kamerabild.

Siehe Tutorial:

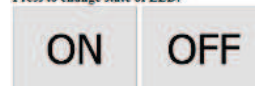
http://www.python-exemplarisch.ch/index_de.php?inhalt_links=navigation_de.inc.php&inhalt_mitte=raspi/de/gsm.inc.php

MQTT (Message Queue Telemetry Transfer)

weit verbreitet für Maschinen-zu-Maschinen-Kommunikation (M2M)

Raspberry Pi Controller

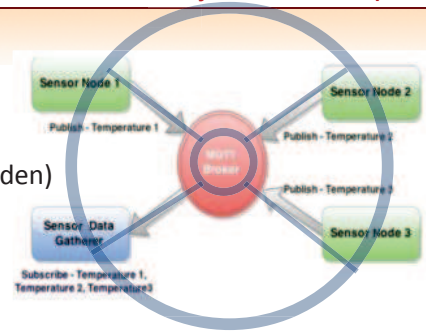
Press to change state of LED:



Current LED state: OFF

MQTT (Prinzip)

- **Prinzip: Hub-and-Spoke (Nabe-Speicher-Architektur)**
 - "Im Zentrum" ein Broker (Host, Radnabe)
 - "In der Peripherie" die Clients (über Radspeichen verbunden)
 - Clients können Themen (**Topics**) **publizieren** und **abonnieren (subscribe)**
 - Alle Transaktionen laufen über den Broker
- **MQTT arbeitet sehr sparsam** (lightweight protocol)
 - Der Overhead beträgt einige wenige Bytes (HTTP-Requests brauchen viel Daten für den Protokoll-Header)
 - Wenn z.B. nur ein Temperaturwert gesendet werden soll, werden bei HTTP für das Protokoll mehr Daten verwendet werden als für den Messwert
- **Clients benötigen keine feste IP-Adresse**
 - Nur der Broker benötigt eine feste IP-Adresse
 - Alle Clients verbinden sich zum Broker und können Push-Benachrichtigungen vom Broker erhalten (d.h. sie müssen nicht regelmässige Abfragen machen)
 - Clients können hinter einer Firewall liegen
- **Clients können sowohl Sender wie Empfänger sein (kein Client-Server-Modell)**
 - Mehrere Clients können dieselben Topics abonnieren
 - Es ist leicht, Statusrückmeldungen zu machen
 - Meldungen können auf dem Broker gespeichert sein, bis der Client sie holt



MQTT (Raspberry Pi Broker)

- **Auf der Raspibrick-Distribution befindet sich der Mosquitto MQTT-Broker**
- **Dieser kann mit der Console gestartet werden. Dazu ein Konsolenfenster öffnen mit VNC oder PuTTY oder TigerJython (Remote Terminal)**
Befehl `mosquitto -v`

```
pi@raspberrypi ~ $ mosquitto -v
1515910424: mosquitto version 1.4.14 (build date Mon,
) starting
1515910424: Using default config.
1515910424: Opening ipv4 listen socket on port 1883.
1515910424: Opening ipv6 listen socket on port 1883.
```

- **Autostart:**
Den Befehl `mosquitto -v` am Ende in die Datei `~/raspibrick/autostart.sh` einbauen, z.B. mit dem nano-Editor:
`nano ~/raspibrick/autostart.sh`

MQTT (subscribe) auf RPi mit LED

```
# MQTTLed.py

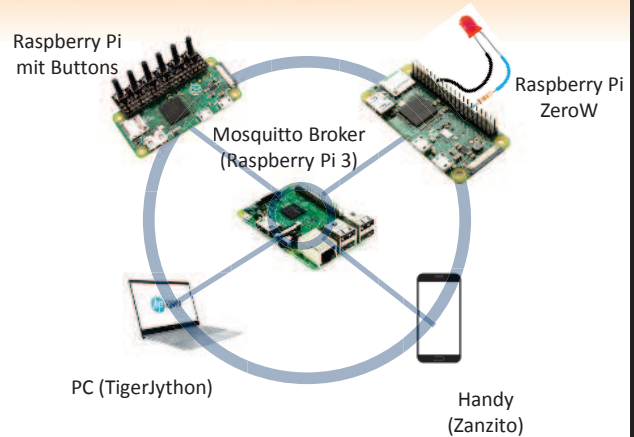
from paho.mqtt import subscribe
import RPi.GPIO as GPIO

P_LED = 22
host = "192.168.0.10"
topic = "uska/led"

def messageReceived(client, userdata, message):
    if message.payload == "aus":
        print "ausschalten"
        GPIO.output(P_LED, GPIO.LOW)
    elif message.payload == "ein":
        print "einschalten"
        GPIO.output(P_LED, GPIO.HIGH)

def setup():
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(P_LED, GPIO.OUT)

print "subscribing to host", host, "for topic", topic
setup()
subscribe.callback(messageReceived, topic, hostname = host)
```



- Client Raspi ZeroW abonniert Topic "uska/switch"
- Falls eine Meldung ankommt, wird der Callback messageReceived() aufgerufen und die Led entsprechend der "payload" (String "ein" oder "aus") geschaltet
- Einige freie Cloud MQTT Brokers

public: test.mosquitto.org, broker.hivemq.com, m2m.eclipse.org private: cloudmqtt.com

MQTT (publish) auf Raspi mit 2 Buttons

```
# MQTTPubRpi.py

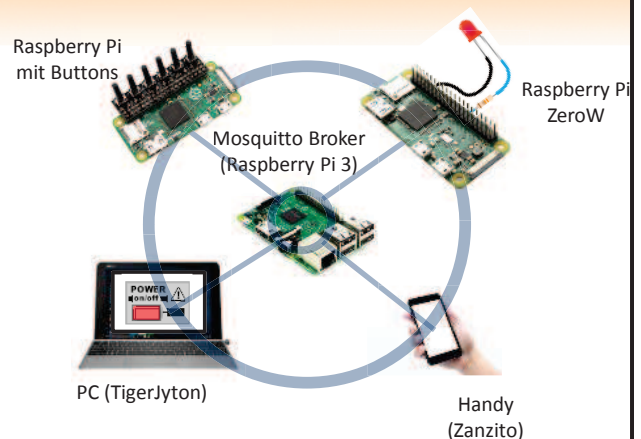
from paho.mqtt import publish
from time import sleep
import RPi.GPIO as GPIO

host = "192.168.0.10"
topic = "uska/led"

ON_BUTTON = 16
OFF_BUTTON = 18

def setup():
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(ON_BUTTON, GPIO.IN, GPIO.PUD_UP)
    GPIO.setup(OFF_BUTTON, GPIO.IN, GPIO.PUD_UP)

setup()
isOnPressed = False
isOffPressed = False
print "Press the on or off button"
while True:
    if not isOnPressed and GPIO.input(ON_BUTTON) == GPIO.LOW:
        publish.single(topic, "ein", hostname = host)
        print "einschalten"
        isOnPressed = True
    elif isOnPressed and GPIO.input(ON_BUTTON) == GPIO.HIGH:
        isOnPressed = False
    elif not isOffPressed and GPIO.input(OFF_BUTTON) == GPIO.LOW:
        publish.single(topic, "aus", hostname = host)
        print "ausschalten"
        isOffPressed = True
    elif isOffPressed and GPIO.input(OFF_BUTTON) == GPIO.HIGH:
        isOffPressed = False
    sleep(0.01) # avoid bouncing events
```



- Client publiziert Topic "uska/led" mit Werten "ein" und "aus"
- Es wird das Know-how über Buttons verwendet

MQTT auf PC (publish)

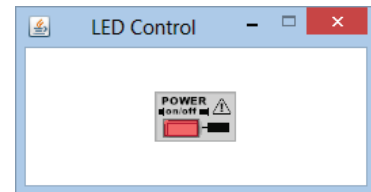
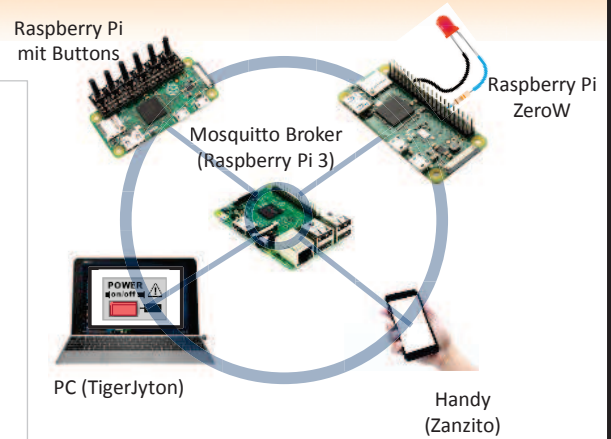
```
# Switch.py

from gturtle import *
from paho.mqtt import publish

host = "192.168.0.10"
topic = "uska/led"

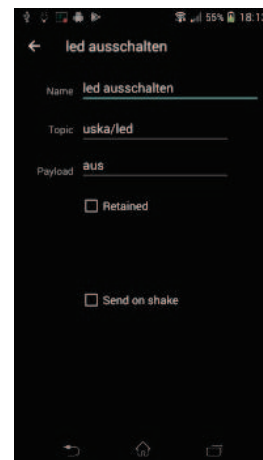
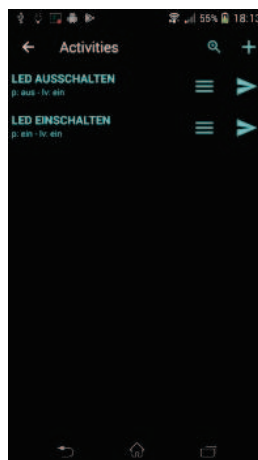
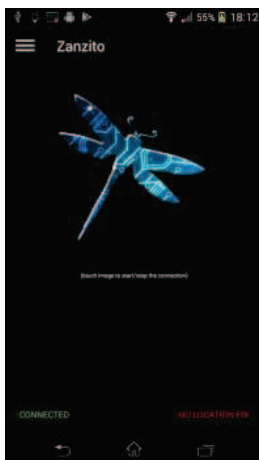
def onMouseClicked(x, y):
    global isOn
    if -30 < x < 30 and -20 < y < 20:
        if isOn:
            drawImage("sprites/switch_0.gif")
            publish.single(topic, "aus", hostname =
host)
            isOn = False
        else:
            drawImage("sprites/switch_1.gif")
            publish.single(topic, "ein", hostname =
host)
            isOn = True

isOn = False
setPlaygroundSize(250, 100)
makeTurtle(mouseClicked = onMouseClicked)
drawImage("sprites/switch_0.gif")
setTitle("LED Control")
ht()
```



- Auf dem PC ist ein Graphikfenster nötig (GUI)
- Eine einfache Möglichkeit ist die Verwendung der Turtlegrafik
- Im Maus-Callback wird das Topic publiziert

MQTT auf Handy (Android: Zanzito)



Bezugsquellen:

www.reichelt.de allgemeines Elektronik-Material

www.conrad.ch allgemeines Elektronik-Material

www.elv.ch allgemeines Elektronik-Material

www.ebay.com Suche nach einem bestimmten Artikel, Lieferung von China meist günstig

www.aliexpress.com Suche nach chinesischem Lieferanten

www.pi-shop.ch Auf Raspberry Pi und Zubehör spezialisiert

www.pimoroni.com Robotik-Material, schnelle problemlose Lieferung aus England

4tronix.co.uk Robotik-Material, schnelle problemlose Lieferung aus England

Links:

www.python-exemplarisch.ch/uskabern

www.python-exemplarisch.ch

www.brickgate.com

www.tigerjython4kids.ch

www.aplu.ch

www.jython.ch