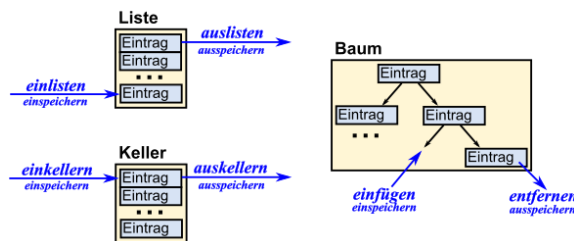


Informatik

für die Sekundarstufe I + II

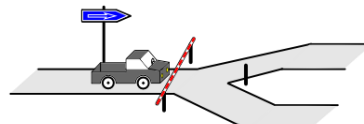
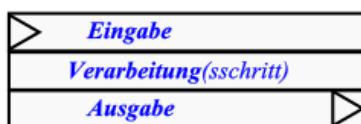
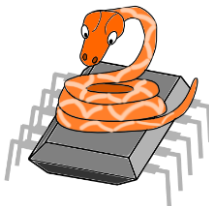
- Programmieren mit Python – Teil 3: für Experten

Autor: L. Drews



Grüner Baum-Python
(s) *Morelia viridis*
Q: de.wikipedia.org (Mwx)

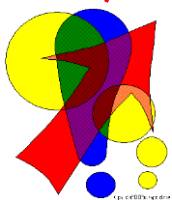
```
while eval(input("?:")) != 0:  
    print("Grüner",end='')
```



teilredigierte Version 0.11c (2026)

Legende:







mit diesem Symbol werden zusätzliche Hinweise, Tipps und weiterführende Ideen gekennzeichnet

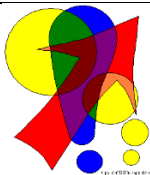
**Nutzungsbestimmungen / Bemerkungen zur Verwendung durch Dritte:**

- (1) Dieses Skript (Werk) ist zur freien Nutzung in der angebotenen Form durch den Anbieter (lern-soft-projekt) bereitgestellt. Es kann unter Angabe der Quelle und / oder des Verfassers gedruckt, vervielfältigt oder in elektronischer Form veröffentlicht werden.
- (2) Das Weglassen von Abschnitten oder Teilen (z.B. Aufgaben und Lösungen) in Teildrucken ist möglich und sinnvoll (Konzentration auf die eigenen Unterrichtsziele, -inhalte und -methoden). Bei angemessen großen Auszügen gehört das vollständige Inhaltsverzeichnis und die Angabe einer Bezugsquelle für das Originalwerk zum Pflichtteil.
- (3) Ein Verkauf in jedweder Form ist ausgeschlossen. Der Aufwand für Kopierleistungen, Datenträger oder den (einfachen) Download usw. ist davon unberührt.
- (4) Änderungswünsche werden gerne entgegen genommen. Ergänzungen, Arbeitsblätter, Aufgaben und Lösungen mit eigener Autorenschaft sind möglich und werden bei konzeptioneller Passung eingearbeitet. Die Teile sind entsprechend der Autorenschaft zu kennzeichnen. Jedes Teil behält die Urheberrechte seiner Autorenschaft bei.
- (5) Zusammenstellungen, die von diesem Skript - über Zitate hinausgehende - Bestandteile enthalten, müssen verpflichtend wieder gleichwertigen Nutzungsbestimmungen unterliegen.
- (6) Diese Nutzungsbestimmungen gehören zu diesem Werk.
- (7) Der Autor behält sich das Recht vor, diese Bestimmungen zu ändern.
- (8) Andere Urheberrechte bleiben von diesen Bestimmungen unberührt.

Rechte Anderer:

Viele der verwendeten Bilder unterliegen verschiedensten freien Lizenzen. Nach meinen Recherchen sollten alle genutzten Bilder zu einer der nachfolgenden freien Lizenzen gehören. Unabhängig von den Vorgaben der einzelnen Lizenzen sind zu jedem extern entstandenen Objekt die Quelle, und wenn bekannt, der Autor / Rechteinhaber angegeben.

public domain (pd)	Zum Gemeingut erklärte Graphiken oder Fotos (u.a.). Viele der verwendeten Bilder entstammen Webseiten / Quellen US-amerikanischer Einrichtungen, die im Regierungsauftrag mit öffentlichen Mitteln finanziert wurden und darüber rechtlich (USA) zum Gemeingut wurden. Andere kreative Leistungen wurden ohne Einschränkungen von den Urhebern freigegeben.
gnu free document licence (GFDL; gnu fdl)	
creativecommons (cc) 	 od. neu  ... Namensnennung  ... nichtkommerziell  ... in der gleichen Form  ... unter gleichen Bedingungen
Die meisten verwendeten Lizenzen schließen eine kommerzielle (Weiter-)Nutzung aus!	

**Bemerkungen zur Rechtschreibung:**

Dieses Skript folgt nicht zwangsläufig der neuen **ODER** alten deutschen Rechtschreibung. Vielmehr wird vom Recht auf künstlerische Freiheit, der Freiheit der Sprache und von der Autokorrektur des Textverarbeitungsprogramms microsoft® WORD® Gebrauch gemacht.
Für Hinweise auf echte Fehler ist der Autor immer dankbar.

Inhaltsverzeichnis:

	Seite
10. Python für spezielle Fälle.....	6
10.1. Python in Zusammenarbeit mit anderen Anwender-Programmen	6
10.2. Steuerung externer Hardware (RaspberryPi, Arduino).....	7
10.2.1. Raspberry Pi und Verwandte.....	7
10.2.1.0. Kurzbeschreibung und allgemeine Einführung zu Raspberry Pi.....	7
10.2.1.1. die GPIO-Schnittstelle	7
10.2.1.2. Steuerung über die GPIO-Schnittstelle	9
10.2.1.3. direkte Steuerung der IO-Port.....	9
10.2.1.4. Objekt-orientiertes Programmieren	10
10.2.1.5. GUI mit Tkinter	10
10.2.1.6. programmiertes Spielen mit microsoft Minecraft	12
10.2.2. Aduino und Verwandte	24
10.2.2.0. Kurzbeschreibung und allgemeine Einführung zu Arduino.....	24
10.2.2.1. Einrichtung einer Umgebung für Programmierung eines Arduino mit Python	24
10.2.2.x. Spezialfall UDOO.....	27
10.2.3. FRANZIS – Experimentierplatine mit FT232R	27
10.2.4. TI-Innovator.....	28
10.2.4.y. externe Hardware	28
RGB-Array.....	28
10.2.5. Steuerung des Calliope mini.....	30
10.3. Datenbank-Zugriff mit Python	31
10.3.1. SQLite 3	31
10.3.1.0. Verbindung herstellen.....	31
10.3.1.1. Erstellen einer Tabelle	31
10.3.1.2. Hinzufügen von Datensätzen zu einer Tabelle.....	32
10.3.1.3. Aktualisieren eines Datensatzes in einer Tabelle.....	32
10.3.1.4. Löschen eines Datensatzes aus einer Tabelle.....	32
10.3.1.5. Löschen einer Tabelle	32
10.3.1.z. Beenden der Verbindung	32
weitere Beispiele:	33
10.4. Web-Server-Anwendungen mit dem (Micro-)Framework Flask	34
10.4.0. Erzeugung einer Web-Seite mit Python (Wiederholung).....	34
10.4.1. das Framework Flask	34
10.4.2. die Flask-Erweiterung bootstrap	37
10.4.3. Programmierung der Web-Oberfläche und Darstellung von Meßwerten.....	37
10.5. Web-Applikationen mit Django.....	41
10.6. MicroPython für Microcontroller	42
10.6.x. MicroPython für micro::bit	45
weitere Editoren für microPython:.....	45
Text-basierte Systeme	45
Block-basierte Systeme	45
10.6.x. MicroPython für ESP-32-Microcontroller	46
10.6.x.0. Vorbereiten des ESP für MicroPython.....	46
10.6.x.0.1. das Tool uPyCraft	49
Installation und Beschreibung des Hilfs-Programms uPyCraft.....	50
10.6.x.0.2. Nutzung eines ESP mit microPython unter Linux	56
10.6.x.0.3. Esp-Tool.....	58
10.6.x.1. Arbeiten mit MicroPython.....	61
10.6.x.y.1. interaktiver Modus - REPL	62
10.6.x.y.2. interaktiver und Internet-fähiger Modus - WebREPL	63
10.6.x.y.3. "Autostart"-Modus	64
10.6.x.4. elementare Programmierung mit MicroPython	67

10.6.x.4.1. Ausgaben	67
10.6.x.4.2. Variablen, Zuweisungen und Berechnungen	70
10.6.x.4.3. Eingaben	71
10.6.x.4.4. Alternativen, Verzweigungen.....	72
10.6.x.4.5. Wiederholungen, Schleifen	72
10.6.x.4.6. eingebaute und mitgelieferte Funktionen	72
10.6.x.5. klassische Programmierung mit MicroPython.....	72
10.6.x.5.1. Listen und Listen-Verarbeitung	72
10.6.x.5.2. Wörterbücher, Dictionary's	72
10.6.x.5.3. Lesen und Schreiben von Dateien, Datei-Verarbeitung.....	72
10.6.x.5.4.....	73
10.6.x.6. spezielle Programmierung mit MicroPython	73
10.6.x.4. weitere spezielle Programm-Beispiele und -Schnipsel	73
10.6.x.5. spezielle Module für ESP-32-Microcontroller.....	85
10.6.x.5.1. Modul "machine"	85
Deep-sleep-Modus ().....	85
RTC (realtime clock).....	85
Zähler / Timer	86
Pin's / GPIO	86
PWM (pulse width modulation)	87
ADC (analog to digital conversion).....	87
SPI-Bus (serial peripheral interface)	88
I2C-Bus.....	89
OneWire-Treiber ()	89
LED-Leisten bzw. -Ringe (NeoPixel).....	90
Touch-Eingabe (capacitive touch).....	90
DHT (Umweltsensoren, Temperatur-Luftfeuchte-Sensor)	91
10.6.x.5.2. Modul "esp".....	91
10.6.x.5.3. Modul "esp32".....	92
10.6.x.5.4. Modul "network"	92
10.6.x.5.5. Modul "time".....	93
10.6.x.y. Sprach-Elemente vom MicroPython (Kurz-Übersicht / Spicker).....	94
(formatierte) Ausgabe:.....	94
Verzweigung:.....	94
Schleifen:.....	94
10.7. Python auf und mit Taschenrechnern / spezieller Hardware	95
10.7.x. Casio-Rechner.....	95
FX-CG50 95	
10.7.x. Texas Instruments-Rechner.....	97
TI-Nspire CXII-T CAS	97
Formeln programmieren.....	98
Nutzung des TI-Innovator	99
Steuern des TI-Rover	99
TI-84 Plus.....	99
Nutzung des mirco::bit.....	100
Vorbereitung des Taschenrechner's	100
10.7.x. Miniroboter Edison (Microbric)	101
10.8. Python und Data Science	103
der Titanic-Daten-Bestand.....	104
10.9. Python und Künstliche Intelligenz	109
10.9.x. Entscheidungs-Bäume.....	109
10.9.x. Korrelation und Regression	109
10.9.x. maschinelles Lernen.....	110
10.10. Python kommuniziert in Discord.....	111
10.10.0. Allgemeines und Vorbereitung.....	111
10.10.2. erste Kommunikations-Versuche	112
10.10.3. Programmierung eines Bot's.....	117
11. Üben, üben und nochmals üben	121

11.x. Aufgaben aus der Abiturprüfung Informatik MV	122
11.x.y. Abitur 2010	122
11.x. Aufgaben der Landesolympiade Informatik MV	122
11.x.y. 2014/2015.....	122
11.x.y.z. Sekundarstufe II	122
Literatur und Quellen:	123

10. Python für spezielle Fälle

10.1. Python in Zusammenarbeit mit anderen Anwender- Programmen

interessante Links:

<https://automatetheboringstuff.com/> (online-Version des Buches: AL SWEIGART: Automate the Boring Stuff with Python – Practical Programming for Total Beginners)

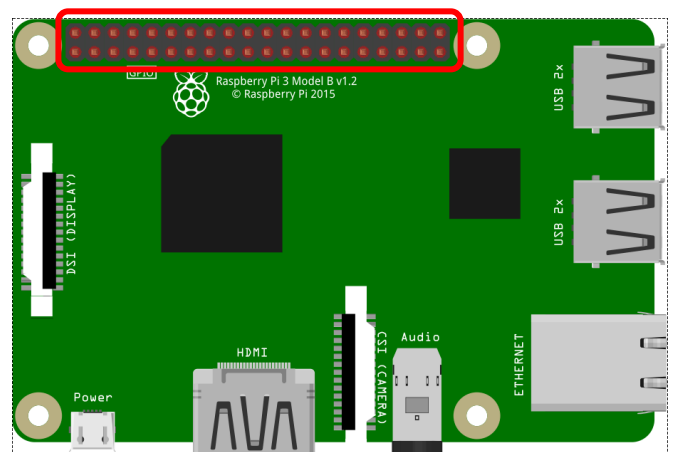
10.2. Steuerung externer Hardware (RaspberryPi, Arduino)

10.2.1. Raspberry Pi und Verwandte

10.2.1.0. Kurzbeschreibung und allgemeine Einführung zu Raspberry Pi

10.2.1.1. die GPIO-Schnittstelle

GPIO ist eine 40-polige Anschluß-Leiste mit verschiedenen Ein- und Ausgängen zum Board. Dient dem Anschluß von Zusatz-Platinen (Shield's) oder von elektrischen / elektronischen Schaltungen.



fritzing

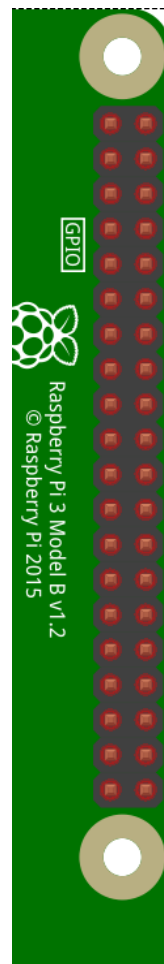


Achtung:

Die Benutzung der Pin's muss exakt eingehalten werden. Ein Wechsel auf andere Pin's ist nur mit genauer Vorüberlegung möglich.

Ein Verwechslung von Pin's oder Polungen kann zur Zerstörung von Bauelementen und / oder der Raspberry-Platine führen.

Schaltungen sollten vor der Benutzung immer noch einmal kontrolliert werden (4-Augen-Prinzip empfohlen!). Erst wenn alles übereinstimmt, dann als Letztes Masse oder Spannung-Pin einstecken!



Q: Fritzing; bearb.: dre

	innen	außen	
1	3,3V	5V	2
3	2	5V	4
5	3	Gnd	6
7	4	14	8
9	Gnd	15	10
11	17	18	12
13	27	Gnd	14
15	22	23	16
17	3,3V	24	18
19	10	Gnd	20
21	9	25	22
23	11	8	24
25	Gnd	7	26
27	ID_SD	ID_SC	28
29	5	Gnd	30
31	6	12	32
33	13	Gnd	34
35	19	16	36
37	26	20	38
39	Gnd	21	40

Pin

Name

10.2.1.2. Steuerung über die GPIO-Schnittstelle

```
# LED an GPIO23 blinken
import RPi.GPIO as GPIO      # Bibliothek für GPIO-Steuerung
import time                  # Bibliothek für Zeitsteuerung

GPIO.setmode(GPIO.BCM)      # GPIO-Namen verwenden
pin=23                       # GPIO-Pin
GPIO.setup(pin, GPIO.OUT)    # GPIO-Pin auf Ausgabe

zeit=0.5                     # Hell-Dunkel-Wartezeit
while True:                  # Endlos-Schleife
    GPIO.output(pin, GPIO.HIGH) # Pin ein
    time.sleep(zeit)
    GPIO.output(pin, GPIO.LOW)  # Pin aus
    time.sleep(zeit)
```

```
# LED per Taster EIN und AUS
import RPi.GPIO as GPIO      # Bibliothek für GPIO-Steuerung
import time                  # Bibliothek für Zeitsteuerung

GPIO.setmode(GPIO.BCM)      # GPIO-Namen verwenden
pinAus=23                    # GPIO-Pin
pinEin=24                     # GPIO-Pin
GPIO.setup(pinAus, GPIO.OUT) # GPIO-Pin auf Ausgabe
GPIO.setup(pinEin, GPIO.IN)  # GPIO-Pin auf Eingabe

zeit=0.5                     # Hell-Dunkel-Wartezeit
for i in range(3):           # 3x wiederholen (blinken)
    GPIO.output(pinAus, GPIO.HIGH) # LED an
    time.sleep(zeit)             # warten
    GPIO.output(pinAus, GPIO.LOW)  # LED aus
    time.sleep(zeit)             # warten

while True:                  # Endlosschleife
    if GPIO.input(pinEin) == 0:    # Abfrage Eingabe-Pin = geschlossen
        GPIO.output(pinAus, GPIO.LOW) # Ausschalten
    else:
        GPIO.output(pinAus, GPIO.HIGH) # Einschalten
```

die Endlos-Schleifen lassen sich mit [Strg] + [c] abbrechen

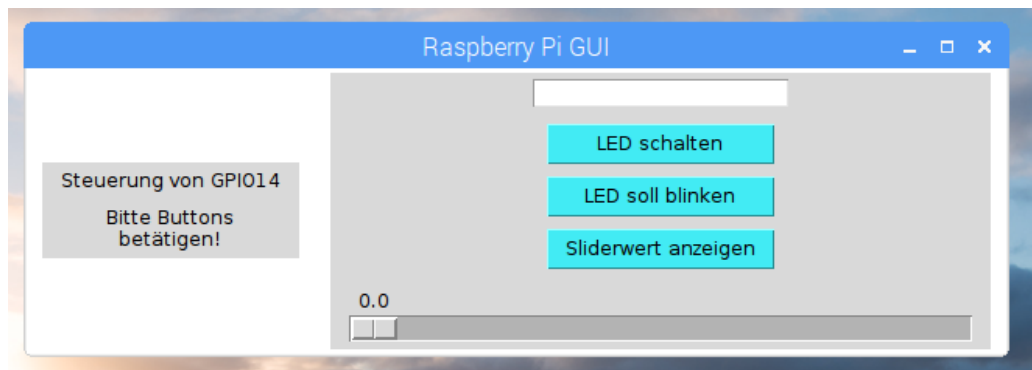
10.2.1.3. direkte Steuerung der IO-Port

10.2.1.4. Objekt-orientiertes Programmieren

eine Sensor-Klasse

→ <http://www.forum-raspberrypi.de/Thread-tutorial-einfuehrung-in-objektorientierte-programmierung-mit-python>

10.2.1.5. GUI mit Tkinter



Q: http://www.elektronik.nmp24.de/?Python-Programmierung:GUI_mit_tkinter

```
#GUI für das Ein- und Ausschalten einer LED an GPIO14
from tkinter import * #Grafikbibliothek
root = Tk() # Fenster erstellen
root.wm_title("Raspberry Pi GUI") # Fenster Titel
root.config(background = "#FFFFFF") # Hintergrundfarbe des Fensters
#GPIO- und time-Bibliothek:
import RPi.GPIO as GPIO
import time
#festlegen, dass GPIO-Nummern verwendet werden:
GPIO.setmode(GPIO.BCM)
#GPIO14 als Ausgang:
GPIO.setup(14, GPIO.OUT)

# Hier werden zwei Frames erzeugt:
leftFrame = Frame(root, width=200, height = 400) # Frame initialisieren
leftFrame.grid(row=0, column=0, padx=10, pady=3) # Relative Position und Seiten-
abstand (padding) angeben
# Hier kommen die Elemente des linken Frames rein
leftLabel1 = Label(leftFrame, text="Steuerung von GPIO14")
leftLabel1.grid(row=0, column=0, padx=10, pady=3)
leftLabel2 = Label(leftFrame, text="Bitte Buttons\nbetätigen!")
leftLabel2.grid(row=1, column=0, padx=10, pady=3)

rightFrame = Frame(root, width=400, height = 400)
rightFrame.grid(row=0, column=1, padx=10, pady=3)

# Hier kommen die Elemente des rechten Frames rein
# callback1 ist der Handler von Button B1

def callback1():
    varLEDStatus = GPIO.input(14)
    if varLEDStatus == 0:
        GPIO.output(14, GPIO.HIGH)
```

```

        E1.delete(0, END)
        E1.insert(0, "LED ist eingeschaltet")
    else:
        GPIO.output(14, GPIO.LOW)
        E1.delete(0, END)
        E1.insert(0, "LED ist ausgeschaltet")

def callback2():
    for i in range(5):
        GPIO.output(14, GPIO.HIGH)
        time.sleep(0.5)
        GPIO.output(14, GPIO.LOW)
        time.sleep(0.5)

def callback3():
    print (Slider.get())
    E1.delete(0, END)
    E1.insert(0, "Slider = ")
    E1.insert(12, Slider.get())

buttonFrame = Frame(rightFrame)
buttonFrame.grid(row=1, column=0, padx=10, pady=3)

E1 = Entry(rightFrame, width=20)
E1.grid(row=0, column=0, padx=10, pady=3)

B1 = Button(buttonFrame, text="LED schalten", bg="#42ebf4", width=15, com-
mand=callback1)
B1.grid(row=1, column=0, padx=10, pady=3)
B2 = Button(buttonFrame, text="LED soll blinken", bg="#42ebf4", width=15, com-
mand=callback2)
B2.grid(row=2, column=0, padx=10, pady=3)
B3 = Button(buttonFrame, text="Sliderwert anzeigen", bg="#42ebf4", width=15,
command=callback3)
B3.grid(row=3, column=0, padx=10, pady=3)

Slider = Scale(rightFrame, from_=0, to=100, resolution=0.1, orient=HORIZONTAL,
length=400)
Slider.grid(row=3, column=0, padx=10, pady=3)

root.mainloop() # GUI wird upgedated. Danach keine Elemente setzen
Q: http://www.elektronik.nmp24.de/?Python-Programmierung:GUI\_mit\_tkinter

```

Links:

<https://tutorials-raspberrypi.de>

10.2.1.6. programmiertes Spielen mit microsoft Minecraft

Programm-Beispiele (gelbbraunlich (hell oliv)) stammen aus dem conrad / Franzis Adventskalender Programmieren mit Minecraft 2018

Python und Minecraft stellen auf dem Raspberry Pi eine besonders Preis-günstige Kombination dar. Praktisch hat man nur die Hardware-Kosten für den Pi und die notwendigen Zubehör-Teile. Weder Python noch Minecraft kostet auf dem Pi Geld. Microsoft hat für den Pi eine kostenlose Variante von Minecraft spendiert. Für erste Erfahrungen ist das schon mehr als genug. Wir können alle wesentlichen Funktionen von Mindecraft programmieren und damit unser eigenes Spiel erstellen.

Für Python-Anfänger ist das Objekt-orientierte sicher eine sehr große Herausforderung. Aber die schon fortgeschrittenen Programmierer mit Grundkenntnissen in OOP werden sicher sehr schnell zurecht kommen.

Position des Spieler auswerten

Die GPIO verfügt über 16 (???) Digital-Ausgänge. Das heißt diese können durch Programme AN oder AUS geschaltet werden. Für die meisten Schaltungen bedeutet das für AN liegt eine Spannung von 5 V an und bei AUS ist es 0 V. An den digitalen Ausgängen sind nur diese beiden Zustände zugelassen. Zwischen-Größen sind nicht möglich. Dies ist nur an analogen Port's möglich (→).

In diesem Projekt soll die Position des Spieler in der Würfel-Welt erfasst und ausgewertet werden. Wenn der Spieler einen bestimmten Bereich in der Minecraft-Welt erreicht hat, dann soll eine zweite LED (rot) leuchten.

an Pin 23 und Pin 25 muss jeweils eine LED entweder mit integriertem Vorwiderstand oder ein solcher in Reihe mit einer normalen LED geschaltet werden (das lange Beinchen der LED's steht für den Plus-Pol und kommt an den Pin

das andere (kürzere) Beinchen ist auf Minus bzw. Masse zu schalten

unbedingt die richtige Polung und auch die Pin-Auswahl beachten, ansonsten könnten die Bauelemente oder gar der Pi beschädigt werden

am Besten Aufbau auf einem SteckBrett

```
#!/usr/bin/python

import mcpi.minecraft as minecraft
import RPi.GPIO as GPIO

mc = minecraft.Minecraft.create()

GPIO.setmode(GPIO.BCM)
GPIO.setup(23, GPIO.OUT)
GPIO.setup(25, GPIO.OUT)

while True:

    p = mc.player.getTilePos()

    if (p.x==8 and p.z>=3 and p.z <=4):
```

notw. Zeile für Kommandozeilenstart von Python-Programmen

Bibliothek für die Verbindung mit Minecraft
Bibliothek für die Ansteuerung der GPIO-Schnittstelle

Erstellen eines Minecraft-Objektes mit dem Namen mc

Setzen der Pin-Namen lt. BCM

Initialisierung des Port 23 für Ausgabe

Initialisierung des Port 25 für Ausgabe

Endlos-Schleife (wird durch Spiel unterbrochen)

Holen der akt. Spieler-Position und Speichern in p

Prüfen ob Spieler in einem bestimmten Be-

```
GPIO.output(23, True)
GPIO.output(25, False)
else:
    GPIO.output(25, True)
    GPIO.output(23, False)
```

reich ist (hier x=8 und z zwischen 3 und 4)
Wenn ja, **dann** Ausgabe 1 (Spannung) auf
Pin 23 und 0 (keine Spannung) auf Pin 25
Sonst
Ausgabe 1 (Spannung) auf Pin 25 und 0 (kei-
ne Spannung) auf Pin 23

an Pin 23 und Pin 25 muss jeweils eine LED entweder mit integriertem Vorwiderstand oder ein solcher in Reihe mit einer normalen LED geschaltet werden (das lange Beinchen der LED's steht für den Plus-Pol und kommt an den Pin
das andere (kürzere) Beinchen ist auf Minus bzw. Masse zu schalten

unbedingt die richtige Polung und auch die Pin-Auswahl beachten, ansonsten könnten die Bauelemente oder gar der Pi beschädigt werden

am Besten Aufbau auf einem SteckBrett

Spiel muss vor dem Starten des Python-Programmes laufen
ev. Warn-Hinweise wegen des Zugriffs auf die GPIO können ignoriert werden, Programm läuft schon im Hintergrund

Material eines Block's auswerten

```
#!/usr/bin/python

import mcpi.minecraft as minecraft
import RPi.GPIO as GPIO

import time

mc = minecraft.Minecraft.create()

rot = 0
gelb = 1
gruen = 2
Ampel = [18,23,25]

GPIO.setmode(GPIO.BCM)
GPIO.setup(Ampel[rot], GPIO.OUT, ↵
initial = True)
GPIO.setup(Ampel[gelb], GPIO.OUT, ↵
initial = False)
GPIO.setup(Ampel[gruen], GPIO.OUT, ↵
initial = False)

try:

    while True:

        p = mc.player.getTilePos()

        mat = mc.getBlock(p.x,p.y-1,p.z)

        if mat == 98:

            GPIO.output(Ampel[gelb], True)
            time.sleep(0.6)
            GPIO.output(Ampel[rot], False)
            GPIO.output(Ampel[gelb], False)
            GPIO.output(Ampel[gruen], True)
            time.sleep(2)
            GPIO.output(Ampel[gruen], False)
            GPIO.output(Ampel[gelb], True)
            time.sleep(0.6)
            GPIO.output(Ampel[gelb], False)
            GPIO.output(Ampel[rot], True)
            time.sleep(2)
except KeyboardInterrupt:

    GPIO.cleanup()
```

notw. Zeile für Kommandozeilenstart von Python-Programmen
Bibliothek für die Verbindung mit Minecraft
Bibliothek für die Ansteuerung der GPIO-Schnittstelle
Bibliothek mit Zeit- Funktionen einbinden
Erstellen eines Minecraft-Objektes mit dem Namen mc

Definition von Variablen (quasi als Konstanten)

Liste der für die Ampel (angeschlossene LED's benutzte Port's
Setzen der Pin-Namen lt. BCM
Initialisierung des Port lt. Ampel-Liste für Ausgabe
... für Gelb

... für Grün

nachfolgender Block wird ausprobiert (Abbruch folgt später!)
Endlos-Schleife (wird durch Spiel unterbrochen)
Holen der akt. Spieler-Position und Speichern in p
Material des Block's unter dem Spieler auslesen
Prüfen ob Material den Code 98 hat
Wenn ja, **dann** übliche Lichtschaltung einer Ampel
kurze Pause 0,6 s

lange Pause 2 s

Abbruch des probierten Block's durch eine Tastatur-Unterbrechung (praktisch Tastendruck)
GPIO-Port-Belegung / -Benutzung löschen

↵ Zeilenumbruch nur für die Darstellung des Quelltextes (Layout), bedeutet, dass in der Zeile weitergeschrieben wird

Schlag mit dem Schwert auswerten

```
#!/usr/bin/python

import mcpi.minecraft as minecraft
import RPi.GPIO as GPIO

mc = minecraft.Minecraft.create()
LED = [18,23,25]

GPIO.setmode(GPIO.BCM)
for i in LED:
    GPIO.setup(i,GPIO.OUT,initial=False)

try:

    while True:

        for hit in mc.events.pollBlockHits():
            bl = mc.getBlockWithData(hit.pos.x, hit.pos.y, hit.pos.z)
            if bl.id == minecraft.GLOWSTONE_BLOCK.id:
                for i in LED:
                    GPIO.output(i,True)
                    time.sleep(0.05)
                    GPIO.output(i,False)

except KeyboardInterrupt:
    GPIO.cleanup()
```

notw. Zeile für Kommandozeilenstart von Python-Programmen
Bibliothek für die Verbindung mit Minecraft
Bibliothek für die Ansteuerung der GPIO-Schnittstelle
Erstellen eines Minecraft-Objektes mit dem Namen mc

Setup der GPIO-Schnittstelle im BCM-Modus und auf Ausgaben; Benutzung der Port's aus der LED-Liste

nachfolgender Block wird ausprobiert (Abbruch folgt später!)
Endlos-Schleife (wird durch Spiel unterbrochen)
für alle Ereignisse (hits) aus der aktuellen Event-Liste dieses Block's
Erfassen der aktuellen Block-Daten

Wenn der akt. Block von dem angegebenen Material ist, **dann** ...
lasse die die LED's hintereinander

kurz
aufblitzern

Abbruch durch eine Tastatur-Betätigung
dann noch GPIO nullen

auf externes Ereignis (z.B. Tasten-Druck) reagieren (und Blöcke erstellen)

```
#!/usr/bin/python

import mcpi.minecraft as minecraft
import mcpi.block as block
import RPi.GPIO as GPIO
import time

mc = minecraft.Minecraft.create()

t1 = 8
GPIO.setmode(GPIO.BCM)
GPIO.setup(t1, GPIO.IN, GPIO.PUD_DOWN)

try:
    while True:
        if GPIO.input(t1)==True:
            p = mc.player.getTilePos()
            mc.setBlocks(p.x-1, p.y, ↵
p.z-1, p.x+1, p.y, p.z+1, block.SAND)
            mc.player.setPos(p.x, p.y+1, p.z)

            time.sleep(0.2)
except KeyboardInterrupt:
    GPIO.cleanup()
```

notw. Zeile für Kommandozeilenstart von Python-Programmen
Bibliothek für die Verbindung mit Minecraft

Bibliothek für die Ansteuerung der GPIO-Schnittstelle

Erstellen eines Minecraft-Objektes mit dem Namen mc

Festlegen des Port's für die Eingabe (Taste)

...
Einschalten des Pull-down-Widerstandes auf dem Rasp Pi
...

Wenn die Taste gedrückt ist, **dann** ...

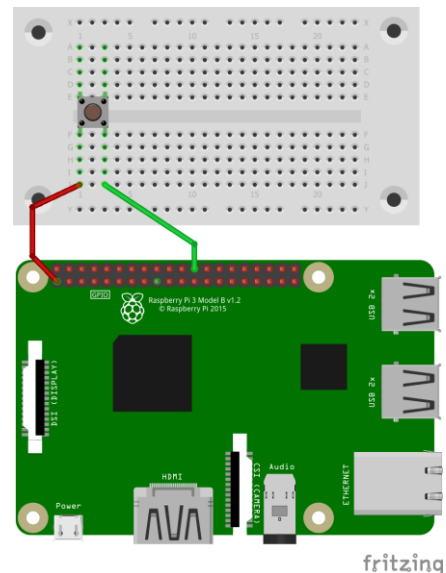
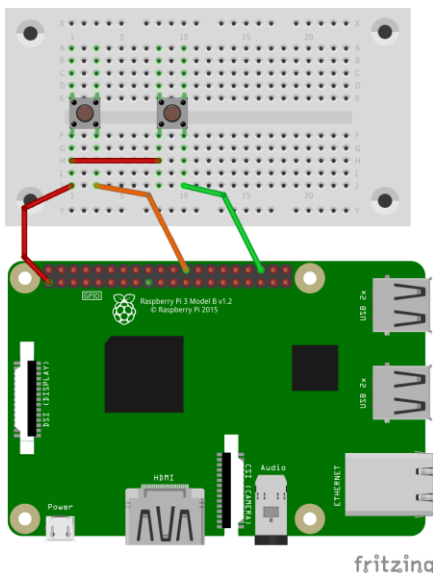
Erstellen von neuen Blöcken als 3x3-Fläche unter dem Spieler (Material ist Sand)

Korrektur der Spieler-Position auf der Sand-Fläche

...

ein Taster wird zwischen "3,3V" (innen Pin 1) und Port 8 (außen 12. Pin (Nr. 24)) geschaltet
s.a. Bau-Plan rechts

an einem weiteren Pin lässt sich ein weiterer Taster betreiben (siehe Aufbau-Plan unten)
diesem können wir unabhängig vom ersten Taster eigene Befehle zuordnen



Es werden die Port's 16 und 8 benutzt, die liegen auf den GPIO-Pin's 36 und 24. Die roten Draht-Brücken sind für den Masse-Kontakt beider Taster zuständig.

Im folgenden Programm soll 3 Blöcke vor der Spieler-Position ein Baum gebaut und mit der anderen Taste soll der Baum wieder entfernt werden. Dazu ändert man den Typ eines Block's einfach auf Luft (Code: 0).

```
#!/usr/bin/python

import mcpi.minecraft as minecraft
import mcpi.block as block
import RPi.GPIO as GPIO
import time

mc = minecraft.Minecraft.create()

t1 = 8
t2 = 16
GPIO.setmode(GPIO.BCM)
GPIO.setup(t1,GPIO.IN, GPIO.PUD_DOWN)
GPIO.setup(t2,GPIO.IN, GPIO.PUD_DOWN)

try:
    while True:
        if GPIO.input(t1)==True:
            p = mc.player.getTilePos()
            mc.setBlock(p.x+3, p.y, p.z, ↵
            block.WOOD)
            mc.setBlocks(p.x+2, p.y+1, ↵
            p.z-1, p.x+4,p.y+2, p.z+1, ↵
            block.LEAVES)

            if GPIO.input(t2)==True:
                p = mc.player.getTilePos()
                mc.setBlock(p.x+3, p.y, p.z, ↵
                block.AIR)
                mc.setBlocks(p.x+2, p.y+1, ↵
                p.z-1, p.x+4,p.y+2, p.z+1, block.AIR)

            time.sleep(0.2)
except KeyboardInterrupt:
    GPIO.cleanup()
```

...

Festlegen des Port's für die Eingabe (Taste1)
und für Taste 2

...

Einschalten der Pull-down-Widerstände auf
dem Rasp Pi

...

Wenn die Taste1 gedrückt ist, dann ...

Erstellen von neuen Blöcken als 3x3-Fläche
unter dem Spieler (Material ist Sand)
Korrektur der Spieler-Position auf der Sand-
Fläche

...

Material-Code	dt. Bezeichnung	ID	Material-Code	dt. Bezeichnung	ID
AIR	Luft	0	STONE_SLAB		44
STONE	Stein	1	BRICK_BLOCK		45
GRASS	Gras	2	TNT	Sprengstoff	46
DIRT		3	BOOKSHELF		47
COBBLESTONE		4	MOSS_STONE		48
WOOD_PLANKS		5	OBSIDIAN	Obsidian	49
SAPLING		6	TORCH		50
BEDROCK		7	FIRE	Feuer	51
WATER_FLOWING		8	STAIRS_WOOD		53
WATER		8	CHEST		54
WATER_STATIONARY		9	DIAMAND_ORE	Diamand-Erz	56
LAVA_FLOWING		10	DIAMAND_BLOCK		57
LAVA		10	CRAFTING_TABLE		58
LAVA_STATIONARY		11	FARMLAND	Ackerland	60
SAND	Sand	12	FURNACE_INACTIVE		61
GRAVEL		13	FURNACE_ACTIVE		62
GOLD_ORE	Gold-Erz	14	DOOR_WOOD	Holztür	64
IRON_ORE	Eisen-Erz	15	LADDER		65
COAL_ORE		16	STAIRS_COBBLESTONE		67
WOOD	Holz	17	DOOR_IRON	Eisentür	71
LEAVES		18	REDSTONE_ORE		73
GLASS		20	SNOW	Schnee	78
LAPIS_LAZULI_ORE		21	ICE	Eis	79
LAPIS_LAZULI_BLOCK		22	SNOW_BLOCK	Schnee-Block	80
SANDSTONE	Sandstein	24	CACTUS	Kaktus	81
BED		26	CLAY		82
COBWEB		30	SUGAR_CANE		83
GRASS_TALL		31	FENCE		85
WOLL	Wolle	35	GLOWSTONE_BLOCK		89
FLOWER_YELLOW		37	REDROCK_INVISIBLE		95
FLOWER_CYAN		38	STONE_BRICK		98
MUSHROOM_BROWN		39	GLASS_PANE		102
MUSHROOM_RED		40	MELON	Melone	103
GOLD_BLOCK		41	FENCE_GATE		107
IRON_BLOCK		42	GLOWING_OBSIDIAN		246
STONE_SLAB_DOUBLE		43	NETHER_REACTOR_CORE		247

Q: Begleitheft zu: conrad / Franzis Adventskalender Programmieren mit Minecraft 2018; erw. drews

eine LED dimmen (frequentes Signal ausgeben)

Problem: LED's lassen sich nicht wie Glühlampen dimmen. Bei diesen kann man durch eine größere oder kleiner Betriebs-Spannung die Leuchtstärke verändern. LED's brauchen einen minimale Spannung – quasi zum Zünden – und die Betriebsspannung kann auch nicht so einfach variiert werden, da es schnell zur Überlastung (Zerstörung) kommen kann.

Die analogen Port's des Rasp Pi (GPIO) sind nicht so auch nicht für eine Steuerung von LED's geeignet.

Der technische Ausweg ist ein Verändern von Leucht- und Dunkel-Phasen. Damit das ständige AN und AUS nicht als Blinken erkannt wird, benutzt man eine Arbeits-Frequenz von hier 50 Hz.

```
#!/usr/bin/python

import mcpi.minecraft as minecraft
import RPi.GPIO as GPIO

mc = minecraft.Minecraft.create()

LED = 25
GPIO.setmode(GPIO.BCM)
GPIO.setup(LED, GPIO.OUT, initial=False)

pwm = 0
l = GPIO.PWM(LED, 50)
l.start(pwm)

try:
    while True:
        p = mc.player.getTilePos()
        if p.z >= 5 and p.z <= 15:
            pwm = 10 * (15 - p.z)

            l.ChangeDutyCycle(pwm)
            time.sleep(0.1)

except KeyboardInterrupt:
    l.stop()
    GPIO.cleanup()
```

notw. Zeile für Kommandozeilenstart von Python-Programmen
 Bibliothek für die Verbindung mit Minecraft
 Bibliothek für die Ansteuerung der GPIO-Schnittstelle
 Erstellen eines Minecraft-Objektes mit dem Namen mc
 LED an Port 25
 Initialisierung der GPIO

Tast-Verhältnis auf Null setzen ()
 Erzeugen eines PWM-Objektes als l
 Starten der Ausgabe

Wenn bestimmte Z-Position benutzt wird, dann wird Tast-Verhältnis schrittweise (in Abhängigkeit von der Z-Position) erhöht
 Setzen des neuen Ausgabe-Signals
 System-Wartezeit

...
 Ausgabe am Port wird beendet

Ein Tast-Verhältnis von 0 bedeutet bei einem Rechteck-Signal einen minimalen AN-Teil. Praktisch ist der Pegel ständig auf 0 V gelegt.

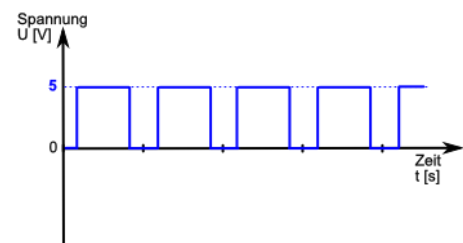
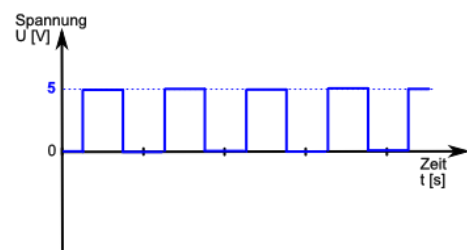
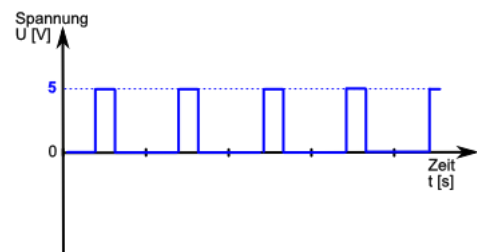
Sind die AN-Phasen nur kurz (s.a. oberes Diagramm), dann blitzt die LED nur kurz auf. Da dies mit einer Frequenz von 50 Hz (so initialisiert im Programm) passiert, sehen wir das als sehr schwaches Leuchten (Licht-Summe).

Bei einem Tastverhältnis von 50 sind AN- und AUS-Teil jeweils 50% - also gleichlang (s.a. mittlere Abb.).

Physikalisch entspricht das der halben Lichtstärke. Da unser optischer Sinn aber nicht linear funktioniert, erkennen wir das immer noch als relativ dunkel.

Je länger der AN-Teil wird, umso heller wird uns die LED erscheinen.

Setzt man den PWM-Wert auf 100, dann ist ein Dauer-AN – also ständig 5 V – am betreffenden Port anliegend. Damit ist die Leuchtstärke ausgeschöpft (zumindestens im regulären Bereich).



einen digitalen Pegel (/ Sensor-Kontakt) auswerten

```
#!/usr/bin/python

import mcpi.minecraft as minecraft
import RPi.GPIO as GPIO

mc = minecraft.Minecraft.create()

LED = 18
K1 = 20
GPIO.setmode(GPIO.BCM)
GPIO.setup(LED, GPIO.OUT, initial=False)
GPIO.setup(K1, GPIO.IN)

try:
    while True:
        if GPIO.input(K1) == False:
            GPIO.output(LED, True)
            mc.setBlocks(3,2,4,3,2,5,↵
            block.GOLD_ORE)
        else:
            GPIO.output(LED, False)
            mc.setBlocks(3,2,4,3,2,5,↵
            block.COAL_ORE)
        time.sleep(0.05)
except KeyboardInterrupt:
    GPIO.cleanup()
```

notw. Zeile für Kommandozeilenstart von Python-Programmen
Bibliothek für die Verbindung mit Minecraft
Bibliothek für die Ansteuerung der GPIO-Schnittstelle
Erstellen eines Minecraft-Objektes mit dem Namen mc

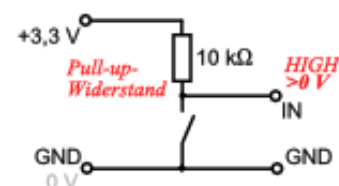
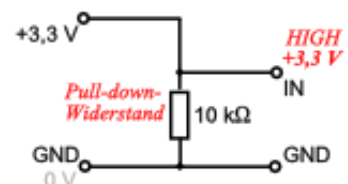
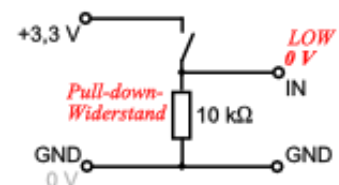
Die digitalen Eingänge an Schaltkreisen sind nicht eindeutig auf Null oder Eins gesetzt. Die interne Schaltung ermöglicht häufig stark schwankende Werte, die nicht eindeutig auswertbar sind. Deshalb sorgt man mit einer einfachen Widerstands-Schaltung dafür, dass immer eindeutige Signale anliegen. Bei Schaltkreisen spricht man beim Null-Signal vom sogenannten LOW-Pegel, der üblicherweise einer Spannung von 0 Volt entspricht. Am Eingang (IN) kommt in der nebenstehenden Schaltung keine Spannung an. Der Widerstand sorgt dafür, dass irgendwelche anderen Spannungen in die Masse (GND, Erdung) abfließen können. Da der Widerstand das Signal auf LOW bzw. DOWN runterzieht, spricht man von einem Pull-down-Widerstand.

Wird nun der Schalter geschlossen, dann liegt die volle Spannung (hier +3,3 V) am Eingang (IN) an. Diese Spannung wird eindeutig als HIGH bzw. Eins interpretiert und die nachfolgenden (internen) Schaltungen reagieren entsprechend.

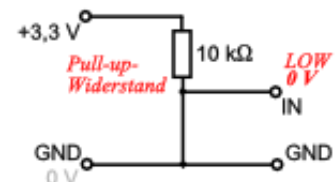
Das Schaltungs-Prinzip lässt sich auch umdrehen.

Hier nutzt man zum Erzeugen eines HIGH-Signals einen sogenannten Pull-up-Widerstand. Dieser reduziert zwar die Betriebsspannung (hier: +3,3 V) auf einen deutlich niedrigeren Wert, aber dieser ist immer größer als Null. Er zieht in quasi über 0 V hoch. Somit interpretiert der Schaltkreis ein HIGH-Signal. Interpretieren heißt hier, dass die internen logischen Schaltkreise umschalten.

Drückt man nun den Taster, dann schließt man den Eingang (IN) mit der Masse (GND) kurz. Es liegt keine Spannung mehr am Eingang an und dieses bedeutet in der Schaltungs-technik eben ein LOW-Signal.

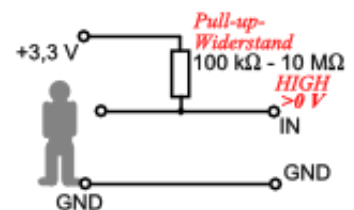


Die Widerstands-Werte können relativ weit gefächert sein. Je nach gewünschter Empfindlichkeit und Schaltsicherheit können Widerstände von mehr als 1 kW bis hoch zu einigen MW verwendet werden. Für praktische Schaltungen kann man die optimalen Werte mit Drehwiderständen ermitteln und dann gegen einen passenden einfachen Widerstand ersetzen.



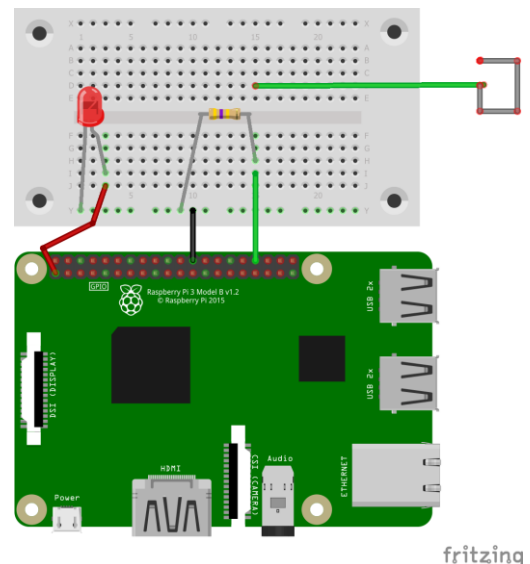
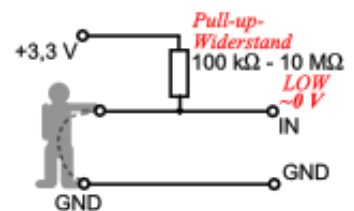
Für eine konkrete "Sensor"-Schaltung nutzen wir die folgende Pull-up-Schaltung. Dabei wird kein Taster oder Schalter genutzt, sondern unser normaler Haut- und Körper-Widerstand. Wir werden somit zum elektrischen Leiter und der minimale Strom, der nun durch uns fließt, kann eine elektronische Schaltung beeinflussen.

Der etwas größer gewählte Widerstand wird zwischen Strom-Versorgung (hier: +3,3 V) und dem Eingang (IN) des Schaltkreises eingebaut. Es fließt ein kleiner Strom, der für einen HIGH-Pegel am Eingang sorgt.



Durch die Berührung des IN-Kontaktes kommt es quasi zu einem "Kurzschluß" über den menschlichen Körper und seiner Haut. Der Pegel wird gegen 0 V gezogen, was der Schaltkreis als LOW-Pegel interpretiert.

Natürlich ist es kein echter Kurzschluß. Unsere Haut und unser Körper haben einen deutlich messbaren elektrischen Widerstand. Feuchte Finger verringern den Widerstand weiter. Weil dementsprechend sehr trockene Haut einen sehr großen Widerstand hat, kann es sein, dass die Schaltung nicht auslöst. Dann hilft z.B. das Anfeuchten der Hand oder der Finger.



eine RGB-LED ansteuern (beliebige Farbe ausgeben)

Bei LED's haben wir schon verschiedene Farben gesehen. Die Farbauswahl ist aber durch die verfügbaren Halbleiter beschränkt.

Aber auch hier können wir wieder unser Auge austricksen. Bringt man jeweils eine LED der drei Grundfarben in einem Gehäuse unter, dann kann man durch die unterschiedliche Ansteuerung der Einzelfarben eine optische Farb-Mischung erreichen. Statt der Farben Rot-Gelb-Blau wird aus historischen Gründen das Farbsystem RGB benutzt. Hier sind die drei Grund-Farben Rot-Grün-Blau. Auch aus diesen drei Farben lassen sich alle Farben – einschließlich weiß – mischen.

Ältere RGB-LED's hatten keine integrierten Vorwiderstände. In den modernen RGB-LED's sind passende Vorwiderstände eingebaut, so dass eine optimale Farbmischung möglich ist. Für "weiß" müssen ja alle drei Grundfarben gleichstark leuchten. Wir gehen hier vereinfacht von drei gleich großen Widerständen aus. Sie betragen 220 Ω . Bei älteren RGB-LED's schalten wir einfach die Vorwiderstände auf dem Steckbrett zwischen Plus-Pol (digitaler Ausgang) und der RGB-LED.

Böse Frage zwischendurch:

von Max Neugierig: "Kann man nicht nur einen einzelnen Vorwiderstand auf die Masse-Seite verwenden? Das würde doch immer zwei Widerstände einsparen, oder?"

Zuerst wollen wir nur die drei Grund-Farben nutzen – also die Einzel-LED's an- und ausschalten. Dadurch sehen wir die Elementar-Farben Rot-Grün-Blau.

```
#!/usr/bin/python

import mcpi.minecraft as minecraft
import RPi.GPIO as GPIO

mc = minecraft.Minecraft.create()
```

notw. Zeile für Kommandozeilenstart von Python-Programmen
Bibliothek für die Verbindung mit Minecraft
Bibliothek für die Ansteuerung der GPIO-Schnittstelle
Erstellen eines Minecraft-Objektes mit dem Namen mc

Im nächsten Schritt sollen entweder zwei oder alle drei Einzel-LED's gemeinsam angesteuert werden. Dadurch erreichen wir erste Farbmischungen. Die RGB-LED leuchtet dann z.B. violett, wenn die rote und die blau Elementar-LED angesteuert wird. Wenn man genau hinsieht, erkennt man noch die Einzel-Komponenten. Aus einiger Entfernung ist nur noch die Mischfarbe zu erkennen.

Richt vielseitig wird unsere RGB-LED, wenn wir die einzelnen Farben "dimmen". Wir können dadurch jede beliebige Farbe mischen.
Nun wollen wir einen schleichenden Farbverlauf programmieren.

eine LED dimmen (frequentes Signal ausgeben)

```
#!/usr/bin/python

import mcpi.minecraft as minecraft
import RPi.GPIO as GPIO

mc = minecraft.Minecraft.create()
```

notw. Zeile für Kommandozeilenstart von Python-Programmen
Bibliothek für die Verbindung mit Minecraft
Bibliothek für die Ansteuerung der GPIO-Schnittstelle
Erstellen eines Minecraft-Objektes mit dem Namen mc

eine LED dimmen (frequentes Signal ausgeben)

```
#!/usr/bin/python

import mcpi.minecraft as minecraft
import RPi.GPIO as GPIO

mc = minecraft.Minecraft.create()
```

notw. Zeile für Kommandozeilenstart von Python-Programmen
Bibliothek für die Verbindung mit Minecraft
Bibliothek für die Ansteuerung der GPIO-Schnittstelle
Erstellen eines Minecraft-Objektes mit dem Namen mc

10.2.2. Aduino und Verwandte

Die Grund-Konzeption der Arduino's ist eine andere, als die bei den Raspberry Pi's. Die Arduino's sind kleine Board's, die sich als Physical-Computing-Plattform verstehen. Dabei sollen sie bestimmte Steuerungs-Aufgaben übernehmen.

Die Programmierung erfolgt ursprünglich über eine C-ähnliche Sprache. Die Programme nennen sich Sketche und können die Arduino's zu beeindruckenden Leistungen bringen.

Seit ein paar Jahren ist auch eine Programmierung mit Python möglich. Dazu müssen aber diverse Vorarbeiten erledigt werden, die wir später vorstellen.

10.2.2.0. Kurzbeschreibung und allgemeine Einführung zu Arduino

sind programmierbare Kleinst-Rechner

Programme werden auf PC erstellt und dann auf den Arduino übertragen
dort laufen sie dann eigenständig (unabhängig vom Programmier-PC)

derzeit sind "Arduino Uno"-Clone für deutlich unter 10 Euro zu haben

ähnlich ist es bei dem kleineren Arduino Micro, diese werden in speziellen Versionen – oft als IoT-Board's – mit WLAN- oder Bluetooth-Schnittstelle angeboten, da liegt der Preis dann aber auch zwischen 20 und 30 Euro

einige – nicht ganz 100%ig kompatible – Uno-Platinen sind sogar unter 5 Euro aus China beziehbar, sie benötigen einen extra Treiber und sind dann kompatibel.

Interessant sind Boxen, die neben der Platine meist Unmengen von elektronischen Bauelementen und Kabeln enthalten. Auch hier sind die Preise sehr interessant und man bekommt ein Bastel-Set zwischen 8 und 50 Euro. Bei den größten Set's erschlagen einen die Möglichkeiten. Da besteht eher die Sorge, dass man da garnicht alles probieren kann. Vielfach fehlen auch Dokumentationen, die den gesamten Bauteile-Bestand erfassen.

10.2.2.1. Einrichtung einer Umgebung für Programmierung eines Arduino mit Python

es gibt verschiedene Herangehensweisen

Bei der Ersten müssen Programmier-Rechner und Arduino immer verbunden sein und die Arduino's können auch nicht selbstständig (weiter-)arbeiten. Das Zauberwort heißt hier pyFirmata (→ [Variante mit pyFirmata](#):).

Nach einem ähnlichen Prinzip funktioniert pySerial (→). Diese Variante wird dann nachfolgend erläutert (→ [Variante mit pySerial](#):). Man kann sich frei zwischen pySerial und pyFirmata entscheiden. Beide Varianten müssen nicht parallel auf dem Rechner installiert werden.

Die zweite Herangehensweise setzt auf eine interne Umsetzung des Python-Programm's in einen C-Code, welcher dann kompiliert auf dem Arduino (auch selbstständig) laufen kann. Das Arbeits-Prinzip nennt man Cross-Compiling. Der Arduino kann dann auch im stand-alone-Betrieb – also ohne den Programmier-Rechner - arbeiten. Dazu weiter hinten Genaueres (→ [Variante mit Shed Skin](#):).

Variante mit pyFirmata:

direkte Interpretation eines Python-Programm's auf dem Arduino ist nicht möglich, da auf dem Arduino solch komplexe Software, wie ein Interpreter, nicht laufen kann. Man kann aber einen Arduino über die USB-Verbindung steuern.

Notwendig ist die sogenannte pyFirmata-Stelle für Python, also eine Erweiterung (ein Modul). Die holen wir uns von github.com (→).

Auf dem Computer, mit dem programmiert werden soll, muss zuerst die Arduino-IDE installiert werden. Die gibt es auf arduino.cc zum Downloaden.

Nun muss auch das Python-System für die Zusammenarbeit mit dem Arduino konfiguriert werden. Das geht z.B. mit:

```
pip install pyfirmata
```

in der Arduino-Programmier-Umgebung den passenden Arduino am richtigen Port einstellen

"File" "Examples" "Firmata" ("Datei" "Beispiele" "Firmata") den Sketch "StandardFirmata" herunterladen und auf den Arduino hochladen

Die Python-Programme müssen dann immer das pyFirmata-Modul importieren:

```
from pyfirmata import Arduino, util
import time
```

Als nächstes setzen wir die Kommunikation auf die zugewiesene USB-Schnittstelle:

```
board = Arduino("COM3")
```

und schon kann das übliche Programmieren beginnen.

Das klassische Start-Programm ist bei den Arduino's ein LED-Blink-Programm. Im einfachsten Fall nutzen wir die Board-interne LED am Pin 13.

Natürlich kann auch eine externe LED mit beliebiger Farbe angesteuert werden. Dazu muss dann eine LED mit einem Vorwiderstand () zwischen Pin X und Gnd (Pin) gesteckt werden. Am Besten sind Aufbauten mit sogenannten Bread-Board's geeignet. Die lassen sich schnell und sicher zusammenstecken.

Den Bauplan sehen wir nebenan.

Im Python-Editor erstellen wir nun den eigentlichen Arbeits-Teil unseres Programm's:

```
while True:
    board.digital[13].write(1)
    time.sleep(0.3)
    board.digital[13].write(0)
    time.sleep(0.3)
```

Aufgaben:

- 1. Ändern Sie die Blink-Frequenz der LED! (Hell- und Dunkel-Phasen sollen aber unbedingt gleichlang sein!)***
- 2. Nun soll die LED am Pin 14 angeschlossen werden! Welche Veränderungen müssen am Board und im Programm vorgenommen werden?***
- 3. Schreiben Sie ein Programm, dass unaufhörlich SOS blinkt!***
- 4. Konzipieren und erstellen Sie nun ein Programm, dass einen beliebigen eingegebenen Text per LED als MORSE-Zeichen sendet!***

Variante mit pySerial:

installieren mit:

```
pip install pyserial
```

Programmieren:

```
import serial # ist pySerial
```

```
board = serial.Serial('/dev/cu.usbmodem143311', 9600, timeout=2) # bzw. Port unter Win
```

```
while True:
    board.write('Testtext')
    antwort = board.readline()
    print("Antwort des Board's: ", antwort)
```

```
print("Verbindung zum Board mit beliebiger Taste unterbrechen!")
```

```
...
try:
    while True:
        ...
        board.digital[13].write(1)
        ...
        board.digital[13].write(0)
except KeyboardInterrupt:
    board.exit()
    print("Verbindung zum Board ist unterbrochen.")
...
```

```
import serial

verbindung = serial.Serial('/dev/tty.usbserial', 9600)

while True:
    print verbindung.readline()
```

Variante mit Shed Skin:

Cross-Compiler
übersetzt Python-Programm in ein C/C++-Programm
derzeit wird nur ein begrenzter Sprach-Umfang von Python unterstützt

- <http://shedskin.github.io/> (Download, ...)
- https://en.wikipedia.org/wiki/Shed_Skin (engl. Wikipedia-Artikel)
- <https://shedskin.readthedocs.io/en/latest/> (Dokumentation)

10.2.2.x. Spezialfall UDOO

Kombination aus Raspberry-Pi-ähnlichem Grundsystem mit Arduino-Hardware

sehr Leistungs-fähig, aber weniger bekannt

Kleinserien-Produktion über Projekt (Finanzierung über ??? Founding (Kickstarter.com))

10.2.3. FRANZIS – Experimentierplatine mit FT232R

USB/Seriell-Wandler
praktisch Schnittstelle zwischen Betriebssystem / Anwender-Programmen und Experimentier-Hardware

leider nur unter Linux verwendbar und somit auch noch ein Rechner notwendig
(Nutzung über virtuelle Maschine noch nicht geprüft)

Kombination mit graphischer Oberfläche Gtk
Fenster-Manager Gnome
Gtk selbst ist auch für Windows verfügbar

erhältlich bei FRANZIS
Preis 99 Euro (etwas teuer)
im Schnäppchen-Angebot für 49 Euro auch immer noch recht teuer
dafür bekommt man schon einen vollwertigen Raspberry Pi mit Zubehör (mSD-Karte, Bauelemente, ...)

10.2.4. TI-Innovator

ab Oktober 2020
Programmierung über TI-Nspire

Befehle sind über die den Werkzeug-Button eingefügt. Dort sind Menü's mit den verfügbaren Python-Befehlen zusammengestellt.
Die Befehle können dann nach Art der Block-Programmierung zusammengestellt werden.

gute Hilfe sind die notwendigen Bibliotheks-Aufrufe (Importe), die als Befehl gleich immer oben in den betreffenden Menü's aufgezählt sind

education.ti.com/de

auf der /fr gibt es die TI-83
ev. auch amerikanische TI-Seite nutzen

<https://education.ti.com/de/activities/ti-codes>

ti-unterrichtsmaterialien.net

10.2.4.y. externe Hardware

RGB-Array

anzuschließen über 4 Drähte

```
from ti_hub import *  
rgb=rgb_array()  
rgb.set(position,rot,gruen,blau)  
sleep(zeit)  
rgb.all_off()
```

```
rgb.set_all(rot,gruen,blau)
```

```
from ti_hub import *  
from random import *
```

```
while get_key()!="esc":  
    position=randint  
    rot=randint(0,255)  
    gruen=randint(0,255)  
    blau=randint(0,255)  
    rgb.set(position,rot,gruen,blau)  
    sleep(1)  
rgb.all_off()
```

mit Liste:

```
liste=[]  
liste=[0 for i in range(16)] #Null-befüllte Liste  
  
for pos in range(0,16):  
    print(liste[i])  
    rgb.set(pos,
```

gibt wert als Binär-Zahl auf dem RGB-Array aus
rgb.pattern(wert)

misst den (Gesamt-)Strom:
wert=rgb.measurement()

Aufgaben:

- 1.
2. *Lassen Sie die einzelnen LED's nach und nach in der gleichen Farbe leuchten! Dabei soll sich die Intensität immer leicht erhöhen.*
3. *Jede LED soll zufällig ausgewählt mit einer zufälligen Farbe belegt werden. Das Programm soll solange laufen, bis die ESC-Taste gedrückt wird.*
4. *Erstellen Sie ein Programm, dass immer eine einzelne LED für 1 Sekunde in einer frei gewählten Farbe leuchten lässt und dann durch die nächste abgelöst wird!*
5. *Lassen Sie einen wieder einen "Leuchtpunkt" wandern! Dieses Mal soll die zuletzt benutzte LED mit halber Stärke nachleuchten!*
6. *Realisieren Sie ein Programm, dass immer eine zufällig ausgewählte LED das gesamte Farbspektrum durchläuft!*
7. *Realisieren Sie ein kleines Spiel bei dem der menschliche Spieler und Ihr Programm jeweils eine Zahl zwischen 1 und 8 setzen! Die gewählten Zahlen*

werden als gegenläufige Leuchtpunkte in den beiden Reihen des RGB-Array dargestellt. Der menschliche Spieler hat eine blaue Reihe, Ihr Programm eine gelbe. Wenn sich die Punktreihen überschneiden erhält der Spieler mit der größten Zahl einen Minus-Punkt, ansonsten bekommt derjenige einen Punkt, der die längste Reihe hatte. Bei Gleichstand erhält jeder einen Punkt / Minus-Punkt! Die Plus-Punkte werden als grüne Punkte und die Minus-Punkte als rote Leucht-Punkte angezeigt. Gewonnen hat derjenige, der zuerst 8 Gewinn-Punkte hat. Ihr Programm darf die aktuelle Eingabe des menschlichen Spielers nicht für die aktuelle Entscheidung benutzen, darf sich aber alle alten Eingaben merken und für eine Strategie auswerten.

(Realisieren Sie das Programm ev. in zwei groben Schritten: Zuerst nur die Anzeige der Spiel-Züge. Die Ergebnisse können dann auf der Shell ausgegeben werden. Im zweiten Schritt kann dann die Spielstand-Anzeige erfolgen.)

Zusatz: Lassen Sie sich eine passende Gewinn-Anzeige einfallen (Leucht-Show)!

10.2.5. Steuerung des Calliope mini

MakeCode lässt neben der typischen Block-basierten Programmierung auch eine Übersetzung in Python zu. Änderungen im Python-Quell-Text werden dann wieder in die Block-Code zurückgespiegelt.

MakeCode-Editor → makecode.calliope.cc

Multi-Editor

makecode.calliope.cc/--multi

lässt z.B. die Programmierung und Simulation von einem Sender und einem Empfänger in einem Browser zu

z.B. zum Testen

MakeCode-Programmierung ist Event-orientiert

OpenRoberta ist dagegen prozedural angelegt
man muss die Endlos-Schleife selbst realisieren

10.3. Datenbank-Zugriff mit Python

→ <https://www.hdm-stuttgart.de/~maucher/Python/html/SQLite.html#connection-und-cursorobjekt-erzeugen>

10.3.1. SQLite 3

```
from sqlite import dbapi2 as sqlite
```

10.3.1.0. Verbindung herstellen

außer der Verbindung brauchen wir noch ein Cursor-Objekt
es stellt quasi die imaginäre Shell (Konsole, Benutzeroberfläche) dar, es ist im Programm so,
als würden wir auf die Shell Anweisungen etc. in SQL schreiben

```
verb = sqlite.connect(Datenbankname)    # Datenbankname ist Unterverzeichnis + Daten-  
bank
```

eine temporäre, lokale (nicht-persistierende) Datenbank lässt sich mit:

```
verb = sqlite.connect(:memory:)  
curs = verb.cursor()
```

erzeugen. Die zweite Anweisung erzeugt einen SQL-Cursor.

10.3.1.1. Erstellen einer Tabelle

```
curs.execute("create table if not exists schueler (name text, vorname text, gebdatum date,  
masse real, groesse integer)")
```

Datentyp in Python	Datentyp in SQLite	
None	NULL	
int	INTEGER	
float	REAL	
str (UTF8)	TEXT	
unicode	TEXT	
buffer	BLOB	

10.3.1.2. Hinzufügen von Datensätzen zu einer Tabelle

```
curs.execute("""insert into schueler values ('Mustermann','Klaus','01.01.2000',63.7,177)""")
...
verb.commit() # eigentliche Speicherung der vorher angegebenen Datensätze
```

10.3.1.3. Aktualisieren eines Datensatzes in einer Tabelle

```
geschlechterListe = ["m","w", ...] # soviele Listen-Einträge, wie Datensätze in Tabelle
curs.execute("""alter table schueler add column geschlecht text""")
for geschl in enumerate(geschlechterListe):
    curs.execute("update schueler set geschlecht=(?) where owid=(?)",
                  [geschlechterListe[geschl],geschl+1])
verb.commit() # eigentliche Aktualisierung der vorher spezifizierten Werte in der
               erweiterten Tabelle
```

10.3.1.4. Löschen eines Datensatzes aus einer Tabelle

```
curs.execute("""delete from schueler where vorname="Klaus" """)
...
verb.commit() # eigentliche Löschung der vorher spezifizierten Datensätze
```

10.3.1.5. Löschen einer Tabelle

```
curs.execute("""drop table if exists schueler""")
```

10.3.1.z. Beenden der Verbindung

weitere Beispiele:

```
import sqlite3

conn = sqlite3.connect('daten/Kontakte.dat')
curs = conn.cursor()
curs.execute("CREATE TABLE personen(PID, Vorname, Name, eMail)")

DatenListe=[(1,"Monika","Musterfrau","musterfrau@webb.de"),
             (2,"Klaus","Mustermann","muma@tee-online.de"),
             (3,"Prof. Lisa","Klug","Prof.L.Klug@uni-mustern.de)]

for Elem in DatenListe:
    curs.execute("Insert INTO personen VALUES (?, ?, ?, ?)", Eleme)

...

curs.close()
conn.close()
```

`curs.fetchone()` liefert eine Ergebnis-Zeile zur Anfrage als Tupel

`curs.fetchmany(n)` liefert n Ergebniszeilen zur Anfrage als n-Tupel von Tupeln

`curs.fetchall()` liefert alle Ergebniszeilen zur Anfrage als Tupel von Tupeln

10.4. Web-Server-Anwendungen mit dem (Micro-)Framework Flask

unterstützt Generierung von Seiten auf einem Web-Server

allgemeines Handling:

auf einem Intranet- oder Internet-Server läuft ein Webserver (Dienst der die Bereitstellung von http-Seiten realisiert)

auf einem Client läuft ein Browser (Betrachter-Programm für http-Seiten)

nach Aufruf der Server-Adresse z.B.: `www.lern-soft-projekt.de` oder `127.0.0.1` (localhost; Webserver läuft auf dem eigenen Rechner) wird eine Seite (normal `index.htm` od.ä.) angefordert (get-Methode) (ev. wird mit Fehler-Meldung geantwortet)

Webserver schickt HTML-Code der angeforderten http-Seite an den Browser

Browser setzt den HTML-Code in eine anzeigbare Seite um oder gibt Fehler-Meldungen aus
Client / Browser können nach Interaktionen neue Inhalte (nach-)laden / liefern / anzeigen

10.4.0. Erzeugung einer Web-Seite mit Python (Wiederholung)

10.4.1. das Framework Flask

dient hauptsächlich der Trennung von Daten und Darstellung

der Programmierer soll sich nicht mehr vorrangig um die Darstellung seiner Daten kümmern, das übernimmt im Wesentlichen das Framework

der Programmierer stellt seine Daten bereit und nutzt vorgefertigte Methoden / Funktionen usw. um die Darstellung fast von alleine dem Framework zu überlassen

bringt u.a. das Kachel-Design mit

sehr gut für Anzeigen etc. von IoT-Daten oder Info-Daten geeignet

ev. muss vorher ein Web-Server installiert und / oder eingerichtet werden

minimales Hello-Welt-Programm mit Flask

```
from flask import Flask

app = Flask(__name__)

@app.route('/')

def index():
    return "Hallo Welt!"
```

der große Vorteil von Flask (wie auch von anderen Frameworks), dass diese mit Templates (Schablonen) arbeiten können. Die Templates enthalten bestimmte Stellen, an denen dann durch einfache Funktionen / Methoden die Inhalte eingefügt werden.

So ist z.B. in einem Template festgelegt, dass eine Überschrift mit einem Rahmen versehen sein soll, größer, unterstrichen und fett dargestellt werden soll. Auch Farben usw. lassen sich definieren.

Der Programmierer ruft jetzt nur eine Funktion auf, wie z.B. `generiereÜberschrift()` und übergibt dieser nur den Text. Die ganze Einstell-Arbeit übernimmt das Framework auf der Basis der bereitgestellten Schablone. Ist für eine – quasi parallel laufende – "andere" Webseite ein anderes Template festgelegt, dann sieht die gleiche Überschrift dort eben anders aus. Darum braucht sich der Programmierer nicht zu kümmern. Das Layout gestaltet ein Designer.

Dieses Prinzip kennt man von vielen Webseiten / Social-Media-Seiten, wo man seine individuellen Farb-Einstellungen etc. vornehmen kann.

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
@app.route('/reserve')
```

Import des Flask-Frameworks
Flask-Applikation def.
Funktionsdekorator
alternativer Dekorator

```
def index()
    return render_template("template.html,
                           var1="Hallo", var2="Welt")
```

beliebiger Funktionsname
Darstellungs-Aufruf unter Nutzung eines Templates und zusätzlichen Daten

Dieser Python-Code und Flask benötigen aber dazu das passende Template:

```
<!doctype html>
<html>
  <head>
    <title>
      Seite: Hallo-Welt
    </title>
  </head>
  <body>
    <h1>Überschrift der Hello-Welt-Seite</h1>
    übergebene Daten sind: {{var1}} und {{var2}}
    zusammen: {{var1}} {{var2}}!
  </body>
</html>
```

Achtung! Im Gegensatz zu Python sind bei HTML die Einrückungen nur Mittel zur übersichtlicheren Darstellung. Sie können vollständig weggelassen werden und sogar alles fortlaufend in eine Zeile geschrieben werden.

Mit weiteren Web-Techniken können weitere Verbesserungen / Funktions-Erweiterungen etc. erreicht werden. So kann man mit CSS (Cascade Style Sheets) Format-Vorlagen und andere Layout-Parameter (z.B. für ein Corporate Design) definieren. Diese werden dann im HTML-Code zugewiesen.

```
...
  <head>
    <link rel="stylesheet" href=stylesheet.css">
  </head>
...
```

JavaScript eignet sich z.B. um Anpassungen der Seite vornehmen zu lassen oder Berechnungen durchzuführen.

Die Java-Scripte werden üblicherweise am Ende des Body-Bereiches angegeben.

```
...
<body>
    ...
    <script scr="funktion.js">
</body>
...
```

Das folgende Python-Programm ist ein schönes – aber auch schon recht komplexes - Beispiel für eine Flask-Anwendung. Für die Nutzung der oben erwähnten Adafruit-Experimentier-Platine oder eines ähnlichen IoT-Systems soll eine Kachel-Oberfläche dienen. Die einzelnen Kacheln dienen entweder der Anzeige von Meßwerten, dem Ein- bzw. Ausschalten oder dem Wechsel zu Unterseiten usw.

Die Datei ist auf github.com gehostet und kann über:

`git clone https://github.com/openHPI/Embedded-Smart-Home-2017.git`

in das aktuelle Verzeichnis kopiert werden. Dort ist die nachfolgend angezeigte `smarthome.py` enthalten. Sie dient als Steuerzentrale des gesamten Projektes.

```
from flask import Flask, render_template, request
from flask_bootstrap import Bootstrap

from tiles import SimpleTile, TileManager
from helper import PageContext

app = Flask(__name__)
Bootstrap(app)

@app.route('/')
def main():
    tiles = [
        SimpleTile("Licht", "#EEEE00", "light/"),
        SimpleTile("Heizung", "#FF0000", "heaters/"),
        SimpleTile("Sicherheit", "#30FF00", "security/"),
        SimpleTile("Wasser", "#0000FF", "water/"),
        SimpleTile("Extrapunkt 1", "#00FFFF", "/"),
        SimpleTile("Extrapunkt 2", "#FF00FF", "/"),
        SimpleTile("Extrapunkt 3", "#A0FFA0", "/"),
        SimpleTile("Extrapunkt 4", "#00A0FF", "/"),
    ]

    manager = TileManager(tiles)
    context = PageContext("Smarthome Projekt", "Home")
    return render_template("main.html", tilerows=manager,
context=context)

@app.route('/light/')
def light():
    living_room = True
    sleeping_room = False

    if("living_room" in request.args):
        living_room = True if request.args["living_room"] == "on" else False

    if("sleeping_room" in request.args):
```

```

        sleeping_room = True if request.args["sleeping_room"]↵
        == "on" else False

    tiles=[]

    tile = SimpleTile("Wohnzimmer: ", "", "?living_room=")
    tile.items[0].text += "an" if living_room else "aus"
    tile.link += "off" if living_room else "on"
    tile.bg = "#AAFF00" if living_room else "#338800"
    tiles.append(tile)

    tile = SimpleTile("Schlafzimmer: ", "", "?sleeping_room=")
    tile.items[0].text += "an" if sleeping_room else "aus"
    tile.link += "off" if sleeping_room else "on"
    tile.bg = "#6666BB" if sleeping_room else "#333388"
    tiles.append(tile)

    manager = TileManager(tiles)
    context = PageContext("Smarthome Projekt", "Licht",↵
        ["/", "Home"])
    return render_template("main.html", tilerows=manager,↵
        context=context)

if __name__ == "__main__":
    app.run(debug=True)

```

10.4.2. die Flask-Erweiterung bootstrap

10.4.3. Programmierung der Web-Oberfläche und Darstellung von Meß- werten

```

from flask import Flask, render_template, request
from flask_bootstrap import Bootstrap

from tiles import SimpleTile, TileManager
from helper import PageContext

import sqlite3
from flask import g

import requests, json

app = Flask(__name__)
Bootstrap(app)
app.config['BOOTSTRAP_SERVE_LOCAL'] = True

```

```

DATABASE='database.sqlite'

def get_db():
    db = getattr(g, '_database', None)
    if db is None:
        db=g._database=sqlite3.connect(DATABASE)
    return db

def query_db(query, args=(), one=False):
    cur = get_db().execute(query, args)
    rv = cur.fetchall()
    cur.close()
    return (tv[0] if rv else None) if one else rv

@app.teardown_appcontext
def close_connecting(exception):
    db = getattr(g, '_database', None)
    if db is not None:
        db.close()

@app.route('/')
def main():
    temp = query_db("SELECT wer, einheit FROM sensoren
                    ORDER BY zeit DESC", one=True)
    print(temp)      #Kontrollanzeige auf Konsole
    tiles = [
        SimpleTile("Licht", "#EEEE00", "light/"),
        SimpleTile("Heizung", "#FF0000", "heaters/"),
        SimpleTile("Sicherheit", "#30FF00", "security/"),
        SimpleTile("Wasser", "#0000FF", "water/"),
        SimpleTile("Innentemperatur: " + temp[0] + " " +
                    temp[1], "#FF0000", "/"),
        SimpleTile("Außentemperatur", "#00FF00", "/"),
        SimpleTile("Luftfeuchtigkeit", "#0000FF", "/"),
        SimpleTile("Helligkeit", "#FFFF00", "/"),
    ]

    manager = TileManager(tiles)
    context = PageContext("Smarthome Projekt", "Home")
    return render_template("main.html", tilerows=manager,
                           context=context)

@app.route('/light/')
def light():
    living_room = True
    sleeping_room = False

    if("living_room" in request.args):
        living_room = True if request.args["living_room"]
        == "on" else False

    if("sleeping_room" in request.args):
        sleeping_room = True if request.args["sleeping_room"]
        == "on" else False

    tiles=[]

    tile = SimpleTile("Wohnzimmer: ", "", "?living_room=")
    tile.items[0].text += "an" if living_room else "aus"
    tile.link += "off" if living_room else "on"

```

gibt Verbindung zur Datenbank zurück

Datenbank-Abfrage
one bestimmt, ob nur 1 Wert zurückgeliefert werden soll

```

tile.bg = "#AAFF00" if living_room else "#338800"
tiles.append(tile)

tile = SimpleTile("Schlafzimmer: ", "", "?sleeping_room=")
tile.items[0].text += "an" if sleeping_room else "aus"
tile.link += "off" if sleeping_room else "on"
tile.bg = "#6666BB" if sleeping_room else "#333388"
tiles.append(tile)

manager = TileManager(tiles)
context = PageContext("Smarthome Projekt", "Licht", ↵
                      ["/", "Home"])
return render_template("main.html", tilerows=manager, ↵
                      context=context)

if __name__ == "__main__":
    app.run(debug=True)

```

fehlt requests bzw. gibt es dahingehend Fehler-Meldungen, dann muss die Bibliothek nachinstalliert werden:

pip3 install requests

Um z.B. externe Wetter-Daten mit anzuzeigen braucht man eine Daten-Quelle für solche Informationen. Dazu ist es bei openweathermap.org sich einen Account-Schlüssel zu besorgen und damit dann rund 60 Wetter-Info-Pakete pro Stunde runterzuladen. Die Daten kommen als JSON-Datei und müssen mittels json-Bibliothek in ein JSON-Objekt umgewandelt werden. Dazu brauchen wir die importierte json-Bibliothek. Natürlich könnte man den Text auch per Hand selbst zerlegen (parsen). Das ist aber recht aufwendig in der Programmierung. Da nutzen wir lieber die vorgefertigten und geprüften Methoden / Funktionen von json.

```

def main():
    temp = query_db(...)
    r = requests.get('http://api.openweathermap.org/data/↵
                    2.5/weather?g=Rostock,de&appid= eigene ID')
    weatherdata = json.loads(r.text)
    temp_out = round(weatherdata['main']['temp'] - 273.15,1)
    weathersymbol = ''

    ...
    SimpleTile("Außentemperatur: " + str(tempout) + " C <br>"↵
              + weathersymbol, "#FF00FF", "/"),
    SimpleTiel(...)

```

Den API-Key muss man sich bei openWeatherMap.org besorgen. Es gibt auf openWeatherMap.org auch API's, die eine Abfrage von Wetter-Vorhersagen erlauben.



10.5. Web-Applikationen mit Django

recht freies, kompakter und beherrschbares Framework
legt Wert auf effektives Programmieren (alles möglichst nur einmal programmieren (Don't Repeat Yourself → DRY))
mit eigenem Web-Server zum schnellen Testen / Ausprobieren

Nachteile / Probleme:
Struktur nicht selbsterklärend

Links:

<http://www.django-workshop.de/> (gutes aufgebautes Tutorial (dt.)) → Beispiel: Rezept-Sammlung
[scheinbar ist die Version des Tutorials veraltet (0.4), einschließlich der verwendeten Programm-Versionen (Python 2?!, 3.3, Django 1.4); ab und zu ist nicht klar, was auf welcher Ebene gemacht werden muss; keine Fehler-Hinweise; es fehlt das Hintergrundwissen, um die Zusammenhänge zu verstehen; einige Texte wirken sehr abstrakt]

10.6. MicroPython für Microcontroller

u.a. Q:

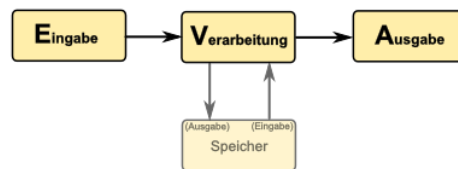
/µP_Q1/ ... Thomas WALDMANN (Vortrag: "Einführung in ESP32 Microcontroller + MicroPython"; EASTERHEGG 2018;
<https://media.ccc.de/v/V8W9DL>)

Es muss nicht immer ein großer Rechner sein, um mit Python zu arbeiten. Wir haben ja schon gesehen, dass die kleinen Raspberry Pi's ebenfalls mit Python daherkommen und nicht wirklich Leistungs-schwächer sind. Natürlich muss man hier die allgemeine Leistungsfähigkeit des Grundsystem's beachten.

Es geht aber noch kleiner. Microcontroller sind minimalste Datenverarbeitungs-Systeme und zielen stark auf den IoT-Bereich ab.

Microcontroller – oft auch als Experimentier-Board's oder IoT-Bausteine bezeichnet - verfügen auf kleinsten Raum über alle EVAS-Teile eines Informatiksystems.

Besonders effektive Microprozessoren steuern unzählige Ein- und Ausgabe-Möglichkeiten.



Klassiker sind sicher die Arduino's. Sie waren und sind noch zu langsam und zu Speicherarm für Python.

Neuere Hardware ist da um Längen besser. Zu den neuen Sternen am Himmel zählen z.B. die ESP-Bausteine.

Fast allen Microcontrollern ist eine sehr offene Hard- und Software gemeinsam. Zwar ist nicht alles OpenSource oder völlig frei zugänglich, aber die offene Arbeitskultur von Hard- und Software-Herstellern erzeugt eine schnelle und breite Nutzung in allen Bereichen.

Vorteile:

- relativ einfache Programmierung (im Vergleich zum sonst üblichen C/C++)
- unendliches Laufen eines Programms auf minimalster Technik
- praktisch fast unbegrenztes direktes und indirektes Ansprechen von Sensoren und Aktoren
- langfristiger und relativ unabhängiger Betrieb über PowerBank-Stromversorgung möglich
- Unterstützung vieler Protokolle und der üblichen Hardware / Peripherie

Nachteile:

- etwas eingeschränkter Befehls-Umfang
- neue Bibliotheken / Module notwendig
- etwas umständliche Handhabung zwischen Entwicklung und Programm-Lauf
- Größe der Programme meist durch relativ kleine Speicher begrenzt
- auf MicroPython umgestellte Systeme müssen wieder speziell auf den üblichen Microcontroller-Betrieb (übliche C/C++-Programmierung od.ä.) umgestellt werden
-

fehlende Funktionen / Features (im Vergleich zum Standard-Python):

- keine Unterstützung von Unicode
- Leerzeichen zwischen Literalen (Zahlen) und Schlüsselwörtern notwendig
- geänderte Methoden-Auflösung bei geschachtelten Klassen
- nur eine Oberklasse festlegbar
- unterschiedliche Ausgabe-Formate für **float**-Zahlen
- für von **int** abgeleitete Typen ist kein Typ-Umwandlung möglich

- Slicing in Listen eingeschränkt
- **eval()** hat keinen Zugriff auf lokale Variablen
- in Generator-Funktionen wird **__exit__()** nicht aufgerufen
- Byte-Array's werden nicht unterstützt
- String-Methode **.endwith()**, **.ljust()** und **.rjust()** nicht implementiert
- **__del__** als spezielle Methode nicht implementiert
- **self** wird als ein Argument gezählt
- begrenzte Unterstützung von Namespace's
- Zeichenketten-Verarbeitung mit Schlüsselwörtern (z.B. **encoding**) nicht möglich
- lokale Variablen werden bei **locals()** nicht einbezogen
- Nutzer-definierte Attribute in Funktionen werden nicht unterstützt
- spezieller Umgang mit property-Getter
- die **__path__**-Eigenschaft von Modulen wird als relativer Pfad ausgegeben
- Verkettung von Exception's nicht implementiert
- die Methode **Exception.__init__** gibt es nicht
- keine Nutzer-definierten Attribute in Builtin-Exception's
- bei Fehler-Anzeigen in while-Schleifen werden Zeilennummern anders gezählt
- für Bytes-Objekte ist eine **.format()**-Methode verfügbar
- die Nutzung von **step != 1** in Byte-Objekten sowie in Tuple und Listen nicht möglich
- Instanzen von **str**-Unterklassen können nicht mit **str**-Instanzen verglichen werden
-

fehlende / geänderte Funktionen / Features in Modulen:

- im **array**-Module
 - keine Suche nach Integer möglich
 - Löschen (**del()**) von Elementen nicht möglich
 - Nutzung von **step != 1** nicht möglich
- builtin's
- kein zweites Argument bei **next()** möglich
- im **collections**-Modul
 - **deque** nicht implementiert
- im **json**-Modul
 - nicht-seralisierte Einträge erzeugen keine Exception's
- im **struct**-Modul
 - zu viele Argumente in der **.pack()**-Methode werden nicht beachtet
- im **sys**-Modul
 - die Attribute **.stdin**, **.stdout** und **.stderr** lassen sich nicht überschreiben
-

Bedeutung des MicroPython in der Microcontroller-Welt:

- leichter zu programmieren als C/C++
- weite und immer weiter steigende Verbreitung von Microcontrollern
-

bekannte Forks zum MicroPython

- CircuitPython
- PyCom

Warum funktioniert das Arbeiten mit Python auf einem Microcontroller, wenn sonst immer auf einem extra (Host-)Rechner editiert und kompiliert werden muss?

Normalerweise sind die Interpreter und Compiler moderner Programmiersprachen sehr große Programme. Der Speicher und meist auch die Leistung der CPU der Microcontroller reicht nicht für sie aus. Auch bei Python ist das so.

Bei MicroPython wird ein extra kleiner Interpreter mit eingeschränkter Leistung verwendet. Wir haben das oben schon thematisiert.

Zum anderen bedient man sich eines Trick's. Normalerweise werden ja immer die kompilierten Programme – also Binär-Dateien – auf den Microcontroller übertragen. Dies nennen wir flashen. Das (fest integrierte und unveränderliche) Boot-System und die variable Firmware des Microcontroiller's isind so ausgelegt, dass sie die gefundene Binär-Datei vom letzten Flashen startet und unermüdlich abarbeitet.

Beim Python wird der Mini-Interpreter geflasht und dazu ein klassisches Datei-System erzeugt. Das Boot-System des Microcontroller's startet also den aufgeflashten MicroPython-Interpreter und dieser kommuniziert zum Einen über die serielle Schnittstelle oder arbeitet ein gefundenes py-Programm ab.

10.6.x. MicroPython für micro::bit

lässt sich auch mit dem pyCraft-Tool (→) bedienen

auf dem micro::bit muss die micropython-Firmware geladen werden
Download unter
dannach einfach auf das micro::bit-Laufwerk kopieren

ab jetzt versteht der micro::bit die Programmiersprache Python
zumindestens eine abgespeckte Version (enthält aber alle elementaren Befehle!)
die Programmierung und Kommunikation kann mit mehreren Editoren usw. erfolgen

soll wieder mit anderen Sprachen / Systemen programmiert werden, dann muss wieder die ursprüngliche Firmware aufgespielt werden (→ <https://microbit.org/get-started/user-guide/firmware/>)

praktischer ist die Verwendung von 2 Geräten mit jeweils anderer Firmware, dann spart man sich Probleme, wenn man nicht mehr weiss, welche Firmware gerade läuft

weitere Editoren für microPython:

Text-basierte Systeme

Block-basierte Systeme

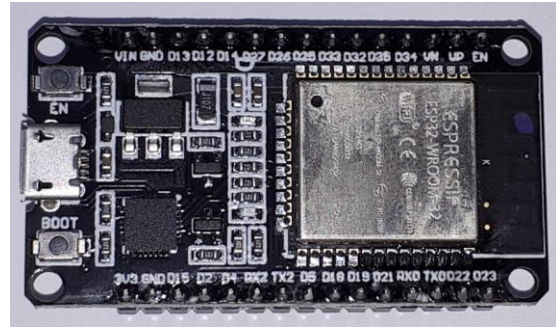
EduBlocks

<https://app.edublocks.org/#MicroBit>

10.6.x. MicroPython für ESP-32-Microcontroller

ESP-8266 und das Nachfolge-Modell ESP-32 sind deutlich leistungsfähiger als die sonst üblichen Arduino's
verschiedene Ausführungen und Hersteller

sie unterscheiden sich vorrangig in der Anzahl der herausgeführten – und damit nutzbaren – Pin's



auffallend ist die extrem funktionell breite Auslegung des Microcontroller
man kann fast schon sagen, alles was das IoT- und Bastler-Herz liebt und braucht ist sehr effektiv im ESP umgesetzt
praktisch billige Massen-Ware, je nach Ausstattung zwischen 6 und 30 Euro
bei den teureren Varianten sind dann oft schon Display's mit dabei
(i.A. insgesamt günstiger als Aduino-System (einschließlich der verschiedenen Billig-Clone)
besonders herausregend in dieser Preisklasse die breite Unterstützung von WLAN und Bluetooth

10.6.x.0. Vorbereiten des ESP für MicroPython

Download des Image von der MicroPython-Webseite (→ <http://micropython.org/download>)
weiterhin ein Fork unter (→) verfügbar (teilweise Leistungs-fähiger, aber eben Spezial-Lösung!)
nicht immer unbedingt das neueste Daily-Build verwenden, da hier schnell Bug's drin sein können, dafür sind aber alte Bug's im Allgemeinen bereinigt

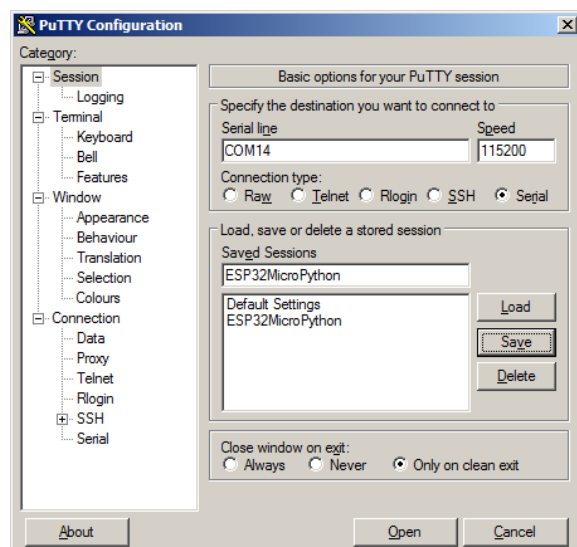
zuerst die alte Firmware auf dem ESP löschen
`esptool.py -chip esp32 -port /dev/ttyUSB0 erase_flash`

neue Firmware (Minimal-OS + MicroPython) hochladen
`esptool.py -chip esp32 -port /dev/ttyUSB0 write_flash -z 0x1000 esp32_firmware.bin`

Verbindung zum ESP-32-MicroPython
über ein Konsolen-Programm
hier PuTTY

wir brauchen eine serielle Kommunikation (also: Serial) mit den Parametern:
Serial line: COM14 (USB-Port)
Speed: 115200 (übliche Baud-Rate
für die Kommunikation mit dem ESP-32

ich habe mir die Parameter gleich unter einem passenden Namen abgespeichert

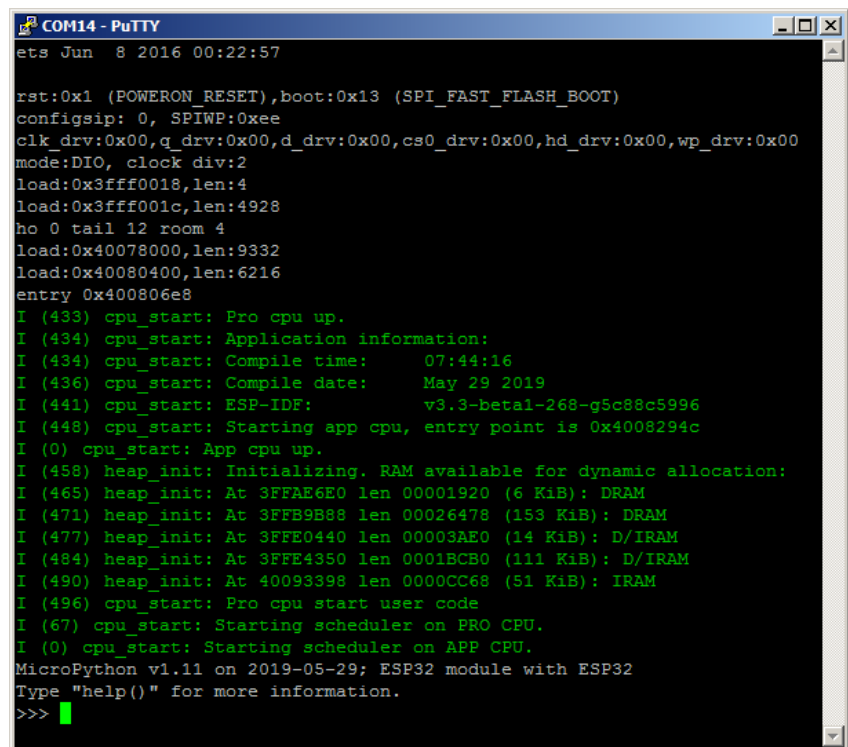


kommuniziert über die USB-serielle Schnittstelle mit dem MicroPython

zuerst bekommt man eine Status-Information zum MicroPython und einigen Ressourcen angezeigt

den seriellen Monitor kann man jetzt auch weiter auflassen und mit dem MicroPython kommunizieren

im Prinzip sind wir jetzt im interaktiven Modus, so wie wir ihn ja schon vom großen Python mit IDLE kennen

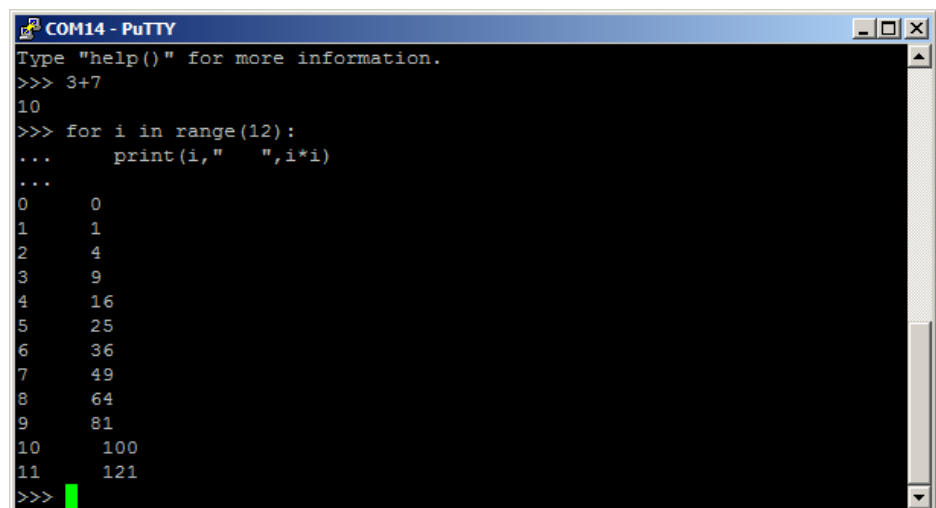


```
COM14 - PuTTY
ets Jun  8 2016 00:22:57

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0018,len:4
load:0x3fff001c,len:4928
ho 0 tail 12 room 4
load:0x40078000,len:9332
load:0x40080400,len:6216
entry 0x400806e8
I (433) cpu_start: Pro cpu up.
I (434) cpu_start: Application information:
I (434) cpu_start: Compile time:      07:44:16
I (436) cpu_start: Compile date:     May 29 2019
I (441) cpu_start: ESP-IDF:          v3.3-beta1-268-g5c88c5996
I (448) cpu_start: Starting app cpu, entry point is 0x4008294c
I (0) cpu_start: App cpu up.
I (458) heap_init: Initializing. RAM available for dynamic allocation:
I (465) heap_init: At 3FFAE6E0 len 00001920 (6 KiB): DRAM
I (471) heap_init: At 3FFB9B88 len 00026478 (153 KiB): DRAM
I (477) heap_init: At 3FFE0440 len 00003AE0 (14 KiB): D/IRAM
I (484) heap_init: At 3FFE4350 len 0001BCB0 (111 KiB): D/IRAM
I (490) heap_init: At 40093398 len 0000CC68 (51 KiB): IRAM
I (496) cpu_start: Pro cpu start user code
I (67) cpu_start: Starting scheduler on PRO CPU.
I (0) cpu_start: Starting scheduler on APP CPU.
MicroPython v1.11 on 2019-05-29; ESP32 module with ESP32
Type "help()" for more information.
>>>
```

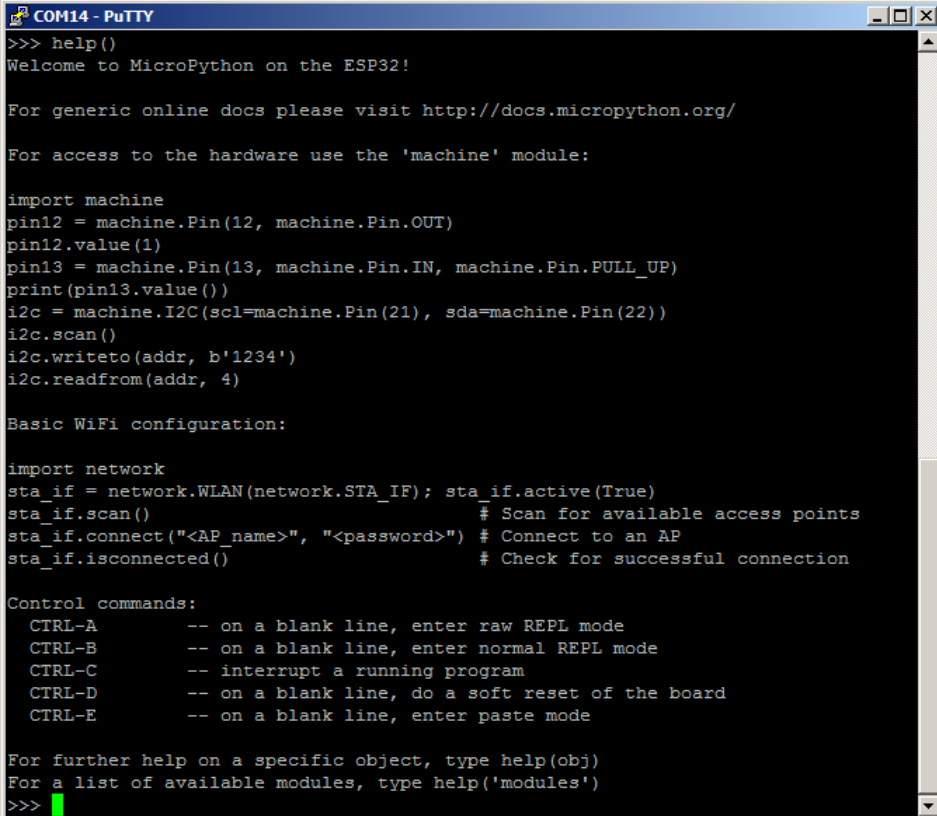
hier zwei kurze und einfache Beispiele

dieser Modus nennt sich REPL (Read-Evalute-Print-Loop)



```
COM14 - PuTTY
Type "help()" for more information.
>>> 3+7
10
>>> for i in range(12):
...     print(i," ",i*i)
...
0      0
1      1
2      4
3      9
4     16
5     25
6     36
7     49
8     64
9     81
10    100
11    121
>>>
```

mit `help()` kann man sich die elementaren Hilfetexte für das MicroPython ansehen



```
>>> help()
Welcome to MicroPython on the ESP32!

For generic online docs please visit http://docs.micropython.org/

For access to the hardware use the 'machine' module:

import machine
pin12 = machine.Pin(12, machine.Pin.OUT)
pin12.value(1)
pin13 = machine.Pin(13, machine.Pin.IN, machine.Pin.PULL_UP)
print(pin13.value())
i2c = machine.I2C(scl=machine.Pin(21), sda=machine.Pin(22))
i2c.scan()
i2c.writeto(addr, b'1234')
i2c.readfrom(addr, 4)

Basic WiFi configuration:

import network
sta_if = network.WLAN(network.STA_IF); sta_if.active(True)
sta_if.scan() # Scan for available access points
sta_if.connect("<AP_name>", "<password>") # Connect to an AP
sta_if.isconnected() # Check for successful connection

Control commands:
CTRL-A -- on a blank line, enter raw REPL mode
CTRL-B -- on a blank line, enter normal REPL mode
CTRL-C -- interrupt a running program
CTRL-D -- on a blank line, do a soft reset of the board
CTRL-E -- on a blank line, enter paste mode

For further help on a specific object, type help(obj)
For a list of available modules, type help('modules')
>>>
```


10.6.x.0.1. das Tool uPyCraft

Alternativ – und insgesamt deutlich komfortabler – lässt sich die MicroPython-Firmware auch mit dem nachfolgend besprochenem Programm (uPyCraft) erledigen.

Falls das Programm noch nicht installiert ist bzw. noch unbekannt ist, dann bitte zuerst weiter hinten lesen (→ [Installation und Beschreibung des Hilfs-Programms uPyCraft](#)).

Dies bringt auch ein eigenes Image mit. Ich bleibe hier bei dem offiziellen von der micro-python.org-Webseite.

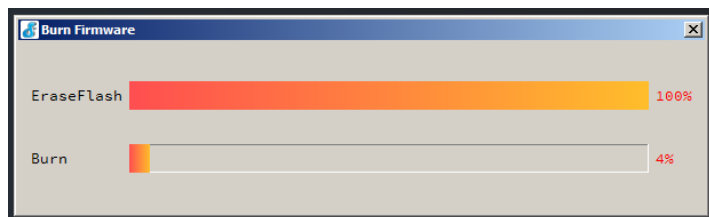
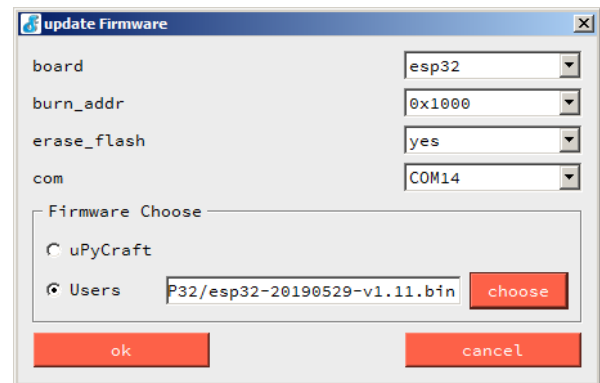
Die Lade-Adresse (burn_addr) ist offiziell die 0x1000. In einigen Anleitungen zu uPyCraft wird dagegen empfohlen, dem Programm die Entscheidung zu überlassen.

(Dann kann es sein, dass als Adresse die 0x0 angegeben ist. Bei mir klappte es mit beiden Adress-Angaben.)

Nach der Bestätigung beginnt das Löschen des Flash-Speichers auf dem ESP. Vorhandene Dateien gehen verloren.

Auf das Löschen folgt das Schreiben (Burn, Brennen) des MicroPython in den Speicher.

Diese Alternative zum oben beschriebenen Flashen ist vor allem dann interessant, wenn das uPyCraft dann zur Verfügung steht und man mal wieder ein neues MicroPython aufsetzen muss.



notwendige Treiber

<https://github.com/Tasm-Devil/Micropython-Tutorial-for-esp32/archive/master.zip>

downloaden und entpacken

so in das Dateisystem kopieren, dass der workspace-Ordner als Unterordner im Programm-Ordner der uPyCraft.EXE liegt

alternativ den workspace-Ordner anders einstellen

Zum ersten Ausprobieren reicht die serielle Konsole. Irgendwann müssen wir Dateien auf den ESP transferieren. Dazu benötigt man ebenfalls ein spezielles Programm.

Das Programm **uPyCraft** ist ein sehr flexibles Werkzeug zum Arbeiten mit MicroPython auf einem Microcontroller.

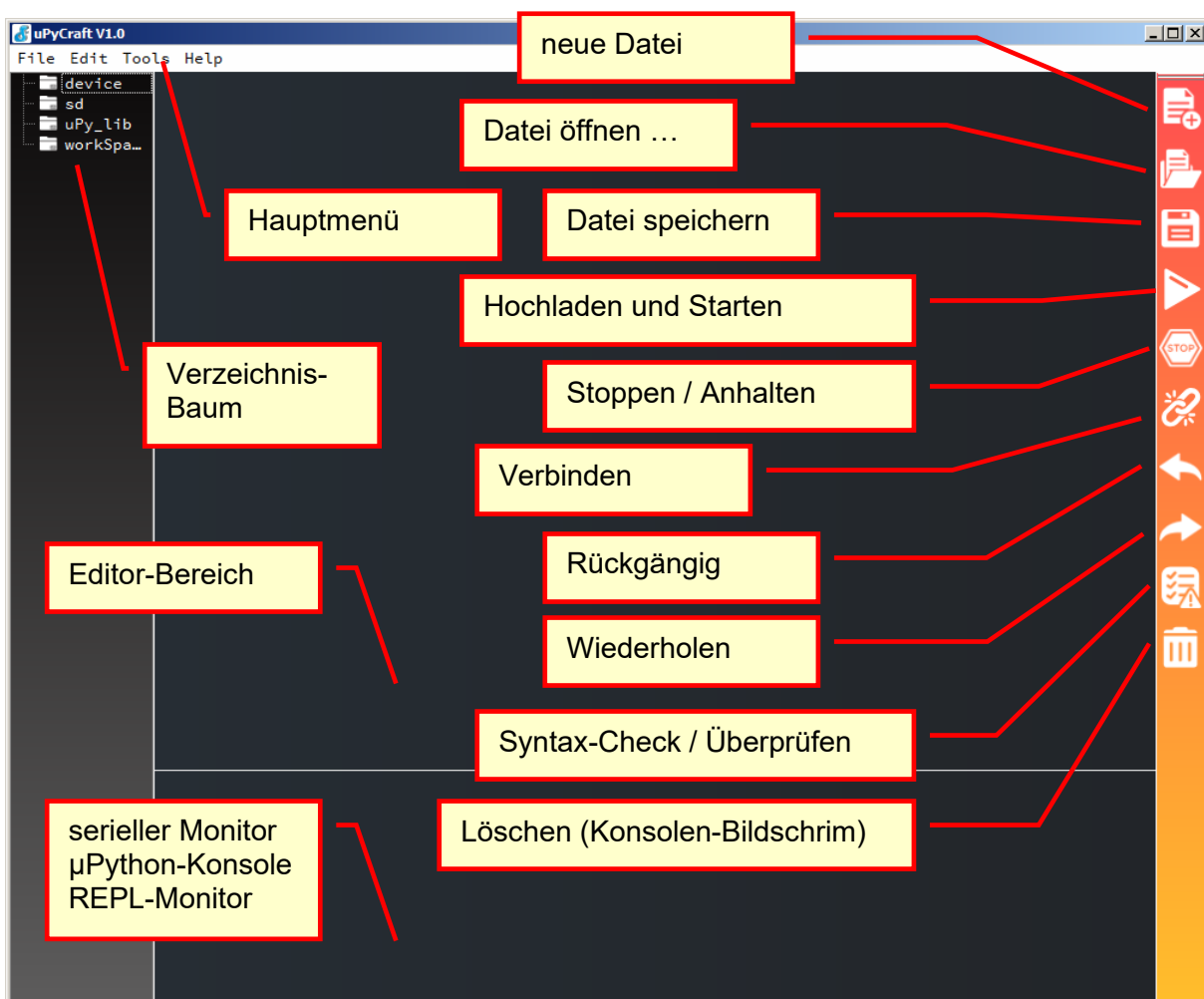
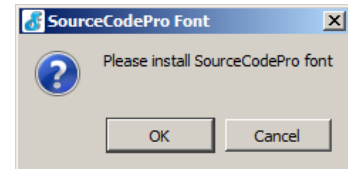
Installation und Beschreibung des Hilfs-Programms uPyCraft

Das u im Namen des Programms steht dabei für μ - also micro. Im Internet hat sich diese Ersetzung bei vielen Projekten manifestiert.

Der Download erfolgt von der Seite <https://randomnerdtutorials.com/uPyCraftWindows>. Man erhält eine funktionsfähige EXE. Diese kann an eine beliebige Stelle kopiert werden – u.a. auch auf einen USB-Stick mit einem portableApps-System.

Nach dem Start der EXE kommt eine Bitte, eine spezielle Schrift-Art zu installieren.

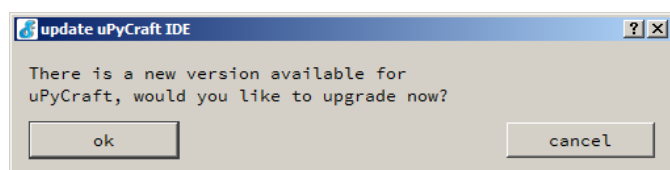
Das kann man tun. Gleich dannach öffnet sich das Programm-Fenster



U.U. bietet uPyCraft jetzt an, eine aktuellere Version herunterzuladen.

Bei mir brach das Upgrade immer mit einer Fehler-Meldung ab.

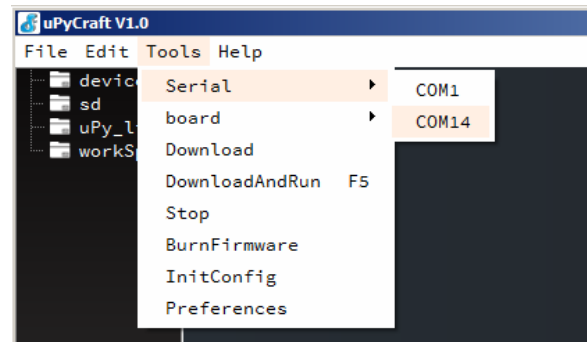
Auf der Projekt-Webseite war aber auch keine neuere Version aufgelistet.



ev. kommt auch noch eine Nachfrage, ob man Beispiel-Dateien aktualisieren will. Auch die sollte man tun.

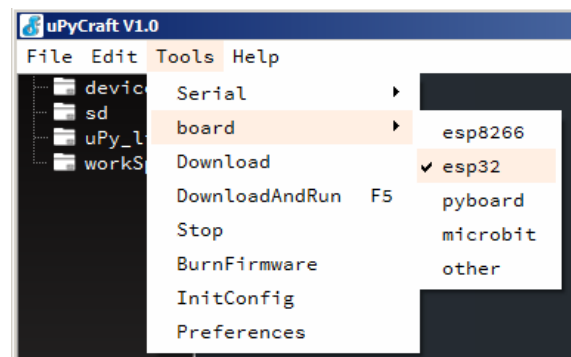
Die grundlegenden Werkzeuge aus der rechten Symbol-Leiste und die wichtigsten Elemente des Programms sind in der obigen Abbildung aufgezeigt.

Zum Testen der IDE muss zuerst einmal der richtige COM-Port unter "Tools" "Serial" ausgewählt werden. Üblicherweise ist es einer mit einer höheren Nummer. Ist nur COM1 verfügbar, dann sollte man den USB-Treiber aktualisieren.



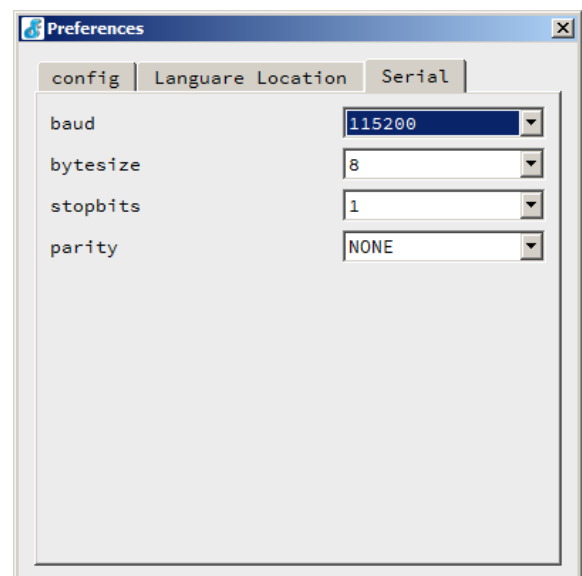
Als nächstes wählt man bei "Tools" "board" das zu benutzende Board.

Wer ein microbit mit Python programmieren will, wird hier auch fündig.



"Tools" "Preferences"

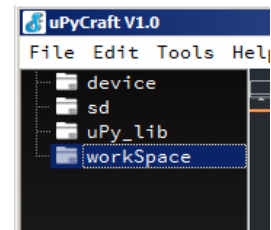
eigentlich nur der Reiter "Serial" interessant, da hier ev. die Übertragungs-Parameter für die USB-seriell-Schnittstelle eingestellt werden.



Klick auf "workSpace" führt zur Ordner-Auswahl. Hier bestimmt man einen Ordner, indem sich der "WorkSpace" – also der "Arbeits-Ordner" befindet.

Nicht den "workSpace"-Ordner selbst auswählen, sondern das übergeordnete Verzeichnis, in dem sich eben "workSpace" befindet.

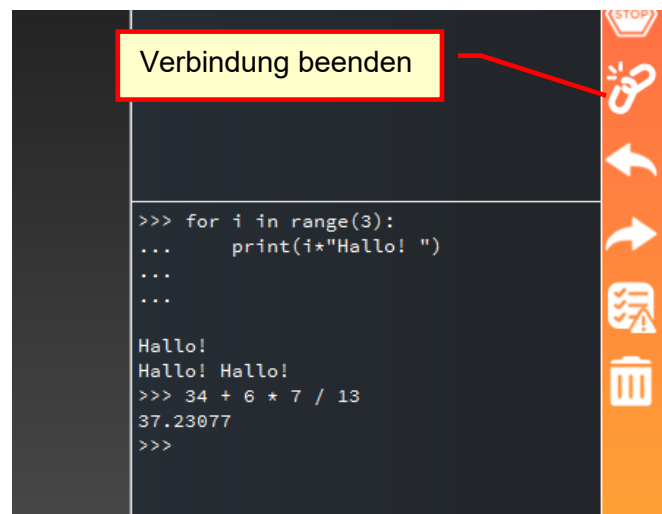
In workSpace müssen sich die Ordner und Dateien befinden, die wir uns von github (<https://github.com/Tasm-Devil/Micropython-Tutorial-for-esp32/archive/master.zip>) runtergeladen und dann entpackt haben.



Mit "Connect" stellt man die Verbindung zur MicroPython-Konsole dar. Hier arbeitet man dann im REPL-Modus.

Praktisch entspricht dies dem interaktiven Modus des "großen" Python.

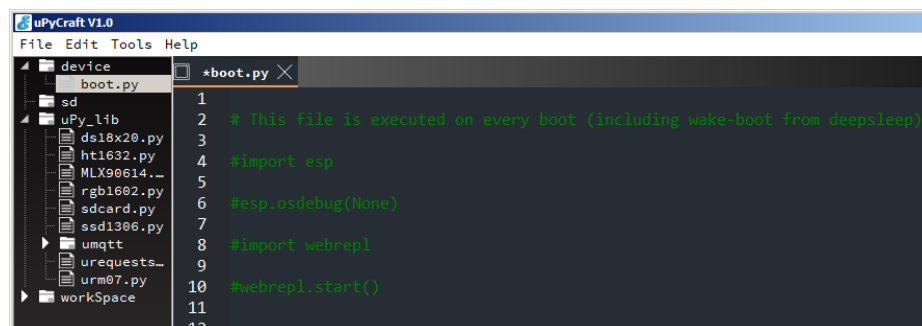
Über "Disconnect" (Verbindung beenden) wird die Kommunikation zum ESP beendet. Dies ist z.B. notwendig, wenn wir andere Aufgaben – wie das Hochladen von Dateien – durchführen wollen. Es ist immer nur eine Verbindung über den USB-Anschluß möglich.



??? Übertragen einer neuen Firmware

Im "Burn Firmware"-Dialog beim board "esp32" einstellen und erase_flash auf "yes" setzen, dann bestätigen

Ein frisch geflash-tes MicroPython bringt schon eine **boot.py** mit. Diese können wir für unsere Zwecke anpassen.



Die meisten Treiber-Dateien aus dem workSpace-Ordner müssen auf den ESP kopiert werden.

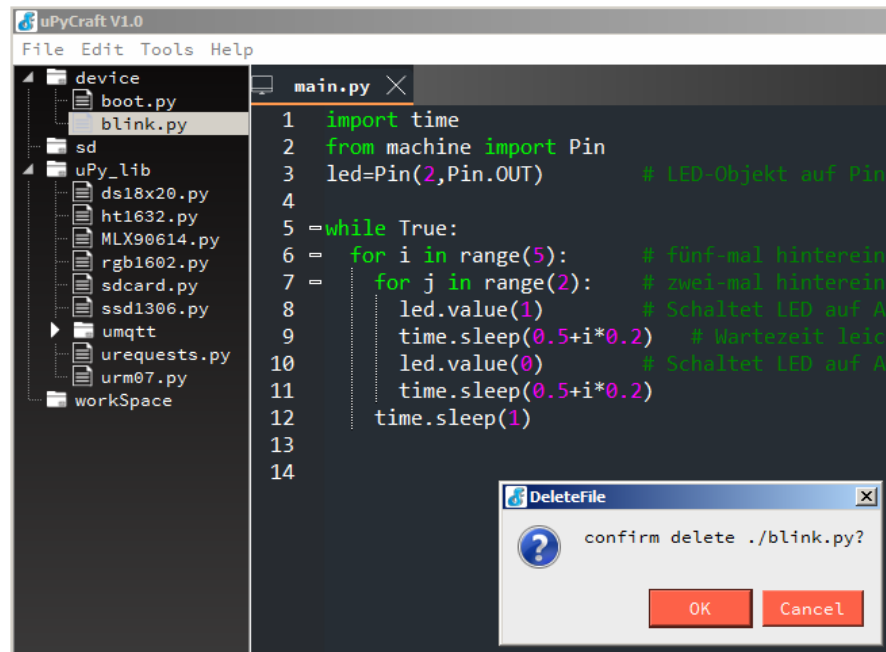
Einen neuen Ordner legt man durch Rechts-Klick auf das übergeordnete Verzeichnis an. Es gibt im Kontext-Menü nur den einen Eintrag "New dir".

Sollte sich der Verzeichnis- und Datei-Baum nach Aktionen nicht ändern, dann frischt ein "File" "Reflush Directory" die Ansicht wieder auf.

Löschen einer Datei
auf dem ESP

Der Ordner "device"
im Verzeichnis-Baum
links zeigt uns die
vorhandenen Dateien
auf dem Microcontrol-
ler.

*bei mir gab es Feh-
lermeldung und kein
Löschen*



Bearbeiten von Dateien im oberen mittleren
Bereich (Editor) möglich.

Das "Speichern unter ..." (Save as) erfolgt
standardmäßig im Workspace.

Da die Start-Programme immer main.py hei-
ßen müssen, bietet sich im Workspace ein
Abspeichern in einem Projekt-Ordner an.

Deutsche Umlaute usw. werden bei einem erneuten Laden (Open = Öffnen) in chinesische
Schriftzeichen umgesetzt. Man sollte also durchgehend – auch bei den Kommentaren – auf
spezielle Zeichen verzichten.

Das eigentliche Laden der Programm-
Dateien usw. erfolgt über "Hochladen und
Starten" (DownloadAndRun).

Vorher sollte man nochmal die "Verbindung"
(Tools → Serial und → board) überprüfen.
Bestimmte fehlende Angaben führen auch
schnell mal zu undefinierten Zuständen, in
denen dann nur noch ein vollständiger Neu-
start des Systems (oder gar ein Neuflashen des
MicroPython) hilft.

Die Bezeichnung "Download" ist sicher et-
was ungewöhnlich, wir sprechen eher von
Upload (Hochladen). Vielleicht ist es aus der
Sicht des ESP-Systems bzw. des MicroPy-
thon gedacht.

nach dem erfolgreichen Hochladen (Download) erhält man im seriellen Monitor die Anzeige:

```
exec("main.py",results())
```

meine Programme führten – auch nach einem Reset am ESP zu keiner Reaktion

*ganz im Gegenteil, es kam zum dauerhaften Abbruch / Verklemmen der USB-Verbindung
es half nur noch Neuflashen des ESP mit MicroPython's (funktionierte aber mit uPyCraft)*

Wenn uPyCraft anstandslos läuft und alle Aufgaben machbar sind, dann kann jetzt direkt bei
→ [10.6.x.1. Arbeiten mit MicroPython](#) weitergelesen werden.

Will oder kann man uPyCraft nicht benutzen, dann bleiben einige Kommandozeilen-
Programme übrig, die zum Hochladen von Dateien auf den ESP gedacht sind.

Hochladen von Dateien z.B. möglich mit ampy (Adafruit MicroPython Tool)

Installation über:
pip install adafruit-ampy

Hilfe aufrufen:
ampy --help

in Windows einstellen des AMPY_PORT über:
set AMPY_PORT=COM1

so ähnlich können auch AMPY_BAUD und AMPY_DELAY gesetzt werden
set AMPY_BAUD=115200
set AMPY_DELAY=0.5

die Hilfe zu **ampy**:

```
$ ampy --help
Usage: ampy [OPTIONS] COMMAND [ARGS]...
  ampy - Adafruit MicroPython Tool
  Ampy is a tool to control MicroPython boards over a serial
  connection. Using ampy you can manipulate files on the board's
  internal filesystem and even run scripts.
Options:
  -p, --port PORT  Name of serial port for connected board. Can
                   optionally specify with AMPY_PORT environemnt
                   variable. [required]
  -b, --baud BAUD  Baud rate for the serial connection (default
                   115200). Can optionally specify with AMPY_BAUD
                   environment variable.
  --version        Show the version and exit.
  --help          Show this message and exit.
Commands:
  get  Retrieve a file from the board.
  ls   List contents of a directory on the board.
  mkdir Create a directory on the board.
  put  Put a file or folder and its contents on the...
  reset Perform soft reset/reboot of the board.
  rm   Remove a file from the board.
  rmdir Forcefully remove a folder and all its...
  run  Run a script and print its output.
```

10.6.x.0.2. Nutzung eines ESP mit microPython unter Linux

Arbeiten im REPL-Modus

```
$ cu -l /dev/ttyUSB0 -s 115200
```

```
>>> import machine
>>> pin = machine.Pin(5, machine.Pin.OUT)
>>> pin.value(True)
```

Äquivalent zu Blinky

```
>>> import machine
>>> import time
>>> pin = machine.Pin(5, machine.Pin.OUT)
>>> while True:
...     pin.value(True)
...     time.sleep(1)
...     pin.value(False)
...     time.sleep(1)
```

Nutzung des Filesystems

```
>>> import os
>>> os.listdir()
['boot.py']
>>> datei = open('hallo.txt', "w")
>>> datei.write("Hallo Welt!")
11
>>> datei.close
>>> os.listdir()
['boot.py', 'hallo.txt']
```

```
>>>
```

weiterhin Arbeiten mit WebREPL, den mpy-utils oder upip (micropython package manager) möglich

notwendige Dateien und Verzeichnisse der ESP32 Repo-Files

/
drivers/
extmod/
lib/
mpy-cross/
py/
tools/

sowie Ordner für die konkrete Plattform, z.B.

esp32/

- Makefile
- modesp.c
- modmachine.c
- modnetwork.c
- modsocket.c
- moduos.c
- modutime.c

```
>>> import esp
>>> dir(esp)
['__name__', 'flash_read', 'flash_write', 'flash_erase',
'flash_size', 'flash_user_start']
>>> esp.flash_size()
4126345
```

esp-idf/components/spi_flash/include/esp_spi_flash.h

Beispiel für die c-
Funktions-
Deklarationen in den
Header-Dateien

```
size_t spi_flash_get_chip_dir();
```

esp32/modesp.c

c-Code

```
#include "esp_spi_flash.h

STATIC mb_obj_t esp_flash_size(void){
    return mp_obj_new_int_from_uint(spi_flash_get_chip_size());
}
STATIC MP_DEFINE_CONST_FUN_OBJ_0(esp_flash_size_obj,
esp_flash_size);

STATIC const mp_rom_map_elem_t esp_module_globals_table[] = {
    { MP_ROM_QSTR(__name__), MP_ROM_QSTR(MP_QSTR_esp) },
    { MP_ROM_QSTR_flash_size), MP_ROM(&esp_flash_size_obj) },
};
STATIC MP_DEFINE_CONST_DICT(esp_module_globals,
esp_module_globals_table);

const mp_obj_module_t esp_module = {
    .base = {},
    .globals = (mp_obj_dict_t*)&esp_module_globals,
};
```

esp/mpconfigport.h

```
extern const struct _mp_obj_module_t esp_module;

#define MICROPY_PORT_BUILTIN_MODULES \
    { MP_OBJ_NEW_QSTR_esp), (mp_obj_t)&esp_module }, \
```

esp32/Makefile

```
SRC_C = \ modesp.c \
```

```
>>> import esp
>>> dir(esp)
['__name__', 'flash_read', 'flash_write', 'flash_erase',
'flash_size', 'flash_user_start']
>>> esp.flash_size()
4126345
```

10.6.x.0.3. Esp-Tool

Download als GitHub-ZIP von <https://github.com/espressif/esptool>

Installation mit

`pip install esptool`

wenn das nicht funktioniert kann man auch:

`python -m pip install esptool`

oder:

`pip2 install esptool`

probieren. (ev. auch vorher pip aktualisieren())

Ganz neue Versionen des ESP-Tool's müssen manuell installiert werden. Das sollte aber den Profi's vorbehalten sein. Es handelt sich dann meist um frische Entwicklungs-Version, deren Stabilität nicht sicher ist. Die stabilen Version sind immer über pip installierbar. Für eine manuelle Installation gibt man:

`python setup.py install`

Gleiches kann man mit pySerial machen:

`pip install pyserial` oder `easy_install pyserial` oder `apt-get install python-serial`

Letzteres funktioniert natürlich nur unter Linux.

Die ESPtool's stellen die folgenden Kommando's zur Verfügung

ESPtool-Kommando's

- **verify_flash**
- **dump_mem**
- **load_ram**
- **read_mem**
- **write_mem**
- **read_flash_status**

- **write_flash_status**
- **chip_id**
- **make_image**
- **run**

Achtung! Die ESP's nutzen 3,3V-TTL-Spannung, während die üblichen Geräteschnittstellen 5V (Standard RS-232) benutzen. Hier muss also ein passender Adapter verwendet werden!

(Zwischen dem Standard-Pin (5V) und dem ESP-Pin (3V3) kommt ein Widerstand von 1kΩ und auf der ESP-Seite zwischen dem ESP-Pin und Ground ein 2,2 kΩ Widerstand. In der anderen Richtung – also bei der Datenübertragung vom ESP zur Standard-Schnittstelle benötigt man praktisch keine Anpassung, da die gelieferten 3,3 V als gültiges Signal akzeptiert wird.)

ESP-Pin	serieller Pin (Host-Schnittstelle)
TX (GPIO1)	RX (empfangen)
RX (GPIO3)	TX (senden)
Ground (GND)	Ground / Masse

Wenn die Kommandozeilen-Version (reines ESPTool) nicht so liegt, kann auch das Programm ESP8266-Flasher (→<http://www.dietrich-kindermann.de/Downloads/ESP8266-Flasher-x32-Installer.zip>) benutzen. Im Vorfeld muss allerdings die graphische Oberfläche wxPython installiert werden, da diese vom Flasher benutzt wird.

```
python -m pip install wxpython
```

Um das den ESP8266-Flasher unabhängig von einer Python-Installation (- also z.B. auf einem Fremd-Rechner -) benutzen zu können, kann man sich mit dem PyInstaller ein selbstständiges Installations-Paket erstellen.

Dazu wird zuerst PyInstaller installiert:

```
python -m pip install pyinstaller
```

Der PyInstaller benötigt noch den UPX-Packer (→<https://upx.github.io/>). Dies ist ein klassisches Pack- und Entpack-Programm, dass sich auf ausführbare Pakete spezialisiert hat.

Alternativ kann des NSIS-Installer benutzt werden.

Backup und Restore (Sichern und Wiederherstellen) der offiziellen Firmware von einem ESP-Microcontroller

Wenn noch nicht geschehen, ESPTool installieren. Dazu in der Konsole in den Ordner mit dem entpackten ESPTool wechseln (oder im Windows-Explorer / Arbeitsplatz / Computer) bei gedrückter [↑]-Taste das Kontext-Menü zum Ordner öffnen und dann "Eingabeaufforderung hier öffnen" auswählen.

```
python setup.py install
pip install pyserial
```

angenommen es handelt sich um einen 4MB-Flash-Speicher auf dem ESP und der ESP hängt am USB-Port COM8, dann lauten die Befehle

```
python esptool.py -b 115200 -port COM8 read_flash 0x000000 0x400000 ↵
flash4M.bin
```

```
python esptool.py erase_flash
```

```
python esptool -b 115200 -port COM8 write_flash -flash_freq 80m 0x000000  
flash4M.bin
```

Die variablen Teile sind farblich hervorgehoben. Die Datei-Namen (bin-Dateien) sind hier natürlich nur Beispiele.

10.6.x.1. Arbeiten mit MicroPython

optimale ESP32-Hardware sind die WROVER-Versionen, da sie zusätzlichen Speicher on-board haben

dieser spRAM ist für größere Python-Programme dann auch notwendig

umgesetzt wurde Python 3 in einer abgespeckten – aber prinzipiell funktionsfähigen – Version

bei anderen Microcontrollern muss immer genau geprüft werden, was geht und was nicht

MicroPython-System muss einmalig auf den Microcontroller gespielt werden

dann gibt es zwei Betriebs-Möglichkeiten

Nutzungs-Möglichkeiten von MicroPython auf einem Microcontroller

- **interaktiver Interpreter** REPL-Console (Read-Evaluate-Print-Loop)
 - lokale Version (→ [10.6.x.y.1. interaktiver Modus - REPL](#))
 - Internet-fähige / Netzwerk-Version (→ [10.6.x.y.2. interaktiver und Internet-fähiger Modus - WebREPL](#))

- **AutoRun-Modus**
 "AutoStart"-Modus führt nach dem Reboot / Reset automatisch die **boot.py** und die **main.py** aus
 ansonsten können beliebige Dateien im Board-eigenen Dateisystem bearbeitet / genutzt werden
 (→ [10.6.x.y.3. "Autostart"-Modus](#))

in der boot.py lassen sich z.B. bestimmte Hilfs-Funktionen / -Makro's / WLAN initiieren usw. ablegen, die beim Starten des ESP abgearbeitet werden

10.6.x.y.1. interaktiver Modus - REPL

praktisch identisch mit dem interaktiven Modus des normalen Python-Interpreter's

REPL steht für Read-Evaluate-Print-Loop

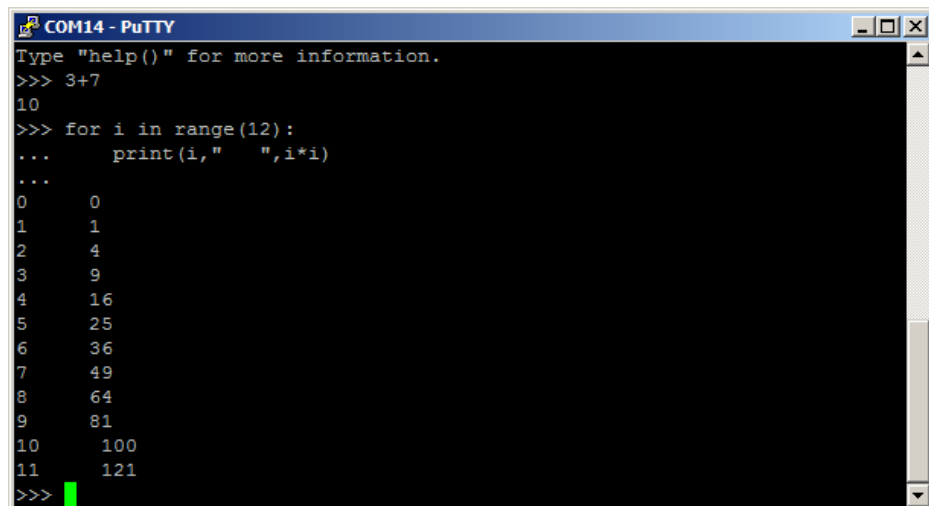
der MicroPython-Interpreter liest die Kommandozeile (Read), überprüft und übersetzt dann das Kommando (Evaluate), was letztendlich zu einer Reaktion (üblich wohl eine Ausgabe mit `print()`)

Das ganze läuft – wie üblich für Microcontroller – in einer Endlos-Schleife (Loop).

Die Menü-Befehle fehlen im REPL-Modus, so dass hier keine Erstellung oder Nutzung von Quellcode-Dateien erfolgen kann. Das muss man extern auf einem echten Rechner mit Editor oder Python-System erledigen.

nach der Verbindung über eine serielle Konsole, können die üblichen interaktiven Befehle oder Programm-Strukturen erledigt werden

hier zwei kurze und einfache Beispiele



```
COM14 - PuTTY
Type "help()" for more information.
>>> 3+7
10
>>> for i in range(12):
...     print(i, "    ",i*i)
...
0    0
1    1
2    4
3    9
4   16
5   25
6   36
7   49
8   64
9   81
10  100
11  121
>>>
```

viele Tools zum Arbeiten mit dem MicroPython nutzen genau diesen Modus und vereinfachen nur die Nutzung

man braucht dann i.A. nur noch ein Programm (eben dieses Tool), um sinnvoll mit dem Microcontroller zu arbeiten

Beispiele sind uMyCraft, ...

Im REPL-Modus sind vor allem die internen Sensoren sowie die anschließbaren Sensoren und Aktoren interessant. Man kann die verschiedensten Busse und Port frei nutzen. Ein isoliertes Arbeiten des Microprocessors mit den Sensoren und Aktoren ist so nicht wirklich möglich. Dafür muss man dann den "Autostart"-Modus (→ [10.6.x.y.3. "Autostart"-Modus](#)) verwenden.

10.6.x.y.2. interaktiver und Internet-fähiger Modus - WebREPL

benötigt wird ein Web-Client, den man unter <https://github.com/micropython/webrepl> downloaden bzw. gehostet unter <http://micropython.org/webrepl> nutzen kann

```
import webrepl_setup  
Konfigurieren des Web-Clients
```

```
import webrepl  
webrepl.start()
```

```
webrepl.start(password='meinPaswort')
```

10.6.x.y.3. "Autostart"-Modus

der Modus heißt nicht wirklich so, der Name beschreibt aber schon, was hier passiert
ein Python-Programm – abgespeichert als **main.py** – wird automatisch nach einem Reboot
gestartet und ausgeführt

zum Boot-System gehört auch eine weitere mögliche Python-Datei, die **boot.py**. In dieser
können noch vor dem Aufruf der main.py bestimmte Einstellungen gemacht und Vorberei-
tungen getroffen werden.

MicroPython-Tools

- **esptool.py** notwendig, um das MicroPython-Image (minimales Be-
triebssystem (Miniatur-RealTime-OS vom Board-Hersteller) und
MicroPython-Interpreter) auf das Board zu flashen
Löschen des Flash-Speichers

- **mpy-utils**
 - **mpy-fuse** mounten des ESP als beschreibbares Dateisystem

 - **mpy-upload** hochladen einer Datei auf den ESP

- **Terminal** klassisches Terminal (serieller Monitor)
(Start mit: `screen /dev/ttyUSB0 115200`
Beenden mit: [Strg] [a] , [k]

esptool gibt es unter <https://github.com/espressif/esptool> zum Downloaden
Installation über pip
`pip install esptool`

bei Problemen, alternativ:
`python -m pip install esptool` oder `pip2 install esptool`

weiterhin manuelle Installation möglich:
`python setup.py install`

oder wiederum alternativ:
`pip install pyserial` oder `easy_install pyserial` oder `apt-get install python-serial`

mounten des Dateisystem ist etwas langsam

anders benannte Programme lassen sich aber auch von der MicroPython-Konsole mit:

`import FileName` (ohne **.py** (also quasi als Modul))

starten

unter Windows USB-Port-Angabe mit: `-p COM1`

scheinbar werden mit uPyCraft gedownloadete (hochgeladene) Python-Programme gleich gestartet
auf der Konsole steht dann

`exec(...)`

ein Umbenennen nach `main.py` scheint für den normalen Start nicht notwendig zu sein

ESP mit neuem Programm starten (unter Windows)

1. **main.py erstellen** z.B. mit IDLE oder einfachem Editor; "Start"-Datei
mus als **main.py** gespeichert werden
2. **ESP-Dateisystem mounten**
3. **main.py hochkopieren**
4. **ESP-Dateisystem unmounten**
5. **ESP resetten**

Ergebnisse können auf seriellen Monitor angezeigt werden (quasi Ausgabe-Bildschirm) geeignet ist z.B. PuTTY. Dieses Programm startet auch ohne Installation aus beliebigem Verzeichnis.

Für portableApps gibt es eine eingebaute Version, die über das portableApps-System auch automatisch geupdatet wird.

Aber es sind natürlich auch andere seriellen Konsolen geeignet.

ESP mit neuem Programm starten (unter Linux (auch Raspberry Pi möglich))

1. **main.py erstellen** z.B. mit IDLE oder einfachem Editor; "Start"-Datei
mus als **main.py** gespeichert werden
2. **ESP-Dateisystem mounten**

`mpy-fuse mnt`

`esp32-mount`
3. **main.py hochkopieren**

`mpy-upload Datei`
4. **ESP-Dateisystem unmounten**

`fusermount -u mnt`
5. **ESP resetten**

mit esp32-terminal auf seriellen Monitor zum ESP zugreifen

Abfrage von freiem Speicher etc.

```
import gc
gc.mem_free()
```

verfügbare Funktionen über gc. abfragbar (Code-Ergänzungs-System)

Garbage-Collection anstoßen
`gc.collect()`

10.6.x.4. elementare Programmierung mit MicroPython

In den folgenden Kapiteln besprechen wir die Programmierung mit Python auf einem Microcontroller (hier vorrangig der ESP32). Das ist eine Wiederholung vieler Abschnitte und Themen von weiter vorne in diesem Skript. Ich möchte hier aber eine auskoppelbare Einheit für Nutzer erstellen, die sich nur mit Microcontrollern und MicroPython auseinandersetzen wollen oder müssen.

Wer die Grundlagen nicht mehr braucht und sich gleich mit den Spezialitäten der Microcontroller beschäftigen möchte kann jetzt zu → springen.

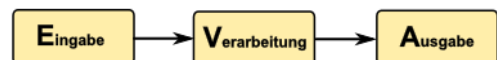
Unter elementarer Programmierung verstehe ich nur einfachste Elemente einer Programmiersprache, die zu den absoluten Grundlagen zählen. Sie folgt gleich im nächsten Abschnitt (→ [10.6.x.4. elementare Programmierung mit MicroPython](#)). Dazu gehören vorrangig Ein- und Ausgaben (auf Konsolen-Niveau) sowie einfache Verzweigungen und Schleifen. Python ist hier nur das spezielle Mittel.

Die klassische Programmierung (→ [10.6.x.5. klassische Programmierung mit MicroPython](#)) beschäftigt sich aus meiner Sicht mit Listen, Wörterbücher (Dictionary's) Funktionen, Objekten usw. Sie gehören zu einem Niveau, bei dem die modernen Aspekte der Programmierung sowie die speziellen Möglichkeiten von Python eine Rolle spielen. Der Python-grundgebildete Leser wird hier hin und wieder die Einschränkungen des MicroPython spüren. Für alle anderen ist es die Besprechung einer Leistungs-fähigen Programmiersprache.

Im Anschluß daran folgt die Geräte-nahe Programmierung. Hier kommen nun die Merkmale und Fähigkeiten der Microcontroller deutlich zum Vorschein. Deshalb nennen ich das auch spezielle Programmierung (→ [10.6.x.6. spezielle Programmierung mit MicroPython](#)). Es ist nicht auszuschließen, dass sich dieser Teil nicht – so wie dargestellt – auf jeden anderen Microcontroller übertragen läßt.

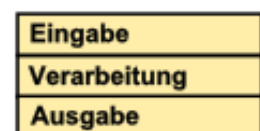
10.6.x.4.1. Ausgaben

Abweichend vom EVA-Prinzip beginnen wir mit den Ausgaben. Dies sollten wir können, damit andere Leistungen eines Programm's von uns zumindestens kontrolliert werden können.



Ein Programm ohne Ausgaben ist praktisch nutzlos. Dabei müssen Ausgaben nicht immer auf dem Bildschirm erfolgen. Oft werden Daten auch ausgedruckt, in eine Datei gespeichert oder einem Aktor (z.B. Ein- und Auschalten einer LED) zugewiesen.

In der Programmierung hat sich die Block-Darstellung in sogenannten Struktogrammen durchgesetzt. Sie dienen der Veranschaulichung von Algorithmen (Programm-Abläufen) unanhängig von einer konkreten Programmiersprache. Gute Struktogramme lassen sich in sehr viele Programmiersprachen übertragen.



Struktogramme sind immer große Blöcke (Rechtecke), die intern in kleinere unterteilt werden (können).

Man liest ein Struktogramm immer von oben nach unten. D.h. im Beispiel beginnt das Programm mit der Eingabe. Es folgt eine Verarbeitung (der Daten) und schließt mit einer Ausgabe ab.

Der klassische Ausgabe-Befehl in Python ist **print()**. In die Klammern können Komma-getrennt mehrere verschiedenartige Ausdruck-Elemente notiert werden.

Schauen wir uns zuerst den Ausdruck jeweils eines einzelnen Elementes an, um dann die Zusammenstellung zu längeren Ausdrücken zu besprechen.

Ausdruck-Element	Beispiel	Erläuterung / Bemerkungen / Hinweise
Text / Zeichenkette Verkettungen	<pre>print("Hallo Welt!")</pre> <pre>print("Hallo "+"Welt!")</pre>	beide print-Befehle erzeugen ein: Hallo Welt! auf dem Bildschirm bei einer Verkettung werden Zeichenketten mit einem Plus verbunden
Zahl Berechnung	<pre>print(24)</pre> <pre>print(2.713)</pre> <pre>print(24+7)</pre>	druckt 24 auf dem Bildschirm druckt die Kommazahl 2.713 aus (die Zahlen-Darstellung entspricht dem englischen Stil / Format! Der Punkt ist der Dezimal-Trenner) druckt den berechneten Betrag von 24+7, also 31 , aus
Variablen-Wert	<pre>print(PI)</pre> <pre>print(x)</pre> <pre>print(Hallotext)</pre>	druckt die Kreiszahl π aus: 3.1412 haben die Variablen x und Hallotext vorher einen Wert bekommen, dann wird dieser ausgedruckt, egal ob das eine Zahl oder ein Text ist ansonsten erscheint eine Fehler-Meldung

Der Unterschied zwischen Texten und Variablen ist die Notierung mit bzw. ohne Anführungsstriche. Statt den doppelten Anführungsstrichen sind auch einfache erlaubt. Sie müssen aber immer paarweise – also am Beginn und am Ende des Textes benutzt werden.

Jeder Befehl wird einzeln in die MicroPython-Konsole eingegeben. Nach einem [Enter] erscheint sofort die Ausgabe in der Zeile darunter. Dieses Wechselspiel von eingegebenen Befehlen und die sofortigen Ausgaben des Python-System's nennt man den **interaktiven Modus**.

Der Python-Kenner wird nun sagen, dass geht aber alles auch einfacher. Das stimmt! Praktisch hätten wir die print-Befehle und die zugehörigen Klammern weglassen können. Im interaktiven Modus sind sie nicht notwendig. Da wir aber später echte und vor allem größere Programme schreiben wollen, gewöhnen wir uns schon mal an die Schreibweise mit Befehl.

Aufgaben:

1. Prüfe zuerst anhand der oben angegebenen Informationen, ob die folgenden Befehle ordnungsgemäße Ausgaben (im interaktiven Modus) erzeugen! (Die Befehle der letzten Zeile werden hintereinander mit jeweils einem Enter eingegeben!)

- | | | |
|---|---|--|
| a) <code>print("Python")</code> | b) <code>print "Hallo"</code> | c) <code>print(24 + "")</code> |
| d) <code>write("Jetzt aber!")</code> | e) <code>print("Test Nr. 5)</code> | f) <code>print(3*""++)</code> |
| g) <code>x = 56</code>
<code>print(x)</code> | h) <code>print(3*x)</code>
<code>a = "4"</code>
<code>print(x+a)</code> | i) <code>c = 4</code>
<code>print("x+c")</code> |

2. Prüfen Sie nun die Befehle im MicroPython-System! Welche Überraschung(en) gibt es?

3. Berichtigen Sie die fehlerhaften Befehle und prüfen Sie diese im MicroPython!

4. Erzeugen Sie die folgenden Ausgaben!

- a) Drucken Sie den folgenden Satz als ganzes aus!
Python ist schon eine tolle Programmiersprache.
- b) Drucken Sie den Satz nun als Zusammensetzung von einzelnen Wörtern!
- c) Erzeugen Sie die Ausgabe Ihres Namens aus einzelnen Buchstaben!
- d) Lassen Sie Python das Ergebnis der folgenden Berechnung ausdrucken! Rechnen Sie zuerst im Kopf!
 $5 * 3 + (21 - 7) * -2 / 2$

Oben wurde schon erwähnt, dass sich mehrere Ausgabe-Elemente Komma-getrennt hintereinander in einem print-Befehl unterbringen lassen. Das kann man z.B. nutzen, um ordentliche Ergebnis-Sätze auszugeben. Die Ausgabe-Elemente können frei kombiniert werden. Auch deren Anzahl ist nicht begrenzt. Insgesamt sollte eine normale Ausgabe aber eine Zeile nicht überschreiten.

```
print("das Volumen des 5x5x5 Würfels beträgt: ", 5*5*5)
```

Zur allgemeinen Beschreibung von Befehlen benutzen wir gerne die folgende Syntyx-Darstellung:

```
print()  
print(ausgabeelement)  
print(ausgabeelement, ausgabeelement)  
print(ausgabeelement {, ausgabe} )
```

Die oberste Zeile beschreibt eine leere Ausgabe. Das entspricht einer Leerzeile. Die blau angezeigten Elemente / Zeichen sind notwendige Details (für die Info-Profi's: Terminale).

In der zweiten Zeile ist der Syntax einer einfachen Ausgabe dargestellt. Das kursiv geschriebene *ausgabeelement* ist ein Platzhalter für eine korrekte Ausgabe. Der Name könnte durch einen anderen ersetzt werden – z.B. *druckobjekt* od.ä. Hier sprechen wir dann von einem Nicht-Terminial.

Sollen zwei Ausgabeelemente ausgegeben werden, dann müssen sie durch ein Komma (,) voneinander getrennt in der Klammer aufgezählt werden. Das Komma ist also vorgeschrieben (also ein Terminal).

In der 4. Zeile wird mit den rot geschriebenen geschweiften Klammern ({ }) gekennzeichnet, was sich beliebig oft hintereinander wiederholen darf: immer ein Komma und dann ein neues

Ausgabeelement. Die geschweiften Klammern dienen hier nur zum Kennzeichnen der Wiederholung. Sie werden nicht mitgeschrieben. Sie sind sogenannte Meta-Symbole im Syntax. Die nächste Zeile ist die gemeinsame Syntax aller obigen Zeilen:

```
print([ ausgabeelement ] | ausgabeelement { , ausgabeelement } )
```

Die eckigen Klammern ([]) stehen für eine Option. Das Element kann sein, muss es aber nicht. Der senkrechte Strich (|) kennzeichnet die Alternative – also entweder das links oder das rechts vom Strich. Das Lesen von Syntax-Darstellungen ist zuerst immer etwas gewöhnungsbedürftig. Später wird es zum effektiven Mittel, um die Möglichkeiten von Befehlen effektiv darzustellen.

Aufgaben:

1. Realisieren Sie die nachfolgenden Ausgaben!

- a) Stellen Sie in einem print-Befehl den Text "Die Summe beträgt: " und die berechnete Zahl aus $34+41+26$ in einer Zeile zusammen!
- c) Geben Sie nachfolgende Berechnung so aus, dass die Zahlen separat im print-Befehl auftauchen (die sollen später durch Variablen ersetzt werden)! Die Berechnung des Ergebnisses darf direkt im print-Befehl erfolgen.
 $12,5 + 24 - 48 / 4 = \dots$
- b) Die obige Rechnung soll mit in den Text eingebunden werden. Dabei dürfen die Zahlen nicht in der Zeichenkette vorkommen, sondern müssen separat eingearbeitet werden!
- d)

2.

Ausgaben auf Aktoren usw. besprechen wir erst später (→ [10.6.x.6. spezielle Programmierung mit MicroPython](#)), da diese vom verwendeten Microcontroller abhängig sind.

???formatierte Ausgaben

10.6.x.4.2. Variablen, Zuweisungen und Berechnungen

Bei den Ausgaben haben wir schon nebenbei mit Variablen gearbeitet. Die am meisten verwendete ist sicher `x`. In Python-Programmen können wir alle Zeichenketten, die mit einem Buchstaben beginnen und dann von Buchstaben, Ziffern und dem Unterstrich gefolgt werden. Gute Programmierer verwenden sprechende Variablen, d.h. solche deren Namen ihren Verwendungszweck beschreibt. Zum Einen verbessert das die Lesbarkeit von Programmen und zum anderen werden Verwechslungen oder versehentliche Doppelbenutzungen vermieden. Oft werden durch die sinnvolle Benennung von Variablen ihre Rollen in Algorithmen deutlicher. Dadurch lassen sich Programmierfehler schneller finden.

Das Volumen in einer Formel kann also z.B. mit `x`, `v`, `V`, `volumen` oder `Volumen` als Variable benutzt werden. Es wird gleich klar, dass die beiden letzteren am besten zu verstehen sind. In Python-Programmen wird die Groß- und Kleinschreibung unterschieden. D.h., dass `volumen` und `Volumen` zwei unterschiedliche Variablen sind. Man sollte sich auf eine Art der Schreibung in seinen Programmen festlegen. Üblich sind klein-geschriebene Variablennamen. Mit Unterstrichen oder Groß-Buchstaben kann man längere Variablennamen wieder besser lesbar machen, z.B.:

`seitenSumme` oder `seiten_summe`

Beim ersten Benutzen muss einer Variable ein Wert zugewiesen werden. Das passiert ganz einfach mit einem Gleichheitszeichen (=). Der Variablenname muss links stehen, der Wert rechts. Als Werte kommen Zahlen und Berechnungen, Texte und Verkettungen oder andere Variablen infrage. Eine weitere Möglichkeit stellen Funktionen dar, wobei die Eingabefunktion (→ [10.6.x.4.3. Eingaben](#)) sicher die verständlichste ist. Gültige Variablen-Deklarationen sind:

```
x = 4
y = x
HalloText = "Good morning!"
seite = 12
wuerfelVolumen = seite * seite * seite
```

Dagegen ist es z.B. falsch, die folgenden Ausdrücke zu benutzen:

```
4 = x
Hallo
```

Beim Versuch, solche Ausdrücke zu übersetzen (zu interpretieren) erzeugt der Python-Interpreter eine Fehlermeldung.

Einige spezielle Zuweisungen (Listen, Objekte, ...) besprechen wir, wenn wir dort angekommen sind. Das Zuweisungs-Prinzip ist immer gleich.

Hat eine Variable erst einmal einen Wert, dann kann sie für Berechnungen und Funktionen benutzt werden. Die klassischen Rechen-Operationen haben wir ja schon so nebenbei mit vorgestellt.

Besonders muss vielleicht noch einmal auf die Multiplikation mit dem Sternchen (*) und die Division mit dem Schrägstrich (/) hingewiesen werden.

Besondere Operatoren sind zwei aufeinanderfolgende Sternchen (**) als Potenz-Operator und das Prozent-Zeichen (%) als Modulo-Operator. Die Modulo-Operation ist die Berechnung des Restes einer ganzzahligen Division. Man verwendet die Modulo-Operation z.B. zum Bestimmen von Teilbarkeiten (s.a. → [10.6.x.4.4. Alternativen, Verzweigungen](#)) oder in der Kryptographie.

Elementare Funktionen sind z.B. **sin()**, **cos()** oder **tan()**. Die Wurzel-Funktion wird mit **sqrt()** aufgerufen. Mit **abs()** erhält man den Absolut-Wert des Wertes in der Klammer. Die Werte in der Klammer einer Funktion sind die Argumente. Sie werden zur Berechnung des Funktionswertes benutzt. (Dazu gleich noch etwas mehr →). Dort stellen wir auch noch weitere Funktionen vor.

10.6.x.4.3. Eingaben

Eingaben von Sensoren usw. besprechen wir erst später (→ [10.6.x.6. spezielle Programmierung mit MicroPython](#)), da diese vom verwendeten Microcontroller abhängig sind.

10.6.x.4.4. Alternativen, Verzweigungen

10.6.x.4.5. Wiederholungen, Schleifen

10.6.x.4.6. eingebaute und mitgelieferte Funktionen

10.6.x.5. *klassische Programmierung mit MicroPython*

10.6.x.5.1. Listen und Listen-Verarbeitung

10.6.x.5.2. Wörterbücher, Dictionary's

10.6.x.5.3. Lesen und Schreiben von Dateien, Datei-Verarbeitung

10.6.x.5.4.

10.6.x.6. spezielle Programmierung mit MicroPython

10.6.x.4. weitere spezielle Programm-Beispiele und -Schnipsel

Besonders wichtige Bibliotheken / Module sind `machine` und `network`. Sie stellen Objekte und Funktionen / Methoden für die spezielle Hardware – den speziellen Microcontroller – zur Verfügung.

Weiterhin werden oft Bibliotheken bzw. Module zu den benutzten Sensoren und Aktoren benötigt. Diese ersparen uns viel Programmier-Aufwand.

Um den RAM-Verbrauch möglichst gering zu halten, sollte man es sich angewöhnen nur die unbedingt notwendigen Bestandteile aus den Modulen zu laden.

klassisches Einstiges-Programm "Blink"
läßt die onboard-LED blinken

```
from time import sleep
from machine import Pin

led=Pin(12,Pin.OUT)

while True:
    led.value(True)
    sleep(0.2)
    led.value(False)
    sleep(0.2)
```

einen Pin Pulsweiten-moduliert ansteuern
mögliche Werte von 0 bis 1023

klassische Anwendungen für diese Ansteuerungs-Art sind zu "dim-mende" LED's und Servo-Motoren

```
from time import sleep
from machine import Pin
from machine import PWM

led=Pin(12,Pin.OUT)

puls = PWM(led)
while True:
```

```
puls.freq(200)
sleep(1)
puls.freq(500)
sleep(2)
puls.freq(1000)
sleep(3)
```

einen Touch-Port abfragen
insgesamt 10 Touch-Eingabe möglich (0 .. 9)

```
from machine import TouchPad

touchpin = Pin(2)
touchpad = TouchPad(touchpin)
while True:
    print("Touch-Wert = ", touchpad.read())
    sleep(1)
```

LED-Ring (NeoPixel) ansteuern

LED-Streifen oder –Ringe usw. gehören heute zu den peppigen Accessoire's
zumeist sind hier die LED's einzeln ansteuerbar, bei einfarbigen LED's funktioniert dann
zumindestens das Ein- und Aus-Schalten
sind RGB-LED's verbaut, dann kann man eigentlich fast immer jede einzelne LED in einer
speziellen Farbe leuchten lassen
wichtig ist hier immer, dass nach dem Setzen / verändern von Werten, diese auf den NeoPi-
xel-Ring rausgeschrieben werden müssen

```
import machine, neopixel
import time
import random

def test(np):
    n = np.n
    b = 5      # Helligkeit
    sl = 10    # Kurzschlafzeit

    time.sleep_ms(1000)
    np.fill((0,0,0))
    time.sleep_ms(1000)

    for i in range(n):
        np[i] = (b,0,0)
        np.write()
        time.sleep_ms(sl)

    time.sleep_ms(1000)
    np.fill((0,0,0))
    time.sleep_ms(1000)

    for i in range(n):
        np[i] = (0,b,0)
        np.write()
        time.sleep_ms(sl)
```

```

time.sleep_ms(1000)
np.fill((0,0,0))
time.sleep_ms(1000)

for i in range(n):
    np[i] = (0,0,b)
    np.write()
    time.sleep_ms(sl)

time.sleep_ms(1000)
np.fill((0,0,0))
time.sleep_ms(1000)

for i in range(n):
    np[i] = (b,b,b)
    np.write()
    time.sleep_ms(sl)

def demo1(np):
    n = np.n

def demo2(np):
    n = np.n
    # Kreis
    for i in range(4*n):
        for j in range(n):
            np[j]=(0,0,0)
            np[i%n]=(255,255,255)
            np.write()
            time.sleep_ms(25)
    # Band
    for i in range(4*n):
        for j in range(n):
            np[j] = (0,0,128)
            if (i//n)%2 == 0:
                np[i%n] = (0,0,0)
            else:
                np[n-1-(i%n)] = (0,0,0)
            np.write()
    # Säubern
    for i in range(n):
        np[i] = (0,0,0)
    np.write()

def demo3(np):
    n = np.n
    b = 5    # Helligkeit
    sl = 1   # Kurzschlafzeit

    for i in range(10000):
        np.fill((0,0,0))
        i = random.randint(0,23)
        r = random.randint(0,100)
        b = random.randint(0,100)
        g = random.randint(0,100)
        np[i] = (r,g,b)
        np.write()
        time.sleep_ms(sl)

np = neopixel.NeoPixel(machine.Pin(15),24,timing=0)

test(np)
demo1(np)

```

```
demo2 (np)
demo3 (np)
```

Q: /µP_Q1/ (leicht geändert: dre)

kleine OLED-Display's sind bei einigen ESP-Bausteinen gleich mit aufgelötet. Sie ermöglichen die Anzeige einiger Text-Zeilen oder kleiner Grafiken
in den meisten Fällen sind die OLED's allerdings Monochrom, was aber für die einfachen Möglichkeiten unserer Microcontroller schon super ausreicht.

ansprechen des OLED-Display's (wenn vorhanden)

screen.py

```
from machine import I2C, Pin
import time

from ssd1306 import SSD1306_I2C

_i2c = I2C(sda=Pin(5), scl=Pin(4))
_display = SSD1306_I2C(128,64,_i2c)

def text(t):
    _display.fill(0) # OLED löschen
    lines = t.splitlines()
    y = 0
    for line in lines:
        _display.text(line,0,y)
        y += 10
    _display.show()
```

da OLED über den i2c-Bus an die Pin's 4 (Clock) und 5 (Data) angeschlossen ist, benötigt man den obigen "Treiber" für eine Text-Ausgabe

Hinweise: 128,64 stehen für Breite und Höhe des Display's in Pixel; Display wird als ganzes angesteuert, es erfolgt kein Scrollen

main.py

ein Sternchen im Ping-Pong-Modus über eine integrierte OLED-Anzeige wandern lassen

```
from time import sleep
from screen import text

def pingpong(i):
    txt('\n\n' + ' ' * i + '*' + ' ' * (15 - i) + '\n\n\n')
    sleep(0.1)

while True:
    for i in range(0,15,1):
        pingpong(i)
    for i in range(15,0,-1):
        pingpong(i)
```

Q: /µP_Q1/

Abfrage eines Licht-Sensors

```
from machine import Pin, I2C
from bh1750 import BH1750
from screen import text
from time import sleep

i2c=I2C(scl=Pin(14), sda=Pin(13))

sensor = BH1750(i2c)

while True:
    lum = sensor.luminance(BH1750.ONCE_HIRES_1)
    print("Lumineszenz =", lum)
    balken = "#" * int(lum/100)
    text("Lumineszenz-Sensor:\n\n%s\n % balken)
    sleep(0.5)
```

Q: /µP_Q1/ (leicht geändert: dre)

das Programm zeigt den aktuellen Meßwert auf dem seriellen Monitor an. Das OLED-Display wird zusätzlich zur Visualisierung der Lichtstärke aus Balken-Diagramm verwendet.

Wenn der ESP eins kann, dann ist das WLAN. Bei vielen Bausteinen ist gleich von der Herstellung schon ein kleines WLAN-Scan-Programm aufgespielt. Häufig werden die verschiedenen einfachen WLAN-Scanner auch als "Hallo Welt"-Programm der ESP-Welt verstanden.

WLAN-Scan

```
from network import WLAN, STA_IF
from time import sleep

wlan = WLAN(STA_IF)
wlan.active(True)

while True:
    nets = wlan.scan()
    print("Scan-Ergebnis: =====")
    for net in sorted(nets):
        print(net)
    print()
    sleep(2)
```

Q: /µP_Q1/ (leicht geändert: dre)

STA_IF ... steht für den Stations-Modus des WLAN (praktisch als Client eingerichtet)

funktionierende Funktion zum Verbinden des ESP mit einem AccessPoint
ESP fungiert als einfache WLAN-Station

```
def verbinden():
    import network

    ssid = "?????"
    password = "?????"
```

```

meinwlan = networ.WLAN(network.STA_IF)
meinwlan.active(True)
if not meinwlan.isconnected():
    print("Verbindung zum WLAN herstellen ...")
    meinwlan.conncet(ssid,password)
    while not meinwlan.connected():
        pass
    print("aktuelle Netzwerk-Konfiguration:",meinwlan.ifconfig())

```

als Funktion mit Argumenten könnte verbinden() auch so aussehen

```

def verbinden(ssid,password):
    import network

    meinwlan = ...

```

Empfang und Zurücksenden von UDP-Nachrichten / -Paketen (Echo-Funktion)

```

#include <ESP8266WiFi.h>
#include <WiFiUDP.h>

// The ESP-12 has a blue LED on GPIO2
#define LED 2

// Name and password of the access point
#define SSID "Pussycat"
#define PASSWORD "supersecret"

// The server accepts connections on this port
#define PORT 5444
WiFiUDP udpServer;

// Buffer for incoming UDP messages
char udp_buffer[WIFI_CLIENT_MAX_PACKET_SIZE+1];

/** Receive UDP messages and send an echo back */
void process_incoming_udp()
{
    if (udpServer.parsePacket())
    {
        // Fetch received message
        int len=udpServer.read(udp_buffer,sizeof(udp_buffer)-1);
        udp_buffer[len] = 0;

        // Display the message
        Serial.print(F("Received from "));
        Serial.print(udpServer.remoteIP());
        Serial.print(":");
        Serial.print(udpServer.remotePort());
        Serial.print(": ");
        Serial.println(udp_buffer);

        // Send echo back
        udpServer.beginPacket(udpServer.remoteIP(), udpServer.remotePort());
        udpServer.print(F("Echo: "));

```

```

        udpServer.print(udp_buffer);
        udpServer.endPacket();

        // Execute some commands
        if (strstr(udp_buffer, "on"))
        {
            digitalWrite(LED, LOW);
            udpServer.println(F("LED is on"));
        }
        else if (strstr(udp_buffer, "off"))
        {
            digitalWrite(LED, HIGH);
            udpServer.println(F("LED is off"));
        }
    }
}

/** Optional: Notify about AP connection status changes */
void check_ap_connection()
{
    static wl_status_t preStatus = WL_DISCONNECTED;

    wl_status_t newStatus = WiFi.status();
    if (newStatus != preStatus)
    {
        if (newStatus == WL_CONNECTED)
        {
            digitalWrite(LED, LOW);

            // Display the own IP address and port
            Serial.print(F("AP connection established, listening on
"));
            Serial.print(WiFi.localIP());
            Serial.print(":");
            Serial.println(PORT);
        }
        else
        {
            digitalWrite(LED, HIGH);
            Serial.println(F("AP connection lost"));
        }
        preStatus = newStatus;
    }
}

/** Runs once at startup */
void setup()
{
    // LED off
    pinMode(LED, OUTPUT);
    digitalWrite(LED, HIGH);

    // Initialize the serial port
    Serial.begin(115200);

    // Give the serial monitor of the Arduino IDE time to start
    delay(500);

    // Use an external AP
    WiFi.mode(WIFI_STA);
    WiFi.begin(SSID, PASSWORD);
}

```



```

    // Start the UDP server
    udpServer.begin(PORT);
}

/** Main loop, executed repeatedly */
void loop()
{
    process_incoming_udp();
    check_ap_connection();
}

```

Q: <http://stefanfrings.de/esp8266/>

TCP-Server

```

#include <ESP8266WiFi.h>

// The ESP-12 has a blue LED on GPIO2
#define LED 2

// Name and password of the access point
#define SSID "Pussycat"
#define PASSWORD "supersecret"

// The server accepts connections on this port
#define PORT 5333
WiFiServer tcpServer(PORT);

// Objects for connections
#define MAX_TCP_CONNECTIONS 5
WiFiClient clients[MAX_TCP_CONNECTIONS];

// Buffer for incoming text
char tcp_buffer[MAX_TCP_CONNECTIONS][30];

/**
 * Collect lines of text.
 * Call this function repeatedly until it returns true, which indicates
 * that you have now a line of text in the buffer. If the line does not fit
 * (buffer too small), it will be truncated.
 *
 * @param source The source stream.
 * @param buffer Target buffer, must contain '\0' initially before calling this function.
 * @param bufSize Size of the target buffer.
 * @param terminator The last character that shall be read, usually '\n'.
 * @return True if the terminating character was received.
 */
bool append_until(Stream& source, char* buffer, int bufSize, char terminator)
{
    int data=source.read();
    if (data>=0)
    {

```

```

        int len=static_cast<int>(strlen(buffer));
        do
        {
            if (len<bufSize-1)
            {
                buffer[len++]=static_cast<char>(data);
            }
            if (data==terminator)
            {
                buffer[len]='\0';
                return true;
            }
            data=source.read();
        }
        while (data>=0);
        buffer[len]='\0';
    }
    return false;
}

/** Optional: Notify about AP connection status changes */
void check_ap_connection()
{
    static wl_status_t preStatus = WL_DISCONNECTED;

    wl_status_t newStatus = WiFi.status();
    if (newStatus != preStatus)
    {
        if (newStatus == WL_CONNECTED)
        {
            digitalWrite(LED, LOW);

            // Display the own IP address and port
            Serial.print(F("AP connection established, listening on
"));
            Serial.print(WiFi.localIP());
            Serial.print(":");
            Serial.println(PORT);
        }
        else
        {
            digitalWrite(LED, HIGH);
            Serial.println(F("AP conection lost"));
        }
        preStatus = newStatus;
    }
}

/**
 * Put new connections into the array and
 * send a welcome message.
 */
void handle_new_connections()
{
    WiFiClient client = tcpServer.available();
    if (client)
    {
        Serial.print(F("New connection from "));
        Serial.println(client.remoteIP().toString());

        // Find a free space in the array
        for (int i = 0; i < MAX_TCP_CONNECTIONS; i++)

```

```

        {
            if (!clients[i].connected())
            {
                // Found free space
                clients[i] = client;
                tcp_buffer[i][0]='\0';
                Serial.print(F("Kanal="));
                Serial.println(i);

                // Send a welcome message
                client.println(F("Hello World!"));
                return;
            }
        }
        Serial.println(F("To many connections"));
        client.stop();
    }
}

/** Receive TCP messages and send echo back */
void process_incoming_tcp()
{
    static int i=0;

    // Only one connection is checked in each call
    if (clients[i].available())
    {
        // Collect characters until line break
        if (ap-
pend_until(clients[i],tcp_buffer[i],sizeof(tcp_buffer[i]),'\n'))
        {
            // Display the received line
            Serial.print(F("Empfangen von "));
            Serial.print(i);
            Serial.print(": ");
            Serial.print(tcp_buffer[i]);

            // Send an echo back
            clients[i].print(F("Echo: "));
            clients[i].print(tcp_buffer[i]);

            // Execute some commands
            if (strstr(tcp_buffer[i], "on"))
            {
                digitalWrite(LED, LOW);
                clients[i].println(F("LED is on"));
            }
            else if (strstr(tcp_buffer[i], "off"))
            {
                digitalWrite(LED, HIGH);
                clients[i].println(F("LED is off"));
            }

            // Prepare the buffer to receive the next line
            tcp_buffer[i][0]='\0';
        }
    }

    // Switch to the next connection for the next call
    if (++i >= MAX_TCP_CONNECTIONS)
    {
        i=0;
    }
}

```

```

}

/** Executes once during start*/
void setup()
{
    // LED off
    pinMode(LED, OUTPUT);
    digitalWrite(LED, HIGH);

    // Initialize the serial port
    Serial.begin(115200);

    // Give the serial monitor of the Arduino IDE time to start
    delay(500);

    // Use an external AP
    WiFi.mode(WIFI_STA);
    WiFi.begin(SSID, PASSWORD);

    // Start the TCP server
    tcpServer.begin();
}

/** Main loop, executed repeatedly */
void loop()
{
    handle_new_connections();
    process_incoming_tcp();
    check_ap_connection();
}

```

Q: <http://stefanfrings.de/esp8266/>



Links:

<http://docs/micropython.org/en/latest/esp8266/> (Dokumentation in Entwicklung, muss für ESP-32 interpretiert werden)
<https://randomnerdtutorials.com/getting-started-micropython-esp32-esp8266/> (online Arbeitsanleitung)

10.6.x.5. spezielle Module für ESP-32-Microcontroller

Info- und Quellcode-Q: docs.micropython.org/en/latest/library/index.html (Quellcode's leicht geändert)

Einige der Module sind auch für andere Microcontroller verfügbar. Meist sind diese Board-spezifisch, d.h. sie müssen auf der micropython-Website als Download-Paket genauestens ausgewählt werden.

In der jeweiligen Nutzung kann zu veränderten Notierungen und Varianten – im Vergleich zu den folgenden Darstellungen – kommen.

10.6.x.5.1. Modul "machine"

import machine

machine.freq()

liefert die aktuelle Prozessor-Frequenz zurück (in Hz)

machine.freq(240000000)

setzt die Prozessor-Frequenz auf 240 MHz

Deep-sleep-Modus ()

machine.deepsleep(100000)

Versetzt den Microcontroller für 100 s in den Tiefschlaf-Modus (Stromspar-Modus)
ohne Parameter wird der Microcontroller dauerhaft in den Tiefschlaf-Modus versetzt
ein weiteres Stromsparen ist durch Setzen / Einschalten von internen Pull-up-Widerständen möglich

p1 = Pin(4, Pin.IN, Pin.PULL_HOLD)

if machine.reset_cause() == machine.DEEPSLEEP_RESET:

print("Microcontroler ist aufgeweckt!")

RTC (realtime clock)

from machine import RTC

rtc = RTC()

rtc.datetime((Jahr, Monat, Tag, Stunde, Minuten, Sekunden, MilliSekunden))

über die Abfrage eines NTC-Servers ist eine recht genaue Zeitsynchronisierung möglich

rtc.datetime()

gibt das aktuelle Datum und die Zeit zurück

Zähler / Timer

from machine import Timer

Zaehler = Timer(-1)

Zaehler.init(period=1000, mode=Timer.ONE_SHOT, callback=lambda t:print(1))

Zähl-Einheit sind MilliSekunden

Zaehler.init(period=2000, mode=Timer.PERIODIC, callback=lambda t:print(2))

Pin's / GPIO

from machine import Pin

verfügbar sind die Pin's 0 .. 19, 21 .. 23, 25 .. 27 und 32 .. 39

abhängig von den Pin's, die auf dem Board nach außen geführt wurden (je nach Hersteller und Board-Art unterschiedlich)

weiterhin gilt:

- Pin 1 ist für TX und Pin 3 für RX der seriellen Verbindung über UART in Gebrauch
- die Pin's 6 .. 8, 11, 16 und 17 sind für die Verbindung mit dem eingebauten Flash-Speicher in Gebrauch und können nicht anderweitig verwendet werden
- Pin's 34 .. 39 sind nur als Input-Pin's nutzbar (und haben auch keinen internen Pull-up-Widerstand)
- der Pull-Wert kann mittels Pin.PULL_HOLD auf einen anderen Wert gesetzt werden, z.B., um sie Strom-sparend im DeepSleep-Modus zu nutzen

meinpin = Pin(4, Pin.OUT)

setzt Pin 4 bei der Initialisierung als Ausgabe-Port

meinpin = Pin(5, Pin.OUT, value=1)

setzt Pin 3 bei der Initialisierung als Ausgabe-Port sofort auf HIGH

meinpin.on()

schaltet den Pin auf HIGH

meinpin.value(0 | 1)

schaltet den Pin auf LOW bzw. HIGH

meinpin.off()

schaltet den Pin auf LOW

meinpin = Pin(6, Pin.IN)

meinpin.value()

gibt 0 oder 1 (für LOW bzw HIGH) zurück

meinpin = Pin(7, Pin.IN, Pin.PULL_UP)

aktiviert den internen Pull-up-Widerstand

meinpin = Pin(8, Pin.OUT, value=1)

PWM (pulse width modulation)

from machine import Pin, PWM

pulsweitmod = PWM(Pin(0))

pulsweitmod.freq()

pulsweitmod.freq(500)

pulsweitmod.duty()

pulsweitmod.duty(100)

pulsweitmod.deinit()

pulsweitmod2 = PWM(Pin(9), freq=10000, duty=1000)

Pulsweiten-Ausgabe in einer Funktion initialisieren

es sind Frequenzen von 1 Hz bis 40 MHz möglich

ADC (analog to digital conversion)

from machine import ADC

analdigwand = ADC(Pin(32))

Eingangsspannungs-Pegel: 0 .. 1,0 V; Auflösung auf 12 bit → Ergebniswerte: 0 .. 4'095

analdigwand.read()

analdigwand.atten(ADC.ATTN_11DB)

Einschalten einer Dämpfung → Eingangsspannungs-Pegel: 0 .. 3,6 V

weitere zugelassene Dämpfungswerte: ATTN_0DB (bis 1,0 V), ATTN_2_5_DB (bis 1,34 V), ATTN_6DB (bis 2,0 V)

analdigwand.width(ADC.WIDTH_9BIT)

Ändern der Auflösung auf 9 bit → Ergebniswerte: 0 .. 511

weitere Auflösungen: WIDTH_10BIT (0 .. 1023), WIDTH_11BIT (0 .. 2047), WIDTH_12BIT (0 .. 4095)

Eingangsspannungen über 3,6 V können den Microcontroller zerstören!

SPI-Bus (serial peripheral interface)

Bus-System von Motorola
synchron, seriell, Master-Slave-System

from machine import Pin, SPI

spi=SPI(baudrate=100000, polarity=1, phase=0, sck= Pin(0), mosi=Pin(2), miso=Pin(4))

Initialisierung des SPI-Busses mit einer Signal-Übertragungs-Rate von 100'000 Bd

sck .. serial clock (Bus-Takt)

MOSI (seltener auch SIMO oder SDO (serial data out)) .. Master Output + Slave Input

MISO (seltener auch SOMI oder SDI (serial data input)) .. Master Input + Slave Output

SDO und SDI werden i.A. aus der Sicht des Device's benannt, d.h. die Leitungen müssen sich kreuzen

die Polarität und die Phase können die Werte 0 oder 1 annehmen und steuern Datenabnahme; Phase bestimmt welcher Flankenwechsel ausgewertet wird; die Polarität, ob die steigende (0) oder fallende Flanke (1) das Signal ist

spi.init(baudrate=200000)

spi.read(AnzahlBytes)

spi.read(AnzahlBytes, Adresse)

Adresse (von MOSI) z.B. 0xff

puffer = bytearray(100)

spi.readinto(puffer)

spi.readinto(puffer, Adresse)

spi.write(b'abcdef')

Schreiben von 6 Bytes an MOSI

puffer = bytearray(5)

spi.write_readinto(b'12345', puffer)

Schreiben der Bytes an MOSI und Lesen von MISO in den Puffer

spi.write_readinto(puffer, puffer)

Schreiben des Puffers an MOSI und Lesen von MISO in den Puffer

SPI-Hardware-Bus

beim ESP-32 sind zwei Hardware-Kanäle steuerbar

Pin's sind festgelegt

HSPI (id=1) → sck = 14, mosi = 13, miso = 12
HSPI (id=2) → sck = 18, mosi = 23, miso = 19

from machine import Pin, SPI

hspl = SPI(1, 100000, sck= Pin(14), mosi=Pin(13), miso=Pin(12))
vspl = SPI(2, baudrate = 100000, polarity=0, phase=0, bits=8, firstbit=0, sck=Pin(18),
mosi=Pin(23), miso=Pin(19))

I2C-Bus

eigentlich I2C für inter-integrated circuit

serieller Daten-Bus von Philips

technisch identisch mit Two-Wire-Interface (von Amtel)

beides sind Zwei-Draht-Schnittstellen

praktisch 3 Leistungen: Betriebs-Spannung V_{DD} sowie zwei – über Pull-up-Widerständen
angeschlossene Arbeits-Leitungen (Takt (SDL) und Daten (SDA))

gearbeitet wird mit positiver Logik (LOW = 0 (max. 0,3 V_{DD}) und HIGH = 1 (min. 0,7 V_{DD}))

bei Daten-Übertragungen ist das 1. Byte die Slave-Adresse (die 7 niedrigen Bits bilden die
eigentliche Adresse, das 8. Bit bestimmt, ob Slave Daten empfangen (0 / LOW) bzw. senden
(1 / HIGH) soll

bestimmte Adressen sind für Sonderzwecke reserviert (insgesamt 112 Slave's ansprechbar)

from machine import Pin, I2C

i2c = I2C(scl = Pin(5), sla = Pin(4), freq = 100000)

i2c.readfrom(Adresse, AnzahlBytes)

Adresse z.B. 0x3a

i2c.writeto(Adresse, Wert)

puffer = bytearray(10)

i2c.writeto(Adresse, puffer)

OneWire-Treiber ()

from machine import Pin

import onewire

eindraht = onewire.OneWire(Pin(12))

aktiviert eine OneWire-Bus an der GPIO12

eindraht.scan()

Gibt eine Liste der gescannten Device's zurück

eindraht.reset()

Setzt den Bus zurück

eindraht.readbyte()

Ließt ein Byte vom Bus

eindraht.writebyte(Adresse)

Adresse könnte z.B. 0x12 sein

eindraht.write(Wert)

Wert wird als Bytes verstanden

eindraht.select_rom(b'12345678')

speziell für Temperatur-Sensoren DS18S20 und DS18B20

import time, ds18x20

ds = ds18x20.DS18x20(ow)

roms = ds.scan()

ds.convert_temp()

time.sleep_ms(1000)

für rom in roms:

print("gemessene Temperatur: ",ds.read_temp(rom))

LED-Leisten bzw. -Ringe (NeoPixel)

from machine import Pin

from neopixel import NeoPixel

pin = Pin(0, Pin.OUT)

neopix = NeoPixel(pin, AnzahlLEDs)

erstellt eine NeoPixel-LED-Reihe an der GPIO0

neopix[0] = (255,255,255)

neopix.write()

Setzt die erste LED auf weiß

die wirkliche Anzeige / Ausgabe erfolgt erst mit .write()

r,g,b = neopix[0]

liefert die Farbwerte der ersten LED zurück

für ESP ist eine weitere Low-Level-Ansteuerung möglich:

import esp

esp.neopixel_write(pin, rgb_buf, is800khz)

800 kHz ist die Default-Einstellung, praktisch sind auch 400 kHz möglich (timing=0)

Touch-Eingabe (capacitive touch)

from machine import TouchPad, Pin

touch = TouchPad(Pin(14))

Aktivieren des Touch-Modus für GPIO14 (Touch6)

touch.read()

Auslesen des Touch-Pin's (gelesener Wert wird bei Berührung (deutlich) kleiner)

Benutzen der Touch-Eingänge für das Aufwecken aus dem Tiefschlaf-Modus

```
import machine
```

```
from machine import TouchPad, Pin
```

```
import esp32
```

```
touch = TouchPad(Pin(14))
```

```
touch.config(500)
```

```
esp32.wake_on_touch(True)
```

```
machine.lightsleep()
```

ESP wird in den Tiefschlaf-Modus versetzt, solange der Touch-Sensor an GPIO14 (Touch6) gedrückt ist

DHT (Umweltsensoren, Temperatur-Luftfeuchte-Sensor)

häufig genutzter Kombinations-Sensor DHT11 für Luftfeuchtigkeit (Humidity) und Temperatur
Sensor arbeitet an allen Pin's

```
import dht
```

```
import machine
```

```
humtemp = dht.DHT11(machine.Pin(4))
```

Aktivieren des Sensors für GPIO4

```
humtemp.measure()
```

eine Messung abfragen

```
humtemp.temperature()
```

Gibt Temperatur in °C zurück

```
humtemp.humidity()
```

Gibt die relative Luftfeuchtigkeit in Prozent zurück

10.6.x.5.2. Modul "esp"

```
import esp
```

```
esp.osdebug(None)
```

Ausschalten der Debugging-Mitteilungen

```
esp.osdebug(0)
```

Umleiten der Debugging-Mitteilungen auf UART(0)

```
esp.flash_size()
```

esp.flash_user_start()

esp.flash_erase(SektorNummer)

esp.flash_write(ByteOffset, Puffer)

esp.flash_read(ByteOffset, Puffer)

10.6.x.5.3. Modul "esp32"

import esp32

esp32.hall_sensor()

Auslesen des (internen) Hall-Sensors

esp32.raw_temperature()

Auslesen des (CPU-internen) Temperatur-Sensors (Angabe in °F)

esp32.ULP()

Zugriff auf den ULP-Coprozessor (ultra low power; Stromspar-Coprozessor)

10.6.x.5.4. Modul "network"

import network

MeinWLAN = network.WLAN(network.STA_IF)

konfigurieren des (eigenen) WLAN-Moduls im Stations-Modus

MeinWLAN.active(True | False)

Ein- bzw. Aus-Schalten des WLAN-Moduls

MeinWLAN.scan()

Scannen des WLAN's nach AccessPoint's

MeinWLAN.isconnected()

Prüfen, ob Station mit dem AccessPoint verbunden ist

MeinWLAN.connect(SSID, Passwort)

Verbindung zum AccessPoint herstellen

MeinWLAN.config('mac')
gibt die MAC-Adresse zurück

MeinWLAN.ifconfig()
gibt die IP-Adresse, die Netzwerk-Maske, den Gateway und die DNS-Adresse zurück

MeinAccessPoint = network.WLAN(network.AP_IF)
konfigurieren des (eigenen) WLAN-Moduls als AccessPoint

MeinAccessPoint.config(essid=WLANName)
Festlegen des Namens für das WLAN

MeinAccessPoint.active(True | False)
Ein- bzw. Aus-Schalten des WLAN-Moduls

10.6.x.5.5. Modul "time"

import time

time.sleep(Sekunden)

time.sleep_ms(MilliSekunden)

time.sleep_us(MikroSekunden)

StartZeit = time.ticks_ms()
Laufzeit = time.ticks_diff(time.ticks_ms(), StartZeit)
Laufzeit ermitteln

time.sleep(Sekunden)

10.6.x.y. Sprach-Elemente vom MicroPython (Kurz-Übersicht / Spicker)

(formatierte) Ausgabe:

```
ausgabe:=wert | berechnung | "Text" | 'Text'
print()
print(ausgabe)
```

Verzweigung:

```
if bedingung:           # Einleitung und Test/Bedingung
    befehle              # Then-/Dann-/Wahr-Zweig (eingerückt!!! mehrzeilig
möglich)                #
{elif bedingung:        # zusätzliche(r) untergeordnete(r) Test/Bedingung
    befehle}            # untergeord. Then-/Dann-/Wahr-Zweig
[else:                  # optionaler Else-/Sonst-/Falsch/Rest-Zweig
    Befehle]
```

Schleifen:

```
while bedingung:        # while True:      # Endlosschleife
                        # (meist break notwendig)
    befehle
    {continue}          # Sprung zum nächsten Schleifendurchlauf /-anfang
    {befehle}
    break               # Sprung hinter Schleife (noch hinter ELSE)
[else:
    befehle}

for laufvariable in liste | tuple: # _ als laufvariable, wenn kein Gebrauch in
    befehle                      Schleife geplant
    [verzweigung : break]        # vorzeitiger Abbruch der Schleife

for laufvariable in range([untere_grenze, ] obere_grenze[, schrittweite]):
    befehle
```

erweiterter Spicker für das "normale" Python (→ [Python-Spicker](#))

10.7. Python auf und mit Taschenrechnern / spezieller Hardware

Zuersteinmal scheint der Einsatz einer Programmiersprache auf einem Taschenrechner nicht wirklich sinnvoll. Das Display ist sehr klein, die Tastatur sehr gewöhnungsbedürftig und die Leistungsfähigkeit ist auch beschränkt. Aber trotzdem gibt es Einsatz-Szenarien, die für einen programmierbaren Taschenrechner sprechen. Möglich Szenarien sind z.B.:

- Lösung komplexerer (sich mehr fach wiederholender) Aufgaben
- Darstellung mathematischer Funktionen und Zusammenhänge auf einem Gerät im Taschen-Format
- Erfassung und Auswertung von Mess-Werten
-

Die meisten programmierbaren Taschenrechner brachten lange Jahre eine BASIC-Variante mit. BASIC erfüllt aber nicht die Anforderungen an eine moderne strukturierte Programmiersprache. Python dagegen ist hier bestens geeignet. Erfahrungen, die man beim Programmieren auf dem Taschenrechner macht, kann man leicht auf die Programmierung größerer Computer usw. übertragen.

10.7.x. Casio-Rechner

FX-CG50

MicroPython 1.9.4, basiert auf Python 3.7.0
verfügbar auf Rechnern mit einer Betriebssystem-Version ab 03.40.0202

Auch in Bezug auf das offizielle MicroPython ist Python auf den Casio-Rechnern nochmals eingeschränkt. Für normale Programmier-Übungen und einer effektive(re)n Nutzung des Taschenrechner's spielt das aber kaum eine Rolle. Echte Programmierung sollte dann schon auf einem ordentlichen PC od.ä. erfolgen.

Eingabe-Möglichkeiten

- | | |
|-----------------------------|---|
| • Text-Eingabe | nach Einstellen von ALPHA am Rechner |
| • Listen-basiert | nach Drücken von [F4] (CHAR) kann aus einer Liste der verfügbaren Zeichen und Symbole mit [F3] (SYMBOL) ausgewählt werden |
| • Katalog-orientiert | Auswahl der Python-Befehle aus einem Katalog mittels [F6] (SHIFT 4 CAT) |

Syntax-Highlighting für die verschiedenen Element-Gruppen (Kommentare, Python-Befehle, Texte, Zahlen, ...)

Abspeichern mit FILE [F1] SAVE

Starten mit [F2] (RUN)

es wird dann automatisch in die Shell gewechselt und die Kommunikation erfolgt an dieser Stelle

Beispiel1:

Berechnung des n-ten Gliedes der FIBONACCHI-Folge nach der Näherungs-Formel von MOIVRE-BINET:

$$a_n = \frac{1}{\sqrt{5}} \cdot \left(\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right)$$

```
1 n=int(input("n="))
2 z=1/5**0.5*((1\
3 +5**0.5)/2)**n-\
4 ((1-5**0.5)/2)**n)
5 print('%d'%(z))
```

Q: LUDWICKI, Wolfgang: Programmieren mit Python mit dem dem FX-CG50.-IN: CASIO forum 2/2020, S. 9

der gespiegelte Schrägstrich ("****"; Backslash) kennzeichnet nur den Umbruch der Eingabezeile. In anderen Systemen kann der Text ohne diese Zeichen hintereinanderweg eingegeben werden.

Beispiel2:

iteratives Berechnen des n-ten Gliedes

```
1 n=int(input("n="))
2 def fibi(n):
3     a,b=1,1
4     for i in range(n-2):
5         a,b=b,a+b
6     return b
7 print('%d'%(fibi(n)))
```

Q: LUDWICKI, Wolfgang: Programmieren mit Python mit dem dem FX-CG50.-IN: CASIO forum 2/2020, S. 9

Beispiel3:

rekursives Berechnen des n-ten Gliedes

```
1 n=int(input("n="))
2 def fibr(n):
3     if n==1 or n==2:
4         return 1
5     else:
6         return fibr(n-2)\
7             +fibr(n-1)
8 print('%d'%(fibr(n)))
```

Q: LUDWICKI, Wolfgang: Programmieren mit Python mit dem dem FX-CG50.-IN: CASIO forum 2/2020, S. 9

es stehen auch erweiternde Bibliotheken in der Material-Datenbank bereit (→ www.casio-schulrechner.de)

z.B. turtle.py und matplotlib.py

10.7.x. Texas Instruments-Rechner

verfügbar z.B. auf:

- TI84 Plus CE-T Python Edition
- TI-Nspire CX II-T CAS

Eingabe-Möglichkeiten

-
-
-

TI-Nspire CXII-T CAS

MicroPython 1.11.0, basiert auf Python 3.4.0

Objekt-Orientierung

bei der Eingabe werden Operanden rot angezeigt
syntaktische Schlüsselwörter werden blau angezeigt

Kombination der Eingabe von Tastatur, aus dem Nspire-Menü ("Werkzeug"-Schaltfläche/Menü) und von der Taschenrechner-Simulation in N-spire möglich

Module:

math
time
random
tiplotlib
ti_hub
ti_rover
ti-draw
cx_turtle2
cmath

starten eines Programm's mit [ctrl] [R]

Schleifen mit Abbruch durch die ESC-Taste

```
while get_key!="esc":
```

lassen sich immer über die erste Zeile im Menü-System in der Nspire-Software beim Hinzufügen von Funktion einbauen

Löschen des aktuellen Anzeige-Fensters über "Extra's"

```
sorted(Liste)
```

```
localtime()
```

liefert Datum, Zeit, Wochentag, Tag im Jahr, Sommerzeit

Formeln programmieren

Im Hauptmenü "A" auswählen

Shell zum einfachen Arbeiten und Ablaufen lassen der Programme
Taschenrechner-Funktionen (für TR natürlich nicht wirklich sinnvoll)

Menü-System für alle Funktionen

niemand muss Befehle lernen, nur noch raussuchen
ev. die Menü-Punkte durchgehen

Shell

Alt-Ctrl-

```
while get_key()!="esc":
```

gut als umgebende Schleife für komplexe Programme, um eine Abbruch-Möglichkeit zu haben

```
store_list(speichername,datenliste)
```

in der Tabellenkalkulation nutzbar

im Spalten-Kopf kann dann die Verknüpfung mit dem speichername herstellen
dann stehen die Daten in der Tabellenkalkulation bereit

bei Neu-Erstellen von Dateien gibt es Vorlagen mit vordefinierten Bibliotheken

```
import ti_rover as rv
```

```
rv.motors("ccw",255,"cw",150)
```

```
sleep(2)
```

```
rv.stop()
```

cw ... mit Uhrzeitsinn

ccw ... entgegen Uhrzeigersinn

Motoren funktionieren entgegengesetzt!

Rover-Zentrum ist der Stifthalter

für Motor-Befehle wird normale Programm-Abarbeitung NICHT unterbrochen

für Manöver müssen die Manöverzeiten als Schlafzeit für Hauptprogramm eingeplant werden

mit geladenem Stift wird die programmierte Figur aufgezichnet

Nutzung des TI-Innovator

Steuern des TI-Rover

TI-84 Plus

Version "CE-T Python Edition"

ist graphischer Taschenrechner (GTR)

Nutzung des micro::bit

spezielle Bibliothek zur Nutzung der micro-bit-Ressourcen

mittlerweile gibt es diverse Python-Editoren / -Systeme für den micro::bit
da macht die Programmierung mit dem Taschenrechner als Editor nicht so viel Sinn, es sei denn, man will Daten austauschen oder spezielle Funktionen des Taschenrechners ausnutzen

ein anderes Szenario ist die Verwendung von micro::bit's, ohne dass PCs oder ähnliches zur Verfügung stehen

praktisch ein Minimal-System (vorausgesetzt entsprechende Taschenrechner stehen standardmäßig zur Verfügung)

Programm-Übertragung mittels mini-zu-micro-USB-Kabel (mini-A auf micro-B)

sehr viele Funktionen (in kleinen Extra-Modulen)

micro-bit muss vor dem ersten Benutzen mit den TI-Taschenrechnern geflasht (ti-runtime) werden

dazu einfach die ti-runtime auf den micro:bit (als Laufwerk im Explorer abgezeigt) ziehen

notwendige Dateien unter:

<https://education.ti.com/de/alles-fuer-die-schule/microbit>

```
from microbit import *
```

Vorbereitung des Taschenrechners

mit Hilfe der "TI Connect"-Software muss zuerst die MICROBIT.8xv-Datei auf dem TI-84 Plus geladen werden

Für einzelne zusätzliche Hardware-Komponenten müssen noch weitere Module geladen werden. Sie enthalten jeweils die Ansteuerung für die Zusatz-Geräte.

Zusatz-Hardware	TI-Modul	Bemerkungen
interne Sensoren	MB_SENSR.8xv	
interne 5x5-LED-Matrix	MB_DISP.8xv	
interne Button A+ B	MB_BUTNS.8xv	
interner Funk	MB_RADIO.8xv	
interne Pin's	MB_PINS.8xv	
Grove-Sensoren	MB_GROVE.8xv	
Neopixel	MB_NEOPX.8xv	
Sound	MB_MUSIC.8xv	

Als nächstes muss der micro::bit mit einer neuen Software ausgestattet werden. Die Datei heißt ti_ce_runtime.hex und wird nach dem Download direkt auf das – vom micro::bit erzeugte – Laufwerk kopiert

Nach der Installation der neuen Firmware (für Python) ist der micro::bit auf diese Aufgabe eingeschränkt. Soll wieder die "normale" Firmware verwendet werden, dann muss diese auf den micro::bit übertragen werden (wie oben die ti_ce_runtime.hex). Die notwendige hex-Datei wird von microbit.org/code/ bereitgestellt.

Nun kann der micro::bit mit einem Mini-zu-Micro-USB-Kabel verbunden werden.
Mit Hilfe der Datei NPTEST.8xv kann und sollte nun ein Test der Funktionsfähigkeit erfolgen.

Links:

<https://education.ti.com/de/alles-fuer-die-schule/microbit>

<https://python.microbit.org/v/2.0> (online-Python-Editor)

<https://python.microbit.org/v/2> (online-Python-Editor (neueste Version))

<https://archive.microbit.org/de/> (weitere Materialien)

Downloads:

<https://ti-unterrichtsmaterialien.net/materialien?country=1&langauge=2&q=micro%3Abit> (→ MICRO-BIT.8xv; MB_SENSR.8xv; ...)

https://ti-unterrichtsmaterialien.net/fileadmin/DE-Materialien/Materialien/TI_Runtime_2.6.hex

(→ ti_ce_runtime.hex)

(→)

10.7.x. Miniroboter Edison (Microbric)



Set besteht aus 1x Edison und 1x Verbindungs-Kabel
mit zusätzl.-Set und / oder LEGO® erweiterbar / ausbaufähig

extrem robust
etwas filigrane Abdeckung des Batterie-Fach's

vorgesehen für AAA-Batterien

Verbindungs-Kabel

→ startedison.com

verschiedene Programmier-Möglichkeiten

- **Barcodes** durch das Scannen (Überfahren) von Barcode's auf der Fahrbahn werden voreingestellte aktiviert
kein Computer etc. zum Lernen notwendig
gedacht ab Alter von 4 Jahren
- **EdBlocks** graphische Programmierung mit Blöcken
Blöcke sind durch Symbol-Bilder charakterisiert (nur noch Eingabe von Parametern notwendig)
gedacht ab Alter von 7 Jahren (Grundschule 2. Klasse)
→ <http://stemgoals.co.uk/>
- **EdScratch** Scratch-basierte Block-Programmierung
gedacht ab Alter von 10 Jahren (Grundschule 4. Klasse / Orientierungstufe)
- **EdPy** Text-basierte Programmierung mit Python
online-Nutzung: www.edpyapp.com
gedacht ab Alter von 13 Jahren (Sekundarstufe I)

Links:

www.meetedison.com

startedison.com

<https://meetedison.com/robot-programming-software/edpy/> (u.a. Video-Tutorial's)

10.8. Python und Data Science

Datenbank-Begriffe im Data science
Datensätze sind Fälle
Attribute / Felder sind Merkmale bzw. Variablen

Öffnen einer Datenbank in den Speicher

```
with open(dateiname, 'rb') as datenbestand:  
    print(dateiname + " hat den Inhalt: "+ datenbestand.read())
```

Öffnen einer Datenbank als Stream

```
with open(dateiname, 'rb') as datenstrom:  
    for auswahl in datenstrom:  
        print("gelesene Daten: " + auswahl)
```

Streamen mit Auswahl einzelner Datensätze (Fälle)

```
bedingung=???  
with open(dateiname, 'rb') as datenstrom:  
    for j, auswahl in enumerate(datenstrom):  
        if j == bedingung:  
            print("gefundene Daten: "+str(j)+" ---> "+auswahl)
```

zufällige Auswahl aus einem Stream

```
from random import random  
beispielwert=0.3333  
with open(dateiname, 'rb') as datenstrom:  
    for j, auswahl in enumerate(datenstrom):  
        if random()<=beispielwert:  
            print("gefundene Daten: "+str(j)+" ---> "+auswahl)
```

Flatfile ist Textdatei (übliche Separator-getrennte Daten-Elemente in einem Datensatz)
Klassische Struktur einer CSV-Datei
In der ersten Zeile sind die Felder definiert
Datensätze (Fälle) sind durch Zeilenumbruch getrennt / beendet
Attribute (Felder, Merkmale, Variablen) sind durch Kommata getrennt
Zeichenketten werden durch Anführungszeichen umschlossen
Integer-Zahlen ohne Anführungszeichen
Reelle Zahlen ebenfalls ohne Anführungszeichen und ein Punkt als Dezimal-Trenner

CSV-Datei über Pandas einlesen:

```
import pandas as pds  
inhalt=pds.io.parsers.read_csv(dateiname)  
wert = inhalt[[attribut]]  
print(wert)
```

EXCEL-Datei mit Pandas einlesen:

```
import pandas as pds
kalk=pds.ExcelFile(dateiname)
ausgeleseneWerte = kalk.parse("Tabelle1", indexZeile=None, na:values=[NA])
print(ausgeleseneWerte)
```

Laden / Öffnen von Dateien mit unstrukturierten Daten

```
from skimage.io import imread
from skimage.transform import resize
from matplotlib import pyplot als plt
import matplotlib.cm as cm

unstrukDaten =
("http://upload.wikimedia.org/"+"wikimedia/commons/7/7d/Dog_face.png")
image = imread(unstrukDaten, as_grey=True)
plt.imshow(image, cmap=cm.grey)
plt.show()
```

Resizen u.ä. möglich (→ Data Science mit Python für DUMMIES, S.116ff)

der Titanic-Daten-Bestand

Daten zu den Passagieren der Titanic

zu installierende Bibliotheken

```
pip install statsmodels
pip install xlrd
pip install openpyxl
```

eigentlichen Programm bzw. interaktives Arbeiten mit den Daten

(nach Q: <https://deeptime.com/@leonard-puttmann-a8ef/Titanic-Dataset-544c6818-f79a-4068-bbc1-d6fdf42d2998>)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm

from statsmodels.formula.api import ols
from scipy.stats import t
from sklearn.linear_model import LogisticRegression
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
```

```

# Laden des Daten-Bestands
titanic = pd.read_excel('3_Titanic.xlsx')

# Analyse der Tabelle über die obersten 10 Zeilen
titanic.head(10)

# Anzeige der Datentypen
titanic.info()

# Analyse mit deskriptiver Statistik
titanic.describe()

#Visualisierung einzelner Attribute
EDA_cols = ['Age', 'Pclass']
EDA_data = titanic[EDA_cols]

plt.figure(figsize = (10,10))
plt.style.use('ggplot')
sns.boxplot(x = 'Pclass', y = 'Age', data = EDA_data, palette = 'YlGnBu')
plt.ylabel('Alter')
plt.xlabel('Klasse')
plt.show()

#Visualisierung Überlebende pro Klasse
EDA_cols2 = ['Survived', 'Pclass']
EDA_data2 = titanic[EDA_cols2]

plt.figure(figsize = (10,10))
plt.style.use('ggplot')
sns.boxplot(x = 'Pclass', y = 'Survived', data = EDA_data2, palette = 'YlGnBu')
plt.ylabel('Anteil Überlebende')
plt.xlabel('Klasse')
plt.show()

# Analyse nach Geschlecht – Verteilung Passagiere
plt.figure(figsize = (10,10))
plt.style.use('ggplot')
sns.boxplot(x = 'Sex', y = titanic.idex, data = titanic, palette = 'tab10', alpha =0.8)
plt.ylabel('Anzahl Passagiere')
plt.xlabel('Geschlecht')
plt.show()

# Analyse nach Geschlecht – Verteilung Überlebende
plt.figure(figsize = (10,10))
plt.style.use('ggplot')
sns.boxplot(x = 'Sex', y = 'Survived', data = titanic, palette = 'tab10', alpha =0.8)
plt.ylabel('Überlebende')
plt.xlabel('Geschlecht')
plt.show()

# kategorische Daten für die weitere Bearbeitung encoden und Voranzeige
titanic['Sex'] = pd.get_dummies(titanic['Sex'])
titanic['Embarked'] = pd.get_dummies(titanic['Embarked'])

print(titanic.head(10))

# Anzeige der Überlebenden (Survived=1)

```

```

titanic[\"Survived\"].value_counts()

# Testen von Hypothesen
print(\"Überlebens-Wahrscheinlichkeit abhängig von mitgereisten Familien-Angehörigen\")
titanic[\"Parch\"].value_counts()
parch_survivors = titanic.query('Survived >= 1 & Parch >= 1')
parch_survivors[\"Survived\"].value_counts()

print(\"Überlebens-Wahrscheinlichkeit abhängig vom Geschlecht\")
fem_parch_survivors = titanic.query('Survived >= 1 & Parch >= 1 & Sex == 1')
fem_parch_survivors[\"Survived\"].value_counts()

print(\"Hypothesen-Test: Überlebens-Chance abhängig von mitgereisten Familien-
Angehörigen und Geschlecht\")
hypothese = pd.DataFrame(columns = ['überlebt', 'mit Familie', 'Familie überlebt', 'Fam. über-
lebt + weibl.'], dtype = float)
werte = {'überlebt':342, 'mit Familie':213, 'Familie überlebt':109, 'Fam. überlebt + weibl.':80}
hypothese = hypothese.append(werte, ignore_index=True)

plt.figure(figsize(10, 10))
plt.style.use('ggplot')
sns.barplot(data = hypothese, palette='YlGnBu')
plt.xticks(rotation=10)
plt.show()

# ANOVA-Analyse
anova_model = ols('Survived ~ Parch', data = titanic).fit()
anova_ergebnisse = sm.stats.anova_lm(anova_model, typ = 2)
print(anova_ergebnisse[\"PR(>F)\"])

# Analyse Korrelationen zwischen Attributen
plt.figure(figsize(10, 10))
plt.style.use('ggplot')
corr_matrix = titanic.corr()
sns.heatmap(corr_matrix, annot = True, linewidths = 1)
plt.show()

print(\"Durchschnitt Alter: \", titanic.Age.mean())
print(\"Median Alter: \", titanic.Age.median())

# Anpassung der Daten fürs MachineLearning
X = titanic.loc[:, feauture_cols]
print(X.shape)

y = titanic.Survived
print(y.shape)

# logistische Regression
logistic_reg = LogisticRegression()

#Training
logistic_reg.fit(X, y)
prd_lr = logistic_reg.predict(X)
print(\"Genauigkeit: \",accuracy_score(y, prd_lr))

# Teilen Trainings- und Test-Daten-Teil
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)

```

```

# Pipeline initialisieren
logreg = LogisticRegression()
pipe = Pipeline(steps = [('logistic_Reg', logreg)])

werte_raster = {'logistic_Reg__C' : np.logspace(-4, 4, 5), 'logistic_Reg__penalty' : ['l1', 'l2',
'none']}

# Modell-Training
model_lr = GridSearchCV(pipe, param_grid = werte_raster, cv = 5, verbose = True)

# Anzeige Leistung
model_lr.fit(X_train, y_train)
pred = model_lr(X_test)
print("Genauigkeit: ",accuracy_score(y_test, pred))

print("beste Ergebnisse: ", model_lr.best_params_)

# Random Forest
schwarzwald = RandomForestClassifier(n_estimators = 100)

schwarzwald.fit(X_train, y_train)

y_pred = schwarzwald.predict(X_test)
acc_random_forest = schwarzwald.score(X_train, y_train)
print("Genauigkeit: ",acc_random_forest)

# Nutzung des Modells für fiktive Daten
fikt_pers = pd.DataFrame(columns = ['Pclass', 'Age', 'Parch', 'Sex', 'SibSp', 'Embarked'],
dtype=float)
fikt_pers.head()

rose = {'Pclass': 1, 'Age': 17, 'Parch': 0, 'Sex': 1, 'SibSp': 1, 'Embarked': 0}
jack = {'Pclass': 3, 'Age': 20, 'Parch': 0, 'Sex': 0, 'SibSp': 0, 'Embarked': 0}

fikt_pers = fikt_per.append(rose, ignore_index = True)
fikt_pers = fikt_per.append(jack, ignore_index = True)

# Vorhersagen für fiktive Personen
wahrscheinlichkeit_fiktpers = schwarzwald.predict_proba(fikt_pers)
println("Überlebens- und Vorhersage-Wahrscheinlichkeit")
println(wahrscheinlichkeit_fiktpers)

```

Wichtigkeit der einzelnen Attribute

```

importances = schwarzwald.feature_importances_
feature_names = ['Pclass', 'Age', 'Parch', 'Sex', 'SibSp', 'Embarked']
forest_importances = pd.Series(importances, index = feature_names)
std = np.std([tree.feature_importances_ for tree in schwarzwald.estimators_], axis = 0)

fig, ax = plt.subplots()
forest_importances.plot.bar(yerr = std, ax = ax)
ax.set_title("Wichtigkeit der Attribute")
ax.set_ylabel("%")
fig.tight_layout()

```

plt.show

10.9. Python und Künstliche Intelligenz

direkt machbar mit diversen zusätzlichen Bibliotheken, die meist extra zu installieren sind

```
pip install numpy  
pip install pandas  
pip install scikit-learn
```

besonders sinnvoll im Zusammenhang mit Jupyter-Notebook's, da so die einzelnen Schritte interaktiv eingegbar und abarbeitbar gemacht werden

online-Jupyter
<https://jupyter.org/try>

offline Installation am Besten über eine aktuelle Anaconda-Installation

Integration in die klassische Python-Umgebung

```
pip install jupyter  
pip install matplotlib  
jupyter notebook
```

der letzte Befehl ist die übliche Start-Sequenz
als Anzeige wird der Standard-Browser genutzt (→ <http://localhost:8888/tree>)
Wechsel zwischen Eingabe-Bereich und einem Ausgabe-Bereich

10.9.x. Entscheidungs-Bäume

praktische Nutzung eines Entscheidungs-Baum's über (geschachtelte) Verzweigungs-Strukturen

10.9.x. Korrelation und Regression

10.9.x. maschinelles Lernen

10.10. Python kommuniziert in Discord

10.10.0. Allgemeines und Vorbereitung

Discord ist ein weit verbreitetes und besonders bei Gamern beliebtes Kommunikations- Programm.

Spiele können diskutiert werden, es kann während des Spiel's gechattet werden

Steuerung und Programmierung von Discord ist über eine spezielle Schnittstelle möglich. Für diese existiert auch ein Python-Modul: discord.py

Mit Python ist eine Programmierung diverser Funktionen von dsicord möglich.

Installation der Libary in der Konsole:

```
python3 -m pip install -U discord.py[voice] --user
```

(ev. auch nur: pip install -U discord.py)

Registrierung der eigenen App auf der Webseite von discord → discordapp.com notwendig unter "Application" -- "New Application" vergibt man der eigenen App einen Namen, z.B. meineApp

im Menü Bot erstellt man dann einen Bot-User über "Add Bot"

der Bot sollte üblicherweise mit "Public Bot" aktiviert werden, sonst können andere Nutzer den Bot nicht einladen

Option "Requires OAuth2 Code Grant" sollt nicht gesetzt werden

Die Einladung des Bot's (meineApp) erfolgt im Hauptmenü des eigenen Server's unter O-Auth2

hier wird die Option "Scopes > Bot" gesetzt

es wird ein Link generiert, der dann im Browser eingegeben werden kann

damit wird der Bot angemeldet

eine Spieler-Gilde wird auch benötigt

in Dicord oder auf der Webseite von discord kann dann eine Instanz erstellt werden dazu auf das Plus-Symbol klicken

weiterführende Links:

https://praxistipps.chip.de/discord-bot-erstellen-eine-anleitung_118538 (allg Hinweise; Einrichtung)

10.10.2. erste Kommunikations-Versuche

Besonderheiten der asynchronen Kommunikation

```
# bot.py
import os

import discord
from dotenv import load_dotenv

load_dotenv()
TOKEN = os.getenv('DISCORD_TOKEN')

client = discord.Client()

@client.event
async def on_ready():
    print(f'{client.user} has connected to Discord!')

client.run(TOKEN)
Q: https://realpython.com/how-to-make-a-discord-bot-python/
```

client ist eine Instanz / ein Objekt der Klasse Client

on_ready() ist der Event-Handler für die bestehenden Kommunikations- und Bedienungs-Möglichkeiten

Sollen bestimmte Informationen z.B. der eigene Token nicht im Programm-Text ersche, können solche Informationen in einer .envim-Datei gespeichert werden. Diese muss sich im gleichen Ordner, wie der Quell-Text befinden

```
# .env
DISCORD_TOKEN={your-bot-token}
```

ev. muss zusätzlich noch die Bibliothek dotenv installiert werden

pip install -U python-dotenv

die Methode client.run() führt dann das Programm aus

für weitere Versuche muss dann auch der Gilde-Token mit angegeben werden
z.B. in der envim-Datei:

```
# .env
DISCORD_TOKEN={your-bot-token}
DISCORD_GUILD={your-guild-name}
```

ansonsten geht auch direkt im Quell-Text:

```
# bot.py
import os

import discord
from dotenv import load_dotenv

load_dotenv()
TOKEN = os.getenv('DISCORD_TOKEN')
GUILD = os.getenv('DISCORD_GUILD')
```



```

client = discord.Client()

@client.event
async def on_ready():
    for guild in client.guilds:
        if guild.name == GUILD:
            break

    print(
        f'{client.user} is connected to the following guild:\n'
        f'{guild.name} (id: {guild.id}) '
    )

client.run(TOKEN)
Q: https://realpython.com/how-to-make-a-discord-bot-python/

```

```

# bot.py
import os

import discord
from dotenv import load_dotenv

load_dotenv()
TOKEN = os.getenv('DISCORD_TOKEN')
GUILD = os.getenv('DISCORD_GUILD')

client = discord.Client()

@client.event
async def on_ready():
    for guild in client.guilds:
        if guild.name == GUILD:
            break

    print(
        f'{client.user} is connected to the following guild:\n'
        f'{guild.name} (id: {guild.id})\n'
    )

    members = '\n - '.join([member.name for member in guild.members])
    print(f'Guild Members:\n - {members}')

client.run(TOKEN)
Q: https://realpython.com/how-to-make-a-discord-bot-python/

```

```

# bot.py
import os

import discord
from dotenv import load_dotenv

load_dotenv()
TOKEN = os.getenv('DISCORD_TOKEN')
GUILD = os.getenv('DISCORD_GUILD')

client = discord.Client()

@client.event
async def on_ready():
    for guild in client.guilds:
        if guild.name == GUILD:
            break

```

```

print(
    f'{client.user} is connected to the following guild:\n'
    f'{guild.name}(id: {guild.id}) '
)

```

client.run(TOKEN)

Q: <https://realpython.com/how-to-make-a-discord-bot-python/>

```

# bot.py
import os

import discord
from dotenv import load_dotenv

load_dotenv()
TOKEN = os.getenv('DISCORD_TOKEN')
GUILD = os.getenv('DISCORD_GUILD')

client = discord.Client()

@client.event
async def on_ready():
    guild = discord.utils.find(lambda g: g.name == GUILD, client.guilds)
    print(
        f'{client.user} is connected to the following guild:\n'
        f'{guild.name}(id: {guild.id}) '
    )

```

client.run(TOKEN)

Q: <https://realpython.com/how-to-make-a-discord-bot-python/>

```

# bot.py
import os

import discord
from dotenv import load_dotenv

load_dotenv()
TOKEN = os.getenv('DISCORD_TOKEN')
GUILD = os.getenv('DISCORD_GUILD')

client = discord.Client()

@client.event
async def on_ready():
    guild = discord.utils.get(client.guilds, name=GUILD)
    print(
        f'{client.user} is connected to the following guild:\n'
        f'{guild.name}(id: {guild.id}) '
    )

```

client.run(TOKEN)

Q: <https://realpython.com/how-to-make-a-discord-bot-python/>

```

# bot.py
import os

import discord
from dotenv import load_dotenv

load_dotenv()

```

```
TOKEN = os.getenv('DISCORD_TOKEN')

class CustomClient(discord.Client):
    async def on_ready(self):
        print(f'{self.user} has connected to Discord!')

client = CustomClient()
client.run(TOKEN)
Q: https://realpython.com/how-to-make-a-discord-bot-python/
```

Begrüßung neuer Mitglieder

```
# bot.py
import os

import discord
from dotenv import load_dotenv

load_dotenv()
TOKEN = os.getenv('DISCORD_TOKEN')

client = discord.Client()

@client.event
async def on_ready():
    print(f'{client.user.name} has connected to Discord!')

@client.event
async def on_member_join(member):
    await member.create_dm()
    await member.dm_channel.send(
        f'Hi {member.name}, welcome to my Discord server!'
    )

client.run(TOKEN)
Q: https://realpython.com/how-to-make-a-discord-bot-python/
```

auf Nachrichten antworten

```
@client.event
async def on_message(message):
    if message.author == client.user:
        return

    brooklyn_99_quotes = [
        'I\'m the human form of the 🍆 emoji.',
        'Bingpot!',
        (
            'Cool. Cool cool cool cool cool cool cool, '
            'no doubt no doubt no doubt no doubt.'
        ),
    ]

    if message.content == '99!':
        response = random.choice(brooklyn_99_quotes)
        await message.channel.send(response)
Q: https://realpython.com/how-to-make-a-discord-bot-python/
```

Geburtstags-Glückwünsche

```
@client.event
async def on_message(message):
    if 'happy birthday' in message.content.lower():
        await message.channel.send('Happy Birthday! 🎂 🎉')
Q: https://realpython.com/how-to-make-a-discord-bot-python/
```

```
# bot.py
import os
import random

import discord
from dotenv import load_dotenv

load_dotenv()
TOKEN = os.getenv('DISCORD_TOKEN')

client = discord.Client()

@client.event
async def on_ready():
    print(f'{client.user.name} has connected to Discord!')

@client.event
async def on_member_join(member):
    await member.create_dm()
    await member.dm_channel.send(
        f'Hi {member.name}, welcome to my Discord server!'
    )

@client.event
async def on_message(message):
    if message.author == client.user:
        return

    brooklyn_99_quotes = [
        'I\'m the human form of the 💯 emoji.',
        'Bingpot!',
        (
            'Cool. Cool cool cool cool cool cool cool, '
            'no doubt no doubt no doubt no doubt.'
        ),
    ]

    if message.content == '99!':
        response = random.choice(brooklyn_99_quotes)
        await message.channel.send(response)

client.run(TOKEN)
Q: https://realpython.com/how-to-make-a-discord-bot-python/
```

Ausnahme-Behandlung

```
# bot.py
import os
import random

import discord
from dotenv import load_dotenv

load_dotenv()
TOKEN = os.getenv('DISCORD_TOKEN')
```

```

client = discord.Client()

@client.event
async def on_ready():
    print(f'{client.user.name} has connected to Discord!')

@client.event
async def on_member_join(member):
    await member.create_dm()
    await member.dm_channel.send(
        f'Hi {member.name}, welcome to my Discord server!'
    )

@client.event
async def on_message(message):
    if message.author == client.user:
        return

    brooklyn_99_quotes = [
        'I\'m the human form of the 🐍 emoji.',
        'Bingpot!',
        (
            'Cool. Cool cool cool cool cool cool cool, '
            'no doubt no doubt no doubt no doubt.'
        ),
    ]

    if message.content == '99!':
        response = random.choice(brooklyn_99_quotes)
        await message.channel.send(response)
    elif message.content == 'raise-exception':
        raise discord.DiscordException

client.run(TOKEN)

```

Q: <https://realpython.com/how-to-make-a-discord-bot-python/>

```

@client.event
async def on_error(event, *args, **kwargs):
    with open('err.log', 'a') as f:
        if event == 'on_message':
            f.write(f'Unhandled message: {args[0]}\n')
        else:
            raise

```

Q: <https://realpython.com/how-to-make-a-discord-bot-python/>

10.10.3. Programmierung eines Bot's

Bot's sind automatisierte Programme, die bestimmte Aufgaben erfüllen in discord kann das z.B.:

- Begrüßung neuer Nutzer / Mitspieler im Team / in der Gilde
-

sein

extra Ordner anlegen

über einen beliebigen Editor die Datei bot.py in diesem Ordner anlegen

der Quell-Text lautet:

```
import discord from discord.ext import commands TOKEN = 'Token hineinkopie-
ren' description = '''ninjaBot in Python''' bot = com-
mands.Bot(command_prefix='?', description=description) @bot.event async def
on_ready(): print('Logged in as') print(bot.user.name) print(bot.user.id)
print('-----') @bot.command() async def hello(ctx): """Says world""" await
ctx.send("world") @bot.command() async def add(ctx, left : int, right :
int): """Adds two numbers together.""" await ctx.send(left + right)
bot.run(TOKEN)
```

Q: https://praxistipps.chip.de/discord-bot-erstellen-eine-anleitung_118538

der gespeicherte Quell-Text kann dann ausgeführt werden:

python3 bot.py

Ausgaben erscheinen im discord-Programm in der lokalen Konsole

in Fortsetzung des obigen Quell-Textes

Bot verbinden

```
# bot.py
import os
import random
from dotenv import load_dotenv

# 1
from discord.ext import commands

load_dotenv()
TOKEN = os.getenv('DISCORD_TOKEN')

# 2
bot = commands.Bot(command_prefix='!')

@bot.event
async def on_ready():
    print(f'{bot.user.name} has connected to Discord!')

bot.run(TOKEN)
```

Q: <https://realpython.com/how-to-make-a-discord-bot-python/>

```
# bot.py
import os
import random

import discord
from dotenv import load_dotenv

load_dotenv()
TOKEN = os.getenv('DISCORD_TOKEN')

client = discord.Client()
```

```

@client.event
async def on_message(message):
    if message.author == client.user:
        return

    brooklyn_99_quotes = [
        'I\'m the human form of the 🍆 emoji.',
        'Bingpot!',
        (
            'Cool. Cool cool cool cool cool cool cool, '
            'no doubt no doubt no doubt no doubt.'
        ),
    ]

    if message.content == '99!':
        response = random.choice(brooklyn_99_quotes)
        await message.channel.send(response)

client.run(TOKEN)
Q: https://realpython.com/how-to-make-a-discord-bot-python/

```

```

# bot.py
import os
import random

from discord.ext import commands
from dotenv import load_dotenv

load_dotenv()
TOKEN = os.getenv('DISCORD_TOKEN')

bot = commands.Bot(command_prefix='!')

@bot.command(name='99')
async def nine_nine(ctx):
    brooklyn_99_quotes = [
        'I\'m the human form of the 🍆 emoji.',
        'Bingpot!',
        (
            'Cool. Cool cool cool cool cool cool cool, '
            'no doubt no doubt no doubt no doubt.'
        ),
    ]

    response = random.choice(brooklyn_99_quotes)
    await ctx.send(response)

bot.run(TOKEN)
Q: https://realpython.com/how-to-make-a-discord-bot-python/

```

```

# bot.py
import os
import random

from discord.ext import commands
from dotenv import load_dotenv

load_dotenv()
TOKEN = os.getenv('DISCORD_TOKEN')

bot = commands.Bot(command_prefix='!')

```

```
@bot.command(name='99', help='Responds with a random quote from Brooklyn 99')
async def nine_nine(ctx):
    brooklyn_99_quotes = [
        'I\'m the human form of the 🍑 emoji.',
        'Bingpot!',
        (
            'Cool. Cool cool cool cool cool cool cool, '
            'no doubt no doubt no doubt no doubt.'
        ),
    ]

    response = random.choice(brooklyn_99_quotes)
    await ctx.send(response)
```

bot.run(TOKEN)

Q: <https://realpython.com/how-to-make-a-discord-bot-python/>

Parameter konvertieren

```
@bot.command(name='roll_dice', help='Simulates rolling dice.')
async def roll(ctx, number_of_dice, number_of_sides):
    dice = [
        str(random.choice(range(1, number_of_sides + 1)))
        for _ in range(number_of_dice)
    ]
    await ctx.send(', '.join(dice))
```

Q: <https://realpython.com/how-to-make-a-discord-bot-python/>

Befehls-Prädikate prüfen

```
if message.author == client.user:
    return
```

```
# bot.py
import os

import discord
from discord.ext import commands
from dotenv import load_dotenv

load_dotenv()
TOKEN = os.getenv('DISCORD_TOKEN')

bot = commands.Bot(command_prefix='!')

@bot.command(name='create-channel')
@commands.has_role('admin')
async def create_channel(ctx, channel_name='real-python'):
    guild = ctx.guild
    existing_channel = discord.utils.get(guild.channels, name=channel_name)
    if not existing_channel:
        print(f'Creating a new channel: {channel_name}')
        await guild.create_text_channel(channel_name)

bot.run(TOKEN)
```

Q: <https://realpython.com/how-to-make-a-discord-bot-python/>

11. Üben, üben und nochmals üben

hier folgen Aufgaben unterschiedlichster Schwierigkeitsgrade und Komplexitäten
keine Abfolge, wie im Skript
einfache Sammlung verschiedener – in diversen Quellen gefundener – Aufgabenstellungen
oder (informatischer) Probleme

nicht täuschen lassen, Aufgaben, die leicht oder einfach zu lösen scheinen, können sich als echte Programmier-Diamanten herausstellen. Dagegen können Aufgaben mit seitenlangen Aufgabenstellungen mit ein paar Zeilen Quelltext erschlagen werden. Man erinnere sich an die Wundertüte Rekursion (→ [8.4.2. Rekursion](#))

im Allgemeinen hilft nur probieren
eine Lösung die ich oder ein anderer als leicht einschätze, kann für jemanden Anderes ein unlösbares Problem sein, aber es geht natürlich auch anders herum. So manche Aufgabe, für die ich viele Zeilen Quelltext brauche erledigt ein findiger Programmierer mit genial wenigen Zeilen. So ist die Welt, und das ist gut so!

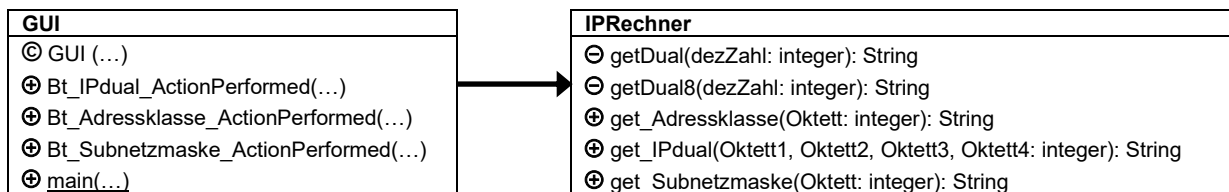
nicht unendlich in eine Aufgabe reinsteigern; Grenzen setzen; auf das Wesentliche konzentrieren
gibt es scheinbar unlösbare Hindernisse / Probleme, dann Problem / Sachverhalt (z.B. im Quelle-Text) kurz notieren; dann erst mal eine andere Aufgabe (zum Ablenken) erledigen

11.x. Aufgaben aus der Abiturprüfung Informatik MV

aus rechtlichen Gründen wurden die Formulierungen der Aufgabenstellungen geändert
die eigentliche Aufgabenstellung bleibt aber erhalten

11.x.y. Abitur 2010

Rechner hat 177.122.66.99/16



Klasse IPRechner

11.x. Aufgaben der Landesolympiade Informatik MV

aus rechtlichen Gründen wurden die Formulierungen der Aufgabenstellungen geändert
die eigentliche Aufgabenstellung bleibt aber erhalten

11.x.y. 2014/2015

11.x.y.z. Sekundarstufe II

Literatur und Quellen:

- /1/ SANDE, Warren D.; SANDE, Carter:
Hello World! – Programmieren für Kids und Anfänger.-München: C. Hanser Verl..-2.,
akt. u. erw. Aufl.
ISBN 978-3-446-43906-4
- /2/ LINGL, Gregor:
Python für Kids.-Heidelberg, München, Landsberg, Frechen, Hamburg: bhv Verl. /
mitp Verl..-4. Aufl.
ISBN 978-3-8266-8673-3
- /3/ WEIGEND, Michael:
Python 3 – Lernen und professionell anwenden.- Heidelberg, München, Landsberg,
Frechen, Hamburg: mitp Verl..-5., akt. Aufl.
ISBN 978-3-8266-9456-1
- /4/ ARNHOLD, Werner:
Lieben Sie PYTHON?-IN: LOG IN, 21(2001) Heft 2.-S. 18 ff.-Berlin: LOG IN Verl.
ISSN 0720-8642
- /5/ MONK, Simon:
Raspberry Pi programmieren – Alle Befehle, und es klappt mit dem Raspberry.-Haar
bei München: Franzis Verl.; 2014
ISBN 978-3-645-60261-7
auch sonst als reine Python-Einführung sehr empfehlenswert
- /6/ VON LÖWIS, Martin; FISCHBECK, Nils:
Das Python-Buch – Referenz der objektorientierten Skriptsprache für GUIs und
Netzwerke.-Bonn: Addison-Wesley-Verl., 1997.- 1. Aufl.
ISBN 3-8273-1110-1
- /7/ ERNESTI, Johannes; KAISER, Peter:
Python 3 – Das umfassende Handbuch.-: Rheinwerk Verl..- 4. Aufl. 2015
ISBN 978-3-8362-3633-1
- /8/ :
.-: Verl..- Aufl.
ISBN 978-3-
- /8/ :
.-: Verl..- Aufl.
ISBN 978-3-
- /8/ :
.-: Verl..- Aufl.
ISBN 978-3-

/A/ Wikipedia
 <http://de.wikipedia.org>

Die originalen sowie detailliertere bibliographische Angaben zu den meisten Literaturquellen sind im Internet unter <http://dnb.ddb.de> zu finden.

Abbildungen und Skizzen entstammen den folgende ClipArt-Sammlungen:

/A/

andere Quellen sind direkt angegeben.

Alle anderen Abbildungen sind geistiges Eigentum:

lern-soft-projekt: drews (c,p) 1997 – 2026 lsp: dre
für die Verwendung außerhalb dieses Skriptes gilt für sie die Lizenz:



CC-BY-NC-SA



Lizenz-Erklärungen und –Bedingungen: <http://de.creativecommons.org/was-ist-cc/>
andere Verwendungen nur mit schriftlicher Vereinbarung!!!

verwendete freie Software:

- **Inkscape** von: inkscape.org (www.inkscape.org)
- **CmapTools** von: Institute for Human and Maschine Cognition (www.ihmc.us)

☐-	(c,p) 2015 - 2026 lern-soft-projekt: drews	-☐
☐-	drews@lern-soft-projekt.de	-☐
☐-	http://www.lern-soft-projekt.de	-☐
☐-	18069 Rostock; Luise-Otto-Peters-Ring 25	-☐
☐-	Tel/AB (0381) 760 12 18 FAX 760 12 11	-☐