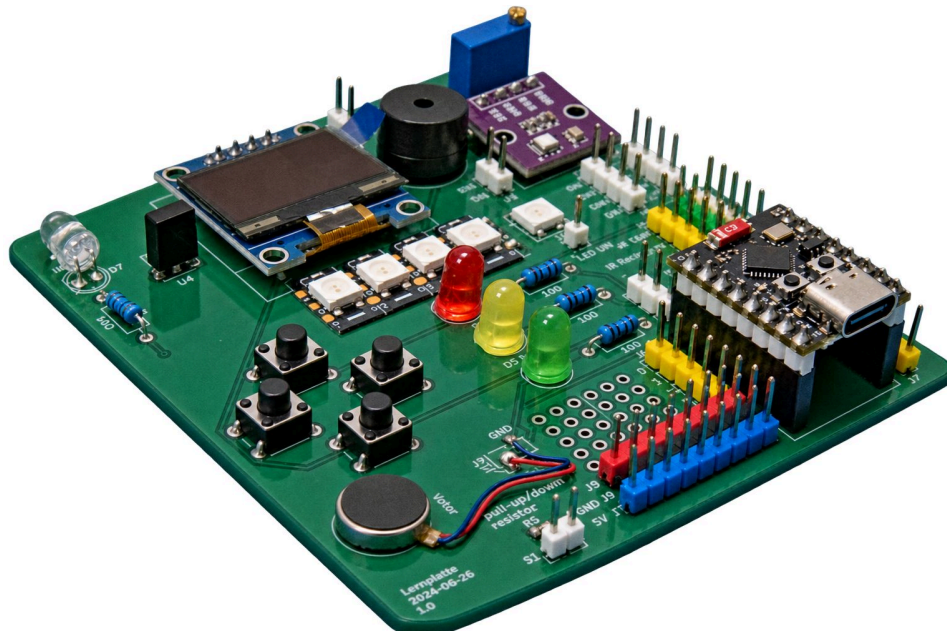


Programmieren lernen mit MicroPython auf dem ESP32-C3



Worum es geht

Am Ende läuft auf deiner Platine eine **Wetterstation**: Sie misst Temperatur, Luftfeuchtigkeit und Luftdruck, zeigt alles auf dem Bildschirm und startet an jeder Powerbank von allein – ohne Computer. Bis dahin sind es zehn Einheiten, jede mit ihrem eigenen Programm.

Das ist deine Platine

Auf der Titelseite siehst du sie in groß. Unten rechts sitzt der **Mikrocontroller** – der winzige Computer, der alles steuert. Ringsherum sitzen die Bauteile, die er ansteuern kann:

Drei LEDs – rot, gelb, grün

Vier Taster zum Drücken

Ein Drehregler zum Werte einstellen

Ein Vibrationsmotor, der brummt

Ein Summer, der piept

Ein Bildschirm mit 128 mal 64 Punkten

Ein Wettersensor für Temperatur, Feuchte, Luftdruck

Eine Lichterkette aus fünf bunten Lichtern

Infrarot zum Senden und Empfangen

Verbunden ist davon erst einmal nichts. Überall ragen Reihen kleiner Stifte aus der Platine, beschriftet mit **Poti**, **Motor**, **Buzzer** – nicht unbedingt dort, wo das Bauteil sitzt. Am Mikrocontroller sind es zwei Reihen: seine **Pins**, jeder mit einer Nummer. Ein Kabel steckst du auf beide, und erst dann kommt dein Programm an das Bauteil.

So ist jede Einheit aufgebaut

Drei Dinge kommen immer wieder vor. Hier siehst du sie einmal, dann erkennst du sie überall.

Kabel: Stufe 1 – rot an 7, gelb an 6, grün an 5.

Ganz oben steht, welche Kabel stecken müssen. Wenn dort „steckt schon“ steht, brauchst du nichts anzufassen.

DAS LERNST DU

Worum es in dieser Einheit geht – in einem Satz.

Der blaue Kasten am Anfang. Wenn du nur eine Zeile liest, dann diese.

PROBIER SELBST

- Lass die LED zehnmal blinken.
- Nimm die grüne statt der roten.

Am Ende jeder Einheit. Kleine Änderungen an dem Programm, das du gerade geschrieben hast – such dir aus, worauf du Lust hast.

Was wo steht

Die zehn Einheiten

01	Hallo Welt	Seite 3
02	Der Taschenrechner	Seite 4
03	Licht an	Seite 7
04	Der Drehregler	Seite 8
05	Die Ampel	Seite 9
06	Ein eigener Werkzeugkasten	Seite 11
07	Der Taster	Seite 14
08	Die Fußgängerampel	Seite 16
09	Die bunte Lichterkette	Seite 17
10	Die Wetterstation	Seite 20

Zum Nachschlagen

A1	Wenn etwas schiefgeht	Seite 22
A2	Spickzettel	Seite 23
A3	Zu Hause weitermachen	Seite 24
A4	Der Baukasten	Seite 27
A5	Alle Befehle	Seite 28
A6	Änderungshistorie	Seite 32

01 · Hallo Welt

DAS LERNST DU

Du sagst dem Mikrocontroller etwas, und er antwortet. Sofort, ohne dass du irgendetwas speichern musst.

Auf deiner Platine sitzt ein winziger Computer: ein **Mikrocontroller**. Viel langsamer als der Computer, an dem du sitzt, und er kann immer nur ein Programm – dafür schaltet er die Bauteile ringsum direkt. Eine Verbindung zu ihm brauchst du trotzdem, und dafür musst du **nichts installieren**: ein Browser reicht.

1. Steck die Platine mit dem USB-Kabel an den Computer.
2. Öffne **viper-ide.org**.
3. Klick auf **USB/Seriell verbinden**.
4. Der Browser fragt, welches Gerät er benutzen darf – wähl deine Platine.

Jetzt siehst du drei Bereiche: links den **Dateimanager** mit den Dateien auf der Platine, in der Mitte den **Editor**, unten das **Terminal**.

Im Terminal unterhältst du dich mit der Platine. Das `>>>` ganz links heißt: „Ich warte auf einen Befehl.“ Du tippst eine Zeile, drückst Enter – und die Antwort kommt sofort. Gespeichert wird dabei nichts, du darfst also alles ausprobieren.

Klick unten ins Terminal und tipp das hier – das `>>>` steht schon da:

```
>>> print("Hallo Welt")
Hallo Welt
```

Das ist dein erstes Programm. `print` heißt „schreib das hin“. Was in den Anführungszeichen steht, ist der Text – und der darf alles sein:

```
>>> print("Ich heiße Mia")
Ich heiße Mia
```

Wichtig sind die **Anführungszeichen** und die **Klammern**. Lass eins davon weg, und die Platine beschwert sich:

```
>>> print(Hallo Welt)
SyntaxError: invalid syntax
```

Das ist nicht schlimm. Tipp die Zeile einfach noch einmal richtig.

PROBIER SELBST

- Schreib deinen eigenen Namen hin.
- `print("Hallo", "Welt")` – was macht das Komma?
- Drück die Pfeiltaste ↑. Da ist deine letzte Zeile wieder.
- `print()` ganz ohne alles. Was passiert?

02 · Der Taschenrechner

DAS LERNST DU

Der Mikrocontroller rechnet. Und du kannst einem Ergebnis einen Namen geben, damit du es später wieder brauchen kannst.

Bleib im Terminal. Tipp einfach eine Rechnung hin – hier brauchst du nicht einmal `print`, die Antwort kommt von allein:

```
>>> 2 + 3
5
>>> 7 * 8
56
>>> 100 - 42
58
>>> 10 / 4
2.5
```

Der Stern `*` heißt „mal“ und der Schrägstrich `/` heißt „geteilt durch“ – die Zeichen `×` und `:` gibt es auf der Tastatur nicht. Klammern gehen auch, und sie bedeuten genau das, was du aus der Schule kennst:

```
>>> 2 + 3 * 4
14
>>> (2 + 3) * 4
20
```

Namen für Zahlen

Jetzt der Trick, auf den alles Weitere aufbaut. Du kannst einer Zahl einen **Namen** geben:

```
>>> alter = 11
>>> alter
11
>>> alter + 1
12
```

Das Gleichheitszeichen bedeutet hier nicht „ist gleich“, sondern „**bekommt den Namen**“. Links steht der Name, rechts das, was er bedeuten soll. So ein Name heißt **Variable**, weil sich dahinter etwas ändern darf:

```
>>> alter = 12
>>> alter
12
```

Namen gehen nicht nur für Zahlen, sondern auch für Text:

```
>>> name = "Mia"
>>> print("Hallo", name)
Hallo Mia
```

Und Namen dürfen miteinander rechnen:

```
>>> preis = 3
>>> anzahl = 4
>>> preis * anzahl
12
```

Genau das machst du gleich mit deiner LED. In der nächsten Einheit steht `led = LED(pin=7)`. Das ist dieselbe Zeile wie `alter = 11`: links ein Name, rechts das, was er bedeuten soll. Nur ist es diesmal keine Zahl, sondern eine LED.

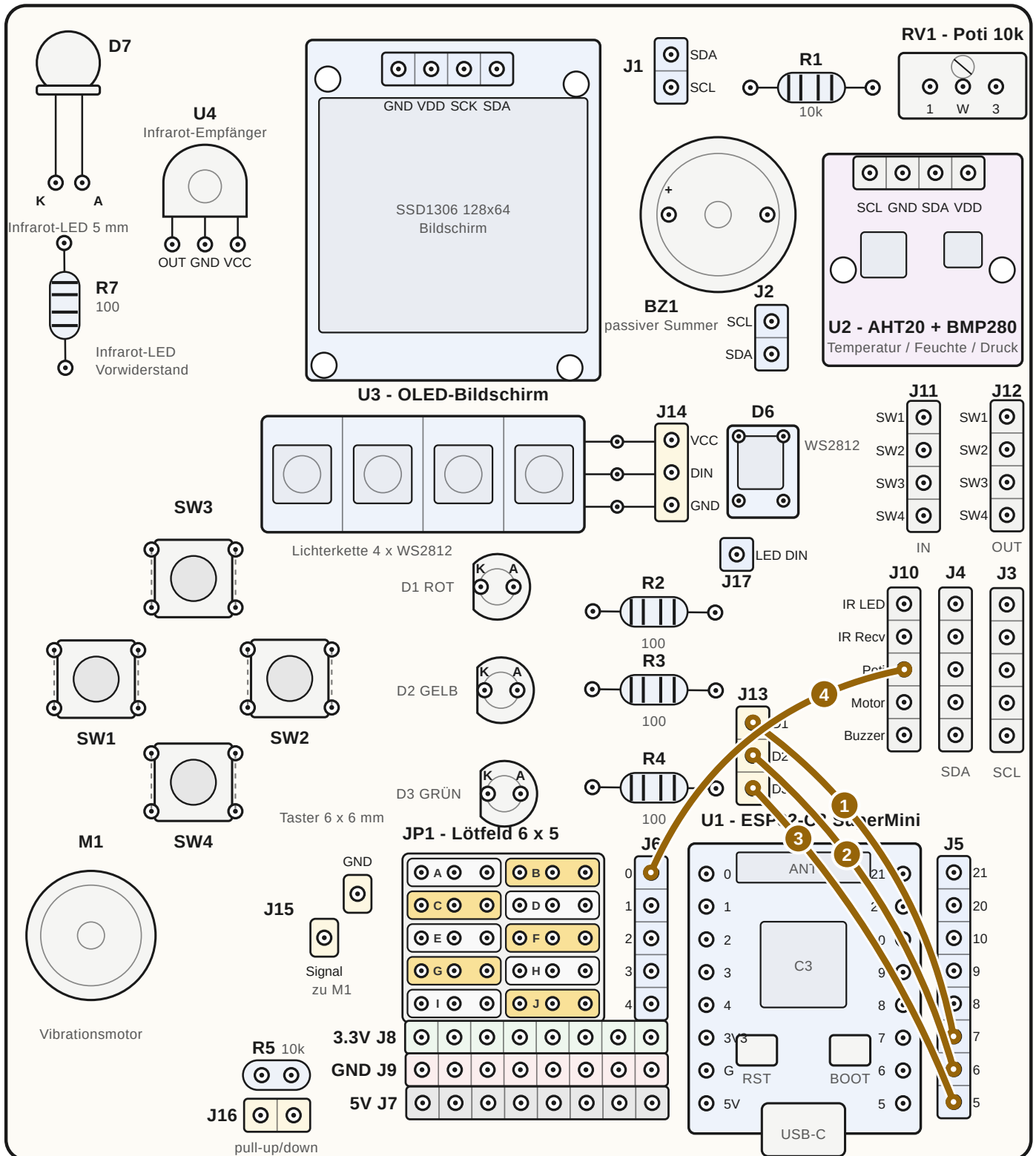
PROBIER SELBST

- Wie viele Sekunden hat ein Tag? `60 * 60 * 24`
- Wie viele Tage bist du alt? Nimm dein Alter mal 365.
- Gib zwei Zahlen Namen und rechne mit ihnen.
- Was sagt die Platine zu `"Mia" + 3`? Lies die Fehlermeldung.

Stufe 1

vor Einheit 03 - Licht an

4 Kabel



SO WIRD GESTECKT

- 1** D1 rot nach Pin 7
- 2** D2 gelb nach Pin 6
- 3** D3 grün nach Pin 5
- 4** Poti nach Pin 0

03 · Licht an

Kabel: Stufe 1 – rote LED an Pin 7, gelbe an 6, grüne an 5.

DAS LERNST DU

Ein Programm ist eine Liste von Befehlen. Der Mikrocontroller macht sie von oben nach unten, einen nach dem anderen.

Am Rand des Mikrocontrollers sitzen die **Pins** mit ihren Nummern. Jeden davon kann er einzeln an- und ausschalten: Strom da oder kein Strom. Die rote LED hängt mit einem Kabel an Pin 7 – schaltet dein Programm den Pin an, fließt Strom durch sie und sie leuchtet, schaltet es ihn aus, bleibt sie dunkel. Genau das machen `led.on()` und `led.off()`. Der Punkt dazwischen heißt „auf diesem Objekt“: `on` und `off` gehören zu `led`.

```
from toolkit import LED, sleep # der Werkzeugkasten - mehr in Einheit 06

led = LED(pin=7) # "led" ist ab jetzt der Name für die rote LED

led.on()
sleep(1) # 1 Sekunde warten
led.off()
sleep(1)

led.blink(times=5) # fünfmal blinken - das macht die Platine allein
```

Eine LED kann mehr als an und aus: Sie kann auch **heller oder dunkler** leuchten. Dafür schaltet der Mikrocontroller den Pin genauso an und aus wie vorher – nur sehr schnell, viele Male pro Sekunde. Ist die LED dabei länger an als aus, leuchtet sie heller; ist sie kürzer an, dunkler. Das Flackern siehst du nicht, dafür ist es viel zu schnell.

Die 0 ist ganz dunkel, die 100 ist ganz hell:

```
led.brightness = 5 # erst einstellen, wie hell ...
led.on() # ... dann einschalten
sleep(1)
led.brightness = 50
sleep(1)
led.brightness = 100
sleep(1)
led.off()
```

Experten nennen dieses schnelle An- und Ausschalten **PWM**, kurz für **Pulsweitenmodulation**.

PROBIER SELBST

- Lass die LED zehnmal blinken.
- Nimm die gelbe LED: `LED(pin=6)`. Dann die grüne: `LED(pin=5)`.
- Lass alle drei nacheinander blinken – wie ein Lauflicht.

04 · Der Drehregler

Kabel: Stufe 1 – Drehregler an Pin 0. Nur Pin 0 bis 4 können messen!

DAS LERNST DU

Ein Wert kann sich ändern, während das Programm läuft. Und du kannst ihn von einem Bauteil zum anderen weiterreichen.

Der Knopf, an dem du drehst, heißt richtig **Potentiometer** – kurz Poti. Drin steckt eine Widerstandsbahn und ein Schleifer, der beim Drehen darüberfährt. Je nachdem wo er steht, ist der **Widerstand** größer oder kleiner: Der Strom hat es schwerer oder leichter.

Und diesen Unterschied kann der Mikrocontroller **messen**. Messen geht nur an den Pins 0 bis 4. Der gemessene Wert wird bei uns auf eine **Zahl von 0 bis 100** umgewandelt.

Für Detailverliebte: Wir messen nicht den Widerstand direkt, sondern die Spannung, die sich durch den Widerstand ändert. Wir verwenden dazu etwas, das sich „Spannungsteiler“ nennt.

Erst mal nur nachschauen, was der Regler sagt. Dreh langsam! Die erste Zeile heißt „mach das 75-mal“ – so etwas nennt man eine **Schleife**, und in Einheit 05 baust du selbst eine.

```
from toolkit import LED, Dial, sleep

led = LED(pin=7)      # die rote
dial = Dial(pin=0)

for nummer in range(75):
    print(nummer, "Der Regler steht auf", dial.value)
    sleep(0.3)
```

Und jetzt der Trick: der Regler zählt von 0 bis 100. Die Helligkeit auch. Darum passt das einfach zusammen:

```
led.on()                # an - wie hell, sagt der Regler
while True:              # "immer wieder"
    led.brightness = dial.value
    sleep(0.05)
```

PROBIER SELBST

- Dreh so, dass die LED ganz schwach leuchtet. Welche Zahl ist das?
- Tausch die rote LED gegen die grüne (`pin=5`).
- `dial.between(0, 10)` rechnet den Regler auf 0 bis 10 um. Was zeigt `print(dial.between(0, 10))`?

05 · Die Ampel

Kabel: Stufe 1 – rot an 7, gelb an 6, grün an 5. Steckt schon.

DAS LERNST DU

Eine eigene Funktion gibt mehreren Befehlen einen Namen. Und eine Funktion darf andere Funktionen benutzen.

Drei Lichter – also drei LEDs, jede mit ihrem eigenen Namen. Genau wie in Beispiel 03, nur eben dreimal:

```
from toolkit import LED, sleep

rot = LED(pin=7)
gelb = LED(pin=6)
gruen = LED(pin=5)
```

Eine deutsche Ampel hat vier Phasen. Erst von Hand:

```
rot.on()           # 1. Rot - alle stehen
sleep(3)

gelb.on()          # 2. Rot UND Gelb - gleich geht es los
sleep(1)

rot.off()
gelb.off()
gruen.on()         # 3. Grün - fahren
sleep(3)

gruen.off()
gelb.on()          # 4. Gelb - langsam werden
sleep(1)
gelb.off()
```

Das war viel Tipperei – und für die nächste Runde nochmal dasselbe. Gib jeder Phase einen Namen. So ein Name für mehrere Befehle ist eine **eigene Funktion**:

```
def zeige_rot():
    rot.on()
    gelb.off()
    gruen.off()

def zeige_rot_gelb():
    rot.on()
    gelb.on()
    gruen.off()
```

Und genauso für die beiden anderen Phasen:

```
def zeige_gruen():
    rot.off()
    gelb.off()
    gruen.on()

def zeige_gelb():
    rot.off()
    gelb.on()
    gruen.off()
```

Ab jetzt schaltest du eine ganze Phase mit einem einzigen Wort:

```
zeige_rot()
sleep(1)
zeige_gruen()
```

Und damit wird eine ganze Runde kurz genug, um sie auf einen Blick zu lesen. Eine Funktion darf nämlich andere Funktionen benutzen – genau dafür sind sie da:

```
def eine_runde():
    zeige_rot()
    sleep(3)
    zeige_rot_gelb()
    sleep(1)
    zeige_gruen()
    sleep(3)
    zeige_gelb()
    sleep(1)
```

Jetzt reicht ein Wort für die ganze Runde. Und mit einer **Schleife** gleich dreimal hintereinander:

```
for nummer in range(3):
    print("Runde", nummer + 1)
    eine_runde()
```

Wusstest du das? Rot und Gelb zusammen gibt es fast nur in Deutschland. In vielen Ländern springt die Ampel direkt von Rot auf Grün.

PROBIER SELBST

- Mach die Grünphase länger: `sleep(3)` zu `sleep(8)`.
- Lass die Ampel zehn Runden fahren.
- Schreib dir eine Funktion `alles_aus()`, die alle drei ausschaltet.
- Nachts blinkt eine Ampel nur gelb: `gelb.blink(times=10)`.
- Bau daraus eine Funktion `nachtmodus()`.

06 · Ein eigener Werkzeugkasten

Kabel: Stufe 1 – unverändert. Es kommt kein Kabel dazu.

DAS LERNST DU

Funktionen, die du öfter brauchst, kommen in eine eigene Datei. Andere Programme holen sie sich von dort.

Deine vier Phasen-Funktionen sind gut – so gut, dass du sie gleich noch einmal brauchst. Und beim nächsten Programm wieder. Sie jedes Mal abzutippen wäre albern.

Also legst du sie dorthin, wo jedes Programm sie findet: in eine **eigene Datei**. Mach in ViperIDE eine neue Datei auf, nenn sie `ampelhilfe.py` und schreib das hier hinein:

```
from toolkit import LED

rot = LED(pin=7)
gelb = LED(pin=6)
gruen = LED(pin=5)

def zeige_rot():
    rot.on()
    gelb.off()
    gruen.off()
```

Darunter kommen `zeige_rot_gelb()`, `zeige_gruen()` und `zeige_gelb()` – genau so, wie du sie in Einheit 05 geschrieben hast. Und ganz unten noch eine, die du sowieso dauernd tippst:

```
def alles_aus():
    rot.off()
    gelb.off()
    gruen.off()
```

Speichern – als `ampelhilfe.py` auf der Platine.

Diese Datei ist **kein Programm zum Zuschauen**. Wenn du sie ausführst, passiert nichts – sie enthält ja nur Bausteine. Sie ist ein Werkzeugkasten, und die Werkzeuge holt sich jemand anders heraus.

Und jetzt hol sie dir

Ein neues Programm. Die erste Zeile kennst du schon – die beiden darunter sind das Neue:

```
from toolkit import sleep
from ampelhilfe import zeige_rot, zeige_rot_gelb, zeige_gruen, zeige_gelb
from ampelhilfe import alles_aus, gelb
```

`from ampelhilfe import ...` heißt: „hol dir diese Namen aus der Datei `ampelhilfe.py`“. Genau wie `from toolkit import LED`, nur dass `ampelhilfe` diesmal von dir ist.

Darunter kommt dieselbe Ampel wie eben – nur ist das Programm auf die Hälfte geschrumpft:

```
def eine_runde():
    zeige_rot()
    sleep(3)
    zeige_rot_gelb()
    sleep(1)
    zeige_gruen()
    sleep(3)
    zeige_gelb()
    sleep(1)

for nummer in range(3):
    eine_runde()

alles_aus()
```

Warum ist `eine_runde()` nicht mitgewandert? In den Werkzeugkasten kommt, was **mehrere** Programme brauchen. Wie lang eine Grünphase dauert, entscheidet aber jedes Programm für sich – eine Fußgängerampel hat da ganz andere Zeiten. Die vier Phasen sind für alle gleich, die Runde nicht.

Auch die drei LEDs wohnen jetzt in `ampelhilfe.py`. Das muss so sein: Ein Pin darf immer nur **einem** Bauteil gehören. Würdest du hier noch einmal `LED(pin=7)` schreiben, gäbe es Streit um den Pin – und die Platine sagt das auch deutlich. Deshalb holst du dir `gelb` aus der Datei, wenn du es brauchst:

```
for nummer in range(10): # nachts blinkt nur Gelb
    gelb.on()
    sleep(0.5)
    gelb.off()
    sleep(0.5)
```

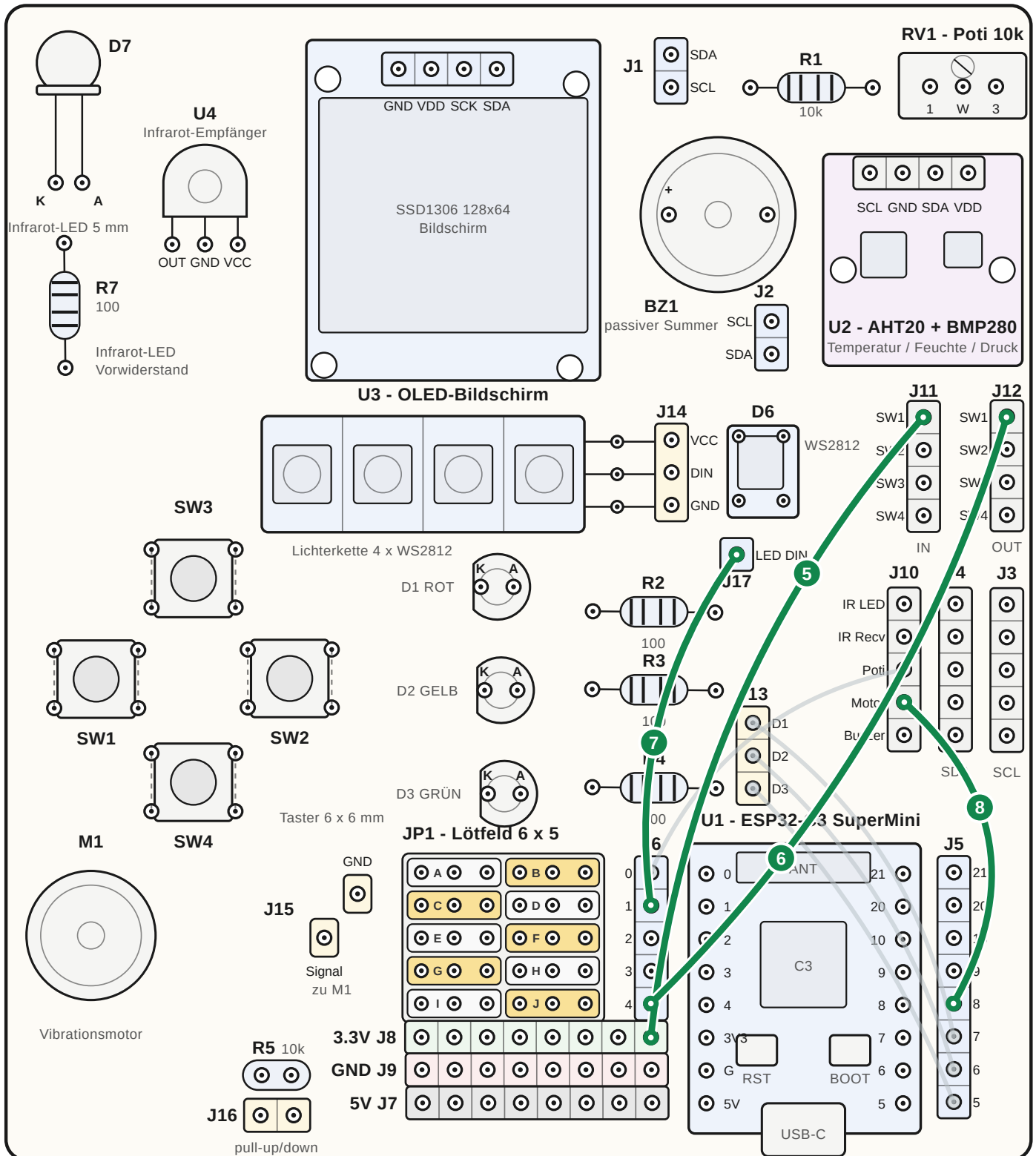
PROBIER SELBST

- Bau `nachtmodus()` in `ampelhilfe.py` ein und ruf es hier auf.
- Ändere in `ampelhilfe.py`, wie `zeige_gruen()` aussieht. Dieses Programm änderst du dabei **nicht** – und es stimmt trotzdem wieder.
- Benenn `ampelhilfe.py` auf der Platine um. Was sagt das Programm dann?

Stufe 2

vor Einheit 07 - Der Taster

4 Kabel



SO WIRD GESTECKT

- 5 SW1 IN (links) nach 3,3 V
- 6 SW1 OUT (rechts) nach Pin 4

- 7 LED DIN nach Pin 1
- 8 Motor Signal nach Pin 8

07 · Der Taster

Kabel: Stufe 2 – der Taster hängt jetzt an Pin 4.

DAS LERNST DU

Mit `if` trifft das Programm eine Entscheidung. Ab jetzt reagiert die Platine auf dich.

Fast alles kennst du schon: Die drei Lichter liegen griffbereit in `ampelhilfe.py`. Neu ist nur der Taster.

```
from toolkit import Button, sleep
from ampelhilfe import zeige_rot, zeige_gruen, alles_aus

taster = Button(pin=4)
```

Das Erste ist **Warten**. Das Programm bleibt in dieser Zeile stehen, so lange es dauert:

```
print("Drücke den Taster!")
taster.wait_for_press()
print("Gedrückt!")

zeige_gruen()
sleep(1)
alles_aus()
```

Die Platine hat dabei übrigens mitgezählt:

```
print("Bis jetzt:", taster.presses, "mal gedrückt.")
```

Warten oder nachschauen? Beim Warten steht alles still, bis endlich gedrückt wird. Beim Nachschauen läuft dein Programm weiter und fragt nur immer wieder kurz nach. Alles, was nebenbei noch passieren soll, braucht das Nachschauen.

Nachschauen statt warten

`taster.is_pressed` sagt, ob der Taster *gerade jetzt* gedrückt ist. Und `if` entscheidet, was dann passieren soll:

```
print("Halt den Taster gedrückt. Strg-C beendet.")

while True:
    if taster.is_pressed:
        zeige_gruen()
    else:
        zeige_rot()
    sleep(0.05)
```

`if` heißt „wenn“, `else` heißt „sonst“. Achte auf den **Doppelpunkt** am Zeilenende und darauf, dass die Zeilen darunter eingerückt sind – genau wie bei `def`.

`while True` – immer wieder

Die erste Zeile kennst du schon vom Drehregler. Jetzt lohnt es sich, genauer hinzuschauen:

`while True:` heißt „**mach das immer wieder**“. Ohne Ende. Dieses Programm hört von allein nicht mehr auf.

Genau das willst du hier. Ein Gerät, das auf dich reagiert, muss ja auch immer wieder nachschauen – ein einziger Blick würde nichts nützen. Ampeln, Wecker und Spiele sind alle so gebaut: Sie laufen, bis jemand den Stecker zieht.

Zwei Dinge, die du dazu wissen musst.

Anhalten kannst du so ein Programm mit *Ausführung unterbrechen* in ViperIDE – oder mit **Strg-C** unten im Terminal.

Alles, was du unter die Schleife schreibst, kommt nie dran. Die Schleife ist ja nie fertig. Wenn also am Ende deines Programms etwas nicht passiert: schau nach, ob darüber ein `while True` steht.

Und das kleine `sleep(0.05)` ganz unten? Das ist die Pause, in der der Mikrocontroller kurz Luft holt und sich um alles kümmert, was nebenbei läuft. Dazu gehört auch nachzusehen, ob der Taster gerade gedrückt ist.

Für Detailverliebte: Im Taster steckt eine Feder, der Knopf muss schließlich nach dem Drücken wieder nach oben. Dabei federt der Kontakt kurz nach und schaltet mehrmals schnell hintereinander an und aus, obwohl du nur einmal gedrückt hast. Fachleute nennen das **Prellen**. Der Mikrocontroller filtert es heraus – auch das passiert in der Pause.

Noch mehr Taster? Auf deiner Platine sitzen vier Stück: SW1 bis SW4. SW1 ist der, den du gerade benutzt – die anderen drei sind mit nichts verbunden, du entscheidest selbst, wohin sie gehen.

Nimm zum Beispiel SW2. Der braucht zwei Kabel, genau wie SW1: eines von **3,3 V** nach **SW2 IN (links)**, und eines von **SW2 OUT (rechts)** zu einem freien Pin, zum Beispiel Pin 2. Im Programm dann `taster2 = Button(pin=2)` – und alles geht wie gehabt.

PROBIER SELBST

- Dreh es um: grün, solange **nicht** gedrückt wird.
- `taster.wait_for_press(timeout=3)` wartet höchstens 3 Sekunden.
- Schreib eine Zeile **unter** die Schleife und lass sie laufen. Passiert etwas?
- Lass bei jedem Druck eine andere LED leuchten. Tipp: `taster.presses` zählt hoch, und `% 3` gibt den **Rest beim Teilen durch 3** – also immer 0, 1 oder 2.

08 · Die Fußgängerampel

Kabel: Stufe 2 – die drei Lichter an 7, 6 und 5, Taster an 4, Vibrationsmotor an 8.

DAS LERNST DU

Jetzt baust du ein richtiges Gerät. Es wartet, es reagiert, es läuft einen Ablauf ab – und wartet dann wieder.

Und jetzt zählt sich der Werkzeugkasten aus: Die vier Phasen holst du dir einfach aus `ampelhilfe.py`. Hier steht nur noch das, was **diese** Ampel von jeder anderen unterscheidet.

```
from toolkit import Button, Vibration, sleep
from ampelhilfe import zeige_rot, zeige_rot_gelb, zeige_gruen, zeige_gelb

taster = Button(pin=4)
motor = Vibration(pin=8)

zeige_gruen()                # die Autos dürfen fahren
print("Autos fahren. Drücke den Taster!")

while True:
    taster.wait_for_press()    # hier steht das Programm still
    print("Anforderung!")
    motor.buzz()               # ich hab dich gehört

    zeige_gelb()
    sleep(2)
    zeige_rot()

    # Jetzt dürfen die Fußgänger gehen. Der Motor brummt im Takt.
    for schritt in range(8):
        motor.buzz(on=0.08, off=0.32)

    zeige_rot_gelb()
    sleep(1)
    zeige_gruen()
    print("Autos fahren wieder.")
```

Das gibt es wirklich. Unten am Ampelkasten sitzt ein kleiner Kegel, der brummt, wenn es grün wird. Der ist für Menschen, die weder sehen noch hören können – sie fühlen ihn mit der Hand.

PROBIER SELBST

- Lass den Motor schneller brummen, kurz bevor Rot zu Ende ist.
- Warte nach der Anforderung noch 3 Sekunden. Echte Ampeln machen das, damit nicht jeder Knopfdruck sofort alles stoppt.
- Zähl, wie oft heute jemand gedrückt hat: `taster.presses`

09 · Die bunte Lichterkette

Kabel: Stufe 2 – die Lichterkette hängt an Pin 1. Steckt schon.

DAS LERNST DU

Jede Farbe ist eine Mischung aus Rot, Grün und Blau. Mit Licht mischt es sich anders als mit Wasserfarbe.

Fünf Lichter in einer Reihe, und jedes kann jede Farbe. Solche Ketten heißen überall **Neopixel** – so findest du sie auch im Internet. Am einfachsten geht es mit Farbnamen:

```
from toolkit import Pixels, Dial, sleep

pixels = Pixels(pin=1, count=5)
pixels.brightness = 30          # sonst blendet die Lichterkette

pixels.set("rot", "gelb", "gruen", "blau", "lila")
sleep(2)
pixels.rainbow(times=2)
pixels.off()
```

Ist dir die Lichterkette zu hell? `pixels.brightness` geht von 0 bis 100 und gilt für alle fünf zusammen – ohne diese Zeile wären es 50. Probier ruhig auch 5. Die Farben bleiben dabei dieselben, nur die Helligkeit ändert sich.

Was hinter den Namen steckt

In jedem der fünf Lichter sitzen **drei** winzige LEDs: eine rote, eine grüne und eine blaue. Mehr gibt es nicht. Jede stellst du einzeln ein, von **0** (aus) bis **255** (ganz hell) – drei Zahlen in Klammern, immer in dieser Reihenfolge:

```
pixels.fill((255, 0, 0))      # nur die rote LED
pixels.fill((0, 255, 0))     # nur die grüne
pixels.fill((0, 0, 255))     # nur die blaue
```

Und jetzt zwei gleichzeitig. Das Ergebnis überrascht die meisten:

```
pixels.fill((255, 255, 0))    # rot + grün  -> GELB
pixels.fill((0, 255, 255))    # grün + blau -> Türkis
pixels.fill((255, 0, 255))    # rot + blau  -> Pink
pixels.fill((255, 255, 255))  # alle drei  -> Weiß
```

Im Malkasten geben Rot und Grün ein schlammiges Braun. Hier geben sie Gelb. Der Unterschied: Farbe auf Papier *schluckt* Licht – je mehr du mischst, desto dunkler wird es. LEDs dagegen *geben* Licht dazu. Je mehr du dazutust, desto heller wird es, bis am Ende Weiß herauskommt.

Du musst nicht 0 oder 255 nehmen, alles dazwischen geht auch:

```
pixels.fill((255, 128, 0))    # viel Rot, halb Grün -> Orange
pixels.fill((128, 0, 255))   # halb Rot, viel Blau -> Violett
```

Und jedes Licht darf etwas anderes zeigen. Die fünf sind durchnummeriert, aber von 0 bis 4:

`pixels[0]` ist das erste, `pixels[4]` das letzte. Hier ein Verlauf von Rot nach Grün über die ganze Lichterkette:

```
pixels[0] = (200, 0, 0)      # ganz rot
pixels[1] = (150, 50, 0)
pixels[2] = (100, 100, 0)
pixels[3] = (50, 150, 0)
pixels[4] = (0, 200, 0)     # ganz grün
```

Der Drehregler mischt live

Der Regler zählt von 0 bis 100. Mal 2 gerechnet kommt er auf 0 bis 200 – genau die richtige Größe für eine LED:

```
dial = Dial(pin=0)

while True:
    menge = dial.value * 2          # 0 bis 200
    pixels.fill((200 - menge, menge, 0))
    sleep(0.05)
```

Wie das Ding eigentlich heißt. Jedes dieser Lichter ist ein eigener kleiner Chip mit dem Namen **WS2812** – so steht er auch auf der Zeichnung deiner Platine. In ihm stecken die drei LEDs und eine winzige Schaltung dazu: Sie nimmt sich die Farbe, die für ihn bestimmt ist, und reicht den Rest an den nächsten weiter. Darum reicht ein einziges Kabel für alle fünf.

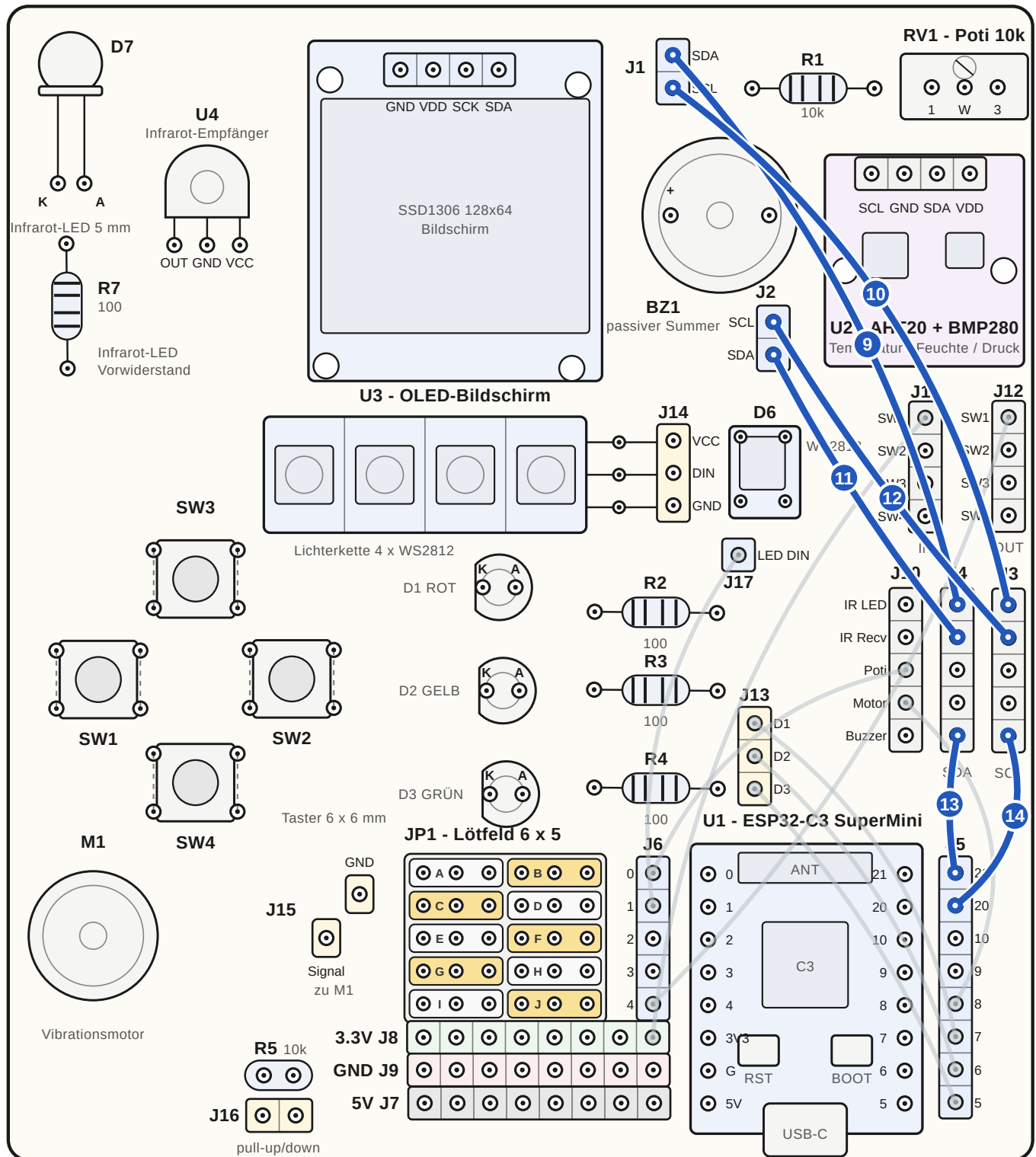
PROBIER SELBST

- Misch deine Lieblingsfarbe. Schreib die drei Zahlen auf.
- `pixels.bar(dial.value)` – der Regler als Balkenanzeige.
- Lass den Regler zwischen Blau und Gelb mischen statt Rot und Grün.
- `pixels.brightness = 5` – das gilt für alle drei LEDs gleichzeitig, die Farbe bleibt dieselbe.

Stufe 3

vor Einheit 10 - Wetterstation

6 Kabel



SO WIRD GESTECKT

- 9 Display SDA nach SDA-Schiene
- 10 Display SCL nach SCL-Schiene
- 11 Sensor SDA nach SDA-Schiene
- 12 Sensor SCL nach SCL-Schiene
- 13 SDA-Schiene nach Pin 21
- 14 SCL-Schiene nach Pin 20

10 · Die Wetterstation

Kabel: Stufe 3 – Bildschirm und Sensor an SCL 20 / SDA 21. Die Pixel stecken schon.

DAS LERNST DU

Mehrere Bauteile können sich Kabel teilen. So etwas nennt sich dann **Bus**.

Bisher hatte jedes Bauteil sein eigenes Kabel: die rote LED an Pin 7, der Taster an Pin 4. Das geht nicht ewig – der Mikrocontroller hat nur dreizehn Pins, und Bildschirm und Wettersensor wären zwei weitere.

Hier machst du es anders. **Bildschirm und zwei Sensorchips hängen an denselben zwei Kabeln.** Zwei Kabel für alles, was noch kommt.

So ein geteiltes Kabelpaar heißt Bus. Es funktioniert wie ein Klassenzimmer: Alle hören mit, aber es antwortet nur der, dessen **Nummer** aufgerufen wird. Jedes Bauteil hat nämlich eine eingebaute Nummer, die es ab Werk mitbringt.

Das eine Kabel heißt **SDA** – darüber gehen die Nummern und die Daten. Das andere heißt **SCL** und gibt den Takt vor, wann Nummer und Daten gesendet werden. SDA und SCL bilden den Bus. Dieser Bus heißt **I²C** – sprich „I-Quadrat-C“.

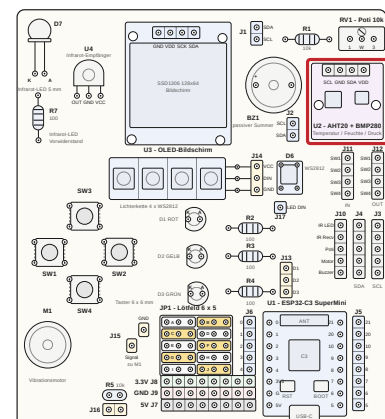
Zuerst nachschauen, wer sich meldet:

```
from toolkit import check_i2c
check_i2c(scl=20, sda=21)
```

```
I2C an SCL=20, SDA=21:
0x3C  Bildschirm
0x38  AHT20 (Temperatur, Feuchte)
0x77  BMP280 (Luftdruck)
```

Das sind genau die Nummern. **0x3C** ist der Bildschirm, und wenn der Mikrocontroller **0x3C** ruft, hört der Wettersensor weg. Drei Zeilen müssen da stehen – fehlt eine, sitzt ein Kabel falsch, und das bringst du **erst** in Ordnung.

Übrigens: Es sind drei Zeilen, obwohl du nur zwei Bauteile gesteckt hast. Auf dem kleinen Brettchen oben rechts sitzen nämlich **zwei** Chips – einer misst Temperatur und Feuchte, der andere den Luftdruck. Für den Bus sind das zwei Teilnehmer mit zwei Nummern.



Das Modul mit zwei Sensorchips

Messen und anzeigen

Jetzt das eigentliche Gerät. Es tut immer wieder dasselbe drei Dinge: den Sensor **lesen**, die Werte auf den Bildschirm **schreiben**, und aus der Temperatur eine **Farbe** machen.

Der Sensor misst von allein weiter, im Hintergrund. Du fragst ihn nur nach dem letzten Wert:

`weather.temperature` sind Grad, `weather.humidity` sind Prozent, `weather.pressure` ist der Luftdruck. Und `screen.line(0, ...)` schreibt in die erste Zeile des Bildschirms,

`screen.line(1, ...)` in die zweite.

Der Bildschirm wartet auf `show()`. Alles, was du malst, sammelt sich erst im Speicher – zu sehen ist noch nichts. Erst `screen.show()` schickt das fertige Bild hin. Danach ist der Speicher wieder leer, das nächste Bild fängt bei Null an. Deshalb steht auch `title()` mit in der Schleife.

```
from toolkit import Screen, Weather, Pixels, sleep

screen = Screen(scl=20, sda=21)
weather = Weather(scl=20, sda=21)
pixels = Pixels(pin=1, count=5)

while True:
    # Erst das ganze Bild malen ...
    screen.title("Wetter")
    screen.line(0, "%.1f Grad" % weather.temperature)
    screen.line(1, "%.0f %% Feuchte" % weather.humidity)
    screen.line(2, "%.0f hPa" % weather.pressure)
    screen.show()          # ... und dann hinschicken

    # Die Pixel als Thermometer
    farbe = "gruen"
    if weather.temperature < 18:
        farbe = "blau"
    if weather.temperature > 24:
        farbe = "rot"
    pixels.fill(farbe)

    sleep(2)
```

Was macht das % im Text? Es ist ein Platzhalter: `%.1f` heißt „hier kommt eine Zahl mit einer Stelle nach dem Komma hin“, `%.0f` heißt „gar keine Stelle nach dem Komma“. Nach dem Text sagt das zweite %, welche Zahl eingesetzt wird. Und weil % selbst der Platzhalter ist, schreibt man `%%`, wenn ein echtes Prozentzeichen dastehen soll.

PROBIER SELBST

- Halt den Finger auf den Sensor. Wie schnell steigt die Zahl?
- `screen.big("21")` macht eine riesige Zahl – `show()` nicht vergessen.
- `pixels.bar(weather.humidity)` zeigt die Feuchte als Balken.
- `weather.altitude` sagt, wie hoch du über dem Meer bist.

A1 · Wenn etwas schiefgeht

Fehlermeldungen sind nichts Schlimmes: Sie sagen dir, was der Mikrocontroller mit deinem Programm nicht anfangen konnte. Was der Werkzeugkasten selbst bemerkt, schreibt er auf Deutsch; alles andere kommt aus der Programmiersprache und ist englisch. Die häufigsten stehen hier: links, wie sie im Terminal erscheinen, rechts, was du tun kannst.

DAS STEHT DA

Pin 5 wird schon ... benutzt

... kann keine Spannung messen

IndentationError

SyntaxError

NameError: ... isn't defined

ImportError: no module named ...

Das Programm hört nicht mehr auf

Es passiert gar nichts

Die rote LED leuchtet, ohne dass du sie anschaltest

DAS MACHST DU

Zwei Objekte greifen nach demselben Pin: `LED(pin=5)` steht zweimal im Programm – oder auch in `ampelhilfe.py`. Benutze das vorhandene weiter.

Der Drehregler muss an Pin 0, 1, 2, 3 oder 4.

Eine Zeile ist falsch eingerückt. Alle Zeilen in einem Block gleich weit nach rechts.

Meistens fehlt ein Doppelpunkt am Zeilenende oder eine Klammer.

Den Namen kennt die Platine nicht. Tippfehler – oder du hast ihn oben noch nicht geholt: `from toolkit import ...`

Die Datei fehlt auf der Platine oder heißt anders. Schau im Dateimanager nach, ob `ampelhilfe.py` wirklich so heißt.

Da läuft ein `while True`. **Ausführung unterbrechen** anklicken, oder **Strg-C** unten im Terminal.

Sitzt das Kabel wirklich auf dem Stift? Und auf dem **richtigen**?

Brummt gerade der Vibrationsmotor? Das ist ein bekannter Fehler der Platine, nicht deines Programms – siehe A6, Seite 32.

Die letzte Zeile ist die wichtige. Wenn etwas schiefgeht, schreibt die Platine oft mehrere Zeilen. Ganz unten steht, was los ist – und meistens auch, in welcher Zeile deines Programms.

A2 · Spickzettel

Alles, was deine Platine kann – auf einer Seite.

LED `LED(pin=7)`

```
led.on()    led.off()
led.toggle()
led.blink(times=5)
led.pulse()
led.brightness = 0..100
led.is_on
```

Vibrationsmotor `Vibration(pin=8)`

```
motor.buzz()
motor.buzz(on=0.1, off=0.4)
motor.on()   motor.off()
motor.strength = 0..100
```

Taster `Button(pin=4)`

```
taster.is_pressed
taster.wait_for_press()
taster.wait_for_release()
taster.presses
taster.when_pressed = machwas
```

Drehregler `Dial(pin=0)`

```
dial.value
dial.between(0, 255)
dial.when_changed = zeigen
```

Summer (Extra-Kabel) `Buzzer(pin=10, volume=25)`

```
summer.beep()
summer.play("C4")
summer.tone(880)
summer.melody("C4 E4 G4")
summer.volume = 0..100
summer.silence()
```

Pixel `Pixels(pin=1, count=5)`

```
pixels[0] = "rot"
pixels.fill("blau")
pixels.bar(50)
pixels.rainbow()
pixels.chase("rot")
pixels.off()
```

Bildschirm `Screen(scl=20, sda=21)`

```
screen.title("Wetter")
screen.line(0, "Hallo")
screen.big("21")
screen.write("mehr Text")
screen.clear()
screen.show()
```

Wetter `Weather(scl=20, sda=21)`

```
weather.temperature
weather.humidity
weather.pressure
weather.altitude
```

Farben Namen: rot · gelb · gruen · blau · lila · rosa · orange · tuerkis · weiss · braun · gold · aus
oder drei Zahlen in der Reihenfolge rot, grün, blau von 0 bis 255: (255, 0, 0)

Immer dabei `sleep(1) · forever() · report_pins() · hilfe() ·`
`check_i2c(scl=20, sda=21)`

A3 · Zu Hause weitermachen

Die Platine läuft auch ohne Computer

Steck ein Handy-Ladegerät oder eine Powerbank an die USB-C-Buchse. Das Programm, das `main.py` heißt, startet dann ganz von allein.

So sind deine Kabel gesteckt

BAUTEIL	PIN
LED rot / gelb / grün	7 / 6 / 5
Drehregler	0
Taster	4
Lichterkette	1
Vibrationsmotor	8
Bildschirm + Wettersensor	SCL 20, SDA 21

Frei zum Weiterbauen: Pin 2, 3 und 10 (der Summer).

Pin 9 lässt du frei – wenn dort etwas hängt, startet die Platine nicht mehr.

Am Computer weiterarbeiten

Du musst gar nichts installieren. Es reicht ein Browser:

1. Platine mit dem USB-Kabel an den Computer stecken.
2. **viper-ide.org** öffnen.
3. Auf **USB/Seriell verbinden** klicken.
4. Der Browser fragt, welches Gerät er benutzen darf – deine Platine auswählen und bestätigen.

Fertig. Im **Dateimanager** links stehen die Dateien auf der Platine, in der Mitte der Editor, unten das **Terminal** mit deinen `print`-Ausgaben.

Nimm dafür **Chrome** (oder Edge, das ist dasselbe in Grün). Safari kann es nicht, und Handy oder Tablet gehen auch nicht – es muss ein Computer sein.

Das sind die vier Dinge, die du dort machst:

- **Ausprobieren:** Datei schreiben, dann *Datei ausführen*.
- **Fest speichern:** *Datei speichern* und sie `main.py` nennen – dann startet sie von allein, sobald Strom da ist, auch ohne Computer.
- **Autostart weg:** `main.py` im *Dateimanager* löschen.
- **Direkt tippen:** unten ins *Terminal*, zum Beispiel `led.on()`.

Wenn die Platine nicht mehr antwortet, weil `main.py` in einer Schleife läuft: *Ausführung unterbrechen* anklicken (oder im Terminal **Strg-C**).

Auf einem Schulcomputer kann es sein, dass gar keine Verbindung angeboten wird – solche Rechner sind oft so eingestellt, dass Webseiten nicht an die USB-Anschlüsse dürfen. Dann hilft nur ein privater Computer.

Die Platine auf den neuesten Stand bringen

Manchmal gibt es eine neuere Fassung unserer Software – mit neuen Funktionen oder mit weniger Fehlern. Dafür gibt es eine eigene Seite, und du musst dafür wieder nichts installieren:

1. Platine mit dem USB-Kabel anstecken.
2. **workshops.backspace.de/2026/ferien-mikrocontroller** öffnen – mit **Chrome** oder **Edge**, wie bei ViperIDE.
3. Auf **Die Software aktualisieren** klicken.
4. Auf **Platine verbinden** klicken und deine Platine auswählen.
5. Auf **Jetzt aktualisieren** klicken.

Die Seite schaut selbst nach, was auf deiner Platine liegt, und schreibt nur das, was sich wirklich geändert hat. Das dauert ein paar Sekunden.

Vor dem Aktualisieren stehen dort ein paar **Kästchen** für Dinge, die nicht jede Platine braucht – zum Beispiel die Vorlagen für den Baukasten. Setz das Häkchen, wenn du sie haben willst. Nimmst du es wieder weg, räumt die Seite den Ordner beim nächsten Mal auch wieder ab.

Achtung, eine Sache hat sich geändert. Der Bildschirm zeigt seit dieser Fassung nichts mehr von allein. Alles, was du malst, sammelt sich erst im Speicher, und erst `screen.show()` schickt es hin.

Ein älteres Programm, das den Bildschirm benutzt, bleibt danach also leer. Der Fehler ist schnell behoben: Schreib `screen.show()` ans Ende deiner Schleife – und `screen.title(...)` mit **in** die Schleife hinein, weil der Speicher nach jedem `show()` wieder leer ist.

Genau so steht es schon im Programm der Wetterstation in **Einheit 10** (Seite 20) – dort kannst du es abschauen.

Deine eigenen Programme bleiben liegen. Erneuert werden nur `toolkit` und `lib` – genau die beiden Ordner, die du sowieso in Ruhe lassen sollst.

Nur wenn du das Kästchen `main.py` *durch das erste Beispiel ersetzen* ankreuzt, wird auch dein `main.py` überschrieben. Das Kästchen ist leer, solange du es nicht selbst anklickst.

MicroPython neu aufspielen

Ganz unten auf dieser Seite steht noch ein Link: **MicroPython aufspielen**. Das ist die Schicht darunter – das Programm, mit dem deine Platine überhaupt erst Python versteht. Das brauchst du nur in zwei Fällen:

- bei einer **fabrikneuen** Platine, auf der noch gar nichts ist,
- oder wenn deine Platine **gar nicht mehr antwortet** und auch *Ausführung unterbrechen* nichts mehr hilft.

Dafür muss der Mikrocontroller von der Platine herunter. Er steckt nur auf zwei Reihen Löchern. Zieh ihn **gerade nach oben** ab – nicht schräg, sonst verbiegen sich die Beinchen. Deine

Kabel dürfen alle stecken bleiben, die brauchst du gleich wieder.

Das USB-Kabel kommt dann an den Mikrocontroller allein, ohne Platine. Erst so lässt sich MicroPython aufspielen.

Danach setzt du ihn wieder auf: Der USB-Anschluss zeigt in dieselbe Richtung wie vorher, und jedes Beinchen muss in sein eigenes Loch. Schau lieber zweimal hin, bevor wieder Strom drankommt.

Dort gibt es ein Kästchen **alles löschen**. Wenn du das ankreuzt, ist die Platine danach wirklich leer – auch deine eigenen Programme sind dann weg. Ohne das Kästchen bleiben sie liegen. Kreuz es nur an, wenn sonst nichts mehr geht: dann ist es das Einzige, was hilft.

A4 · Der Baukasten

Jetzt bist du dran. Such dir ein Projekt aus – oder denk dir was Eigenes aus. Für jedes gibt es einen fertigen Anfang, den du weiterbaust.

Diese Dateien holst du dir selbst. Du bekommst sie über dieselbe Seite, mit der du die Platine aktualisierst (Seite 25): Dort ist ein Kästchen *Baukasten in das Vorlagenverzeichnis kopieren*. Häkchen setzen, aktualisieren – danach liegen sie im Ordner `vorlagen/baukasten/` auf deiner Platine.

Ändere die Vorlagen nicht direkt. Der Ordner `vorlagen/` gehört uns: Beim nächsten Update wird alles darin wieder durch unsere Fassung ersetzt, und deine Änderungen sind weg. Mach es so: Datei öffnen, **unter einem neuen Namen außerhalb von `vorlagen/` speichern** – zum Beispiel als `meine_musikbox.py` –, und dann diese Kopie umbauen. Was du selbst anlegst, fassen wir nie an.

Reaktionsspiel `01_reaktionsspiel.py`

Die grüne LED geht irgendwann an – dann so schnell wie möglich drücken. Wer hat die beste Zeit?

Zahlenschloss `02_zahlenschloss.py`

Dreh den Regler auf die Geheimzahl und drücke. Drei richtige Zahlen, dann geht das Schloss auf.

Musikbox `03_musikbox.py`

Mit dem Regler suchst du ein Lied aus, mit dem Taster spielst du es ab. Das einzige Projekt, das Krach macht.

Dafür brauchst du ein neues Kabel: Summer an Pin 10.

Farbenlampe `04_farbenlampe.py`

Der Regler sucht die Farbe aus, der Taster schaltet um: Farbe, Helligkeit, Regenbogen.

Fernbedienung `05_fernbedienung.py`

Die Platine hört auf eine echte Fernbedienung. Erst musst du herausfinden, welche Nummer welche Taste hat.

Dafür brauchst du ein neues Kabel: Infrarot-Empfänger an Pin 3.

Alles gleichzeitig `06_alles_gleichzeitig.py`

Blinken, Regenbogen und Musik – alles auf einmal, mit `wait=False`.

Viel Spaß beim Weiterbauen!

A5 · Alle Befehle

Vier Regeln, die überall gelten. Alles, was eine Stärke hat, geht von **0 bis 100** – darum passt `led.brightness = dial.value` einfach so zusammen. Jede Aktion darf im Hintergrund laufen, schreib `wait=False` dazu. `sleep(2)` hält dein Programm an – was mit `wait=False` läuft, läuft weiter. Und die Zahl bei `pin=` ist immer genau die Zahl, an der dein Kabel steckt – steckst du es um, änderst du die Zahl.

Licht

```
from toolkit import LED

rot = LED(pin=7)           # rot 7, gelb 6, gruen 5
rot.on(); rot.off()
rot.blink(times=5)
rot.pulse(times=3)         # langsam heller und dunkler
rot.brightness = 40        # wie hell, wenn an: 0 bis 100

# Jedes Licht braucht seinen eigenen Namen - drei zusammen sind eine Ampel:
gelb = LED(pin=6)
gruen = LED(pin=5)
```

Die bunte Lichterkette

```
from toolkit import Pixels

pixels = Pixels(pin=1, count=5)
pixels[0] = "rot"           # ein einzelnes Licht
pixels.fill("blau")         # alle
pixels.set("rot", "gelb", "gruen", "blau", "lila")
pixels.bar(70)              # Balkenanzeige, 0 bis 100
pixels.rainbow(); pixels.chase("rot"); pixels.breathe("blau")
pixels.brightness = 30
pixels.off()
```

Farben: rot gelb gruen blau lila rosa orange tuerkis weiss braun gold aus

Taster

```
from toolkit import Button

taster = Button(pin=4)
taster.is_pressed           # True oder False, gerade jetzt
taster.wait_for_press()    # warten, bis jemand drueckt
taster.wait_for_release()  # warten, bis er wieder losgelassen wird
taster.presses              # wie oft schon gedrueckt wurde
```

Taster: von allein reagieren

Bisher musst du selbst nachschauen oder warten. Es geht auch andersherum: Du gibst der Platine eine Funktion, und sie ruft sie bei jedem Druck auf, während dein Programm etwas anderes tut.

```
from toolkit import Button, LED, forever

taster = Button(pin=4)
rot = LED(pin=7)

def pulsieren():                # deine eigene Funktion ...
    rot.pulse(times=2)

taster.when_pressed = pulsieren # ... ihr Name kommt hier hin
forever()                      # offen halten, sonst ist sofort Schluss
```

Zwei Fallen: Der **Name ohne Klammern** – mit Klammern würdest du die Funktion sofort selbst ausführen. Und ohne `forever()` wäre das Programm zu Ende, bevor du überhaupt drücken kannst. Genauso funktioniert `dial.when_changed` beim Drehregler.

Drehregler

```
from toolkit import Dial

dial = Dial(pin=0)                # nur Pin 0 bis 4 koennen messen
dial.value                        # 0 bis 100
dial.between(0, 255)             # in einen anderen Bereich umrechnen
dial.when_changed = zeigen
```

Summer

```
from toolkit import Buzzer

summer = Buzzer(pin=10)           # nur wenn du ihn angesteckt hast
summer.beep(times=3)
summer.play("C4")                 # ein Ton
summer.tone(880)                  # oder direkt eine Frequenz in Hertz
summer.melody("happy")            # eine fertige Melodie
summer.melody("C4 E4 G4")         # oder eigene Noten
summer.volume = 25                # 0 bis 100
summer.silence()                  # sofort still
```

Fertige Melodien: `happy` `alle_meine_entchen` `haenschen_klein` `fanfare` `alarm`

Vibrationsmotor

```
from toolkit import Vibration

motor = Vibration(pin=8)
motor.buzz(times=2)
motor.buzz(on=0.1, off=0.4)  # wie lang an, wie lang aus
motor.on(); motor.off()
motor.strength = 60          # 0 bis 100
```

Bildschirm: Text

```
from toolkit import Screen

screen = Screen(scl=20, sda=21)
screen.title("Wetter")          # Ueberschrift mit Trennlinie
screen.line(0, "oben")          # eine bestimmte Zeile (0 bis 5)
screen.write("langer Text")     # ab Zeile 0, bricht selbst um
screen.big("23.5")              # gross, gut fuer eine Zahl
screen.clear()                  # Speicher leeren
screen.show()                   # ERST JETZT ist etwas zu sehen
```

Nichts ist zu sehen, bevor `show()` kommt – male erst das ganze Bild. Danach ist der Speicher wieder leer; `screen.show(clear=False)` laesst das Bild stehen und malt weiter darauf.

Bildschirm: Zeichnen

```
# x geht von 0 (links) bis 127 (rechts), y von 0 (oben) bis 63 (unten)
screen.width          # 128
screen.height         # 64
screen.middle         # (64, 32) - genau die Mitte

screen.dot(3, 4)      # ein Punkt
screen.stroke(0, 0, 127, 63) # Linie von Punkt zu Punkt
screen.rect(30, 30, 60, 30) # x, y = linke OBERE Ecke
screen.circle(64, 32, 20)  # x, y = die MITTE
screen.oval(64, 32, 30, 10) # zwei Radien statt einem
screen.triangle(25, 30, 60, 8, 95, 30) # drei Ecken
screen.polygon([(0, 0), (20, 0), (10, 15)]) # beliebig viele Ecken
screen.text(5, 40, "Hallo", size=2) # Text irgendwohin, size 1 bis 3
screen.erase(10, 10, 20, 20) # ein Stueck wegradieren
screen.show()
```

Flaechen kannst du ausmalen: `screen.circle(64, 32, 20, fill=True)`. Geht bei `rect`, `circle`, `oval`, `triangle` und `polygon`.

Die gezeichnete Linie heisst `stroke()`, nicht `line()` – `line()` ist schon die Textzeile.

Wetter

```
from toolkit import Weather

weather = Weather(scl=20, sda=21)
weather.temperature      # Grad
weather.humidity         # Prozent
weather.pressure         # hPa
weather.altitude         # Meter ueber dem Meer
```

WLAN

```
from toolkit import Wireless

wlan = Wireless()          # noch ist nichts an
wlan.on()                  # jetzt geht der Sender an
wlan.ip; wlan.signal; wlan.is_connected
wlan.networks()           # was in der Naehel ist
wlan.off()
```

Mehreres gleichzeitig

Normalerweise wartet jede Zeile, bis sie fertig ist: `rot.blink(times=20)` hält dein Programm auf, bis zwanzigmal geblinkt wurde. Mit `wait=False` läuft die Aktion im Hintergrund weiter, und dein Programm ist sofort in der nächsten Zeile.

```
from toolkit import LED, Pixels, forever

rot = LED(pin=7)
pixels = Pixels(pin=1, count=5)

rot.blink(times=20, wait=False)    # laeuft los und kommt sofort zurueck
pixels.rainbow(times=5, wait=False) # laeuft daneben her

print("Beide laufen schon!")      # das siehst du sofort
forever()                         # offen halten, sonst ist gleich Schluss
```

Das geht bei allem, was eine Weile dauert: `blink`, `pulse`, `rainbow`, `chase`, `breathe`, `buzz` und `melody`. Und wie beim Taster gilt: ohne `forever()` – oder ein `sleep()` – wäre das Programm zu Ende, und die Hintergrundaktionen mit ihm.

Helfer

```
from toolkit import sleep, forever, report_pins, check_i2c, hilfe

sleep(2)                  # anhalten; Hintergrundaktionen laufen weiter
forever()                 # Programm offen halten
report_pins()             # welcher Pin gerade belegt ist
check_i2c(scl=20, sda=21) # was haengt am I2C-Bus?
hilfe()                  # was kann die Platine?
```

A6 · Änderungshistorie und Errata

Welche Fassung du in der Hand hältst, steht unten auf jeder Seite. Hier steht, was sich seitdem geändert hat – und ob in deinem Heft oder auf deiner Platine ein Fehler bekannt ist.

Version 2 · 23.09.2026

Die Helligkeit einer LED ist jetzt ein Dimmer: `led.brightness` stellt nur ein, wie hell sie ist, `led.on()` schaltet sie ein. Einheit 03 und 04 haben dafür je ein `led.on()` dazubekommen. Dasselbe gilt für `motor.strength`.

ERRATA

Bisher ist kein Fehler bekannt.

Version 1 · 02.09.2026

Erste veröffentlichte Fassung.

ERRATA

Ab Software-Fassung 3 auf der Platine bleibt die LED in Einheit 03 und 04 dunkel. Schreib vor das erste `led.brightness` ein `led.on()`.

Bekannter Fehler in der Platine

Steht auf deiner Platine „Lernplatine, 2026-08-18, 1.0“? Dann leuchtet die rote LED mit, sobald der Vibrationsmotor läuft. Das kommt von der Platine selbst und lässt sich mit keinem Programm abstellen.

Dein Programm ist also in Ordnung, auch wenn es nicht so aussieht.