TURN ON YOUR CREATIVITY

FRANZIS
RASPBERRY PI
MAKER KIT

HANDBUCH

Gültig für alle aktuellen Modelle

Updates abrufbar unter www.buch.cd

FRANZIS

Franzis Raspberry Pi Maker Kit

Bibliografische Information der Deutschen Bibliothek

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte Daten sind im Internet über http://dnb.ddb.de abrufbar.

Alle in diesem Buch vorgestellten Schaltungen und Programme wurden mit der größtmöglichen Sorgfalt entwickelt, geprüft und getestet. Trotzdem können Fehler im Buch und in der Software nicht vollständig ausgeschlossen werden. Verlag und Autor haften in Fällen des Vorsatzes oder der groben Fahrlässigkeit nach den gesetzlichen Bestimmungen. Im Übrigen haften Verlag und Autor nur nach dem Produkthaftungsgesetz wegen der Verletzung des Lebens, des Körpers oder der Gesundheit oder wegen der schuldhaften Verletzung wesentlicher Vertragspflichten. Der Schadensersatzanspruch für die Verletzung wesentlicher Vertragspflichten ist auf den vertragstypischen, vorhersehbaren Schaden begrenzt, soweit nicht ein Fall der zwingenden Haftung nach dem Produkthaftungsgesetz gegeben ist.

All circuits and programs presented in this book have been created, checked and tested with great care. Nevertheless, mistakes in the book and the software can not be precluded entirely. Publishing house and author are liable in accordance with existing laws in cases of deliberate intention or wanton negligence. Otherwise, publishing house and author are liable exclusively in accordance with the Product Liability Act with regards to injury of life, body and health or the culpable neglegence of essential contractual obligations.
Damage claims for neglegence of essential contractual obligations are limited to forseeable damage typical for the contract, with the exception of obligatory liability in accordance with the Product Liability Act.

Warnung! Augenschutz und LEDs:

Blicken Sie nicht aus geringer Entfernung direkt in eine LED, denn ein direkter Blick kann Netzhautschäden verursachen! Dies gilt besonders für helle LEDs im klaren Gehäuse sowie in besonderem Maße für Power-LEDs. Bei weißen, blauen, violetten und ultravioletten LEDs gibt die scheinbare Helligkeit einen falschen Eindruck von der tatsächlichen Gefahr für Ihre Augen. Besondere Vorsicht ist bei der Verwendung von Sammellinsen geboten. Betreiben Sie die LEDs so wie in der Anleitung vorgesehen, nicht aber mit größeren Strömen.

Warnung! Augenschutz und LEDs:

Blicken Sie nicht aus geringer Entfernung direkt in eine LED,

Liebe Kunden!

Dieses Produkt wurde in Übereinstimmung mit den geltenden europäischen Richtlinien hergestellt und trägt daher das CE-Zeichen. Der bestimmungsgemäße Gebrauch ist in der beiliegenden Anleitung beschrieben.

Bei jeder anderen Nutzung oder Veränderung des Produktes sind allein Sie für die Einhaltung der geltenden Regeln verantwortlich. Bauen Sie die Schaltungen deshalb genau so auf, wie es in der Anleitung beschrieben wird.
Das Produkt darf nur zusammen mit dieser Anleitung weitergegeben werden.

Das Symbol der durchkreuzten Mülltonne bedeutet, dass dieses Produkt getrennt vom Hausmüll als Elektroschrott dem Recycling zugeführt werden muss. Wo Sie die nächstgelegene kostenlose Annahmestelle finden, sagt Ihnen Ihre kommunale Verwaltung.

Dear Customers!

This product has been manufactured in compliance with existing European guidelines and is thus CE-certified. The intended usage is described in the included manual.

If you choose to alter the product or use it in any other way, you are exclusively responsible for the compliance with existing guidelines. For this reason, wire the circuits exactly as described in the manual. The product may only be passed on together with this manual.

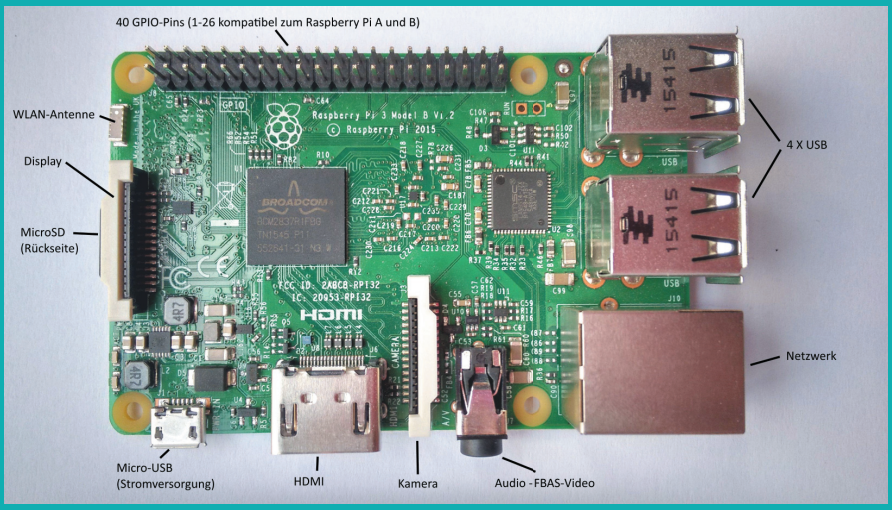
The symbol of the crossed waste container means that this product must be recycled as electronic scrap, separately from domestic waste. For information on the nearest free point of acceptance, please contact your municipal administration.

Autor/Author: Christian Immler
Art & Design: www.ideehoch2.de

INHALT

BEVOR ES LOS-GEHT...

Kaum ein elektronisches Gerät in seiner Preisklasse hat in den letzten Monaten so viel von sich reden gemacht wie der Raspberry Pi. Der Raspberry Pi ist, auch wenn es auf den ersten Blick gar nicht so aussieht, ein vollwertiger Computer etwa in der Größe einer Kreditkarte – und vor allem zu einem sehr günstigen Preis. Nicht nur die Hardware ist günstig, die Software noch mehr: Das Betriebssystem und alle im Alltag notwendigen Anwendungen werden kostenlos zum Download angeboten.



Die Anschlüsse des Raspberry Pi 3

Seit Sommer 2014 haben die neuen Raspberry-Pi-Modelle die abgebildete Bauform. Dies gilt für die Modelle Raspberry Pi B+, Pi2 und Pi3. Alle diese Modelle, sowie auch der Raspberry Pi A+ mit nur einem USB-Anschluss und ohne LAN, verwenden die erweiterte GPIO-Schnittstelle mit 40 Pins. Die älteren Modelle Raspberry Pi A und B aus der Anfangszeit sind heute nur noch mit Einschränkungen kompatibel.

Mit dem speziell angepassten Linux mit grafischer Oberfläche ist der Raspberry Pi ein stromsparender, lautloser PC-Ersatz. Seine frei programmierbare GPIO-Schnittstelle macht den Raspberry Pi besonders interessant für Hardwarebastler und die neue Maker-Szene.

Der Name Raspberry Pi

Raspberry ist das englische Wort für Himbeere. Schon früher wurden Computer nach Früchten benannt, wie z. B. Apple, Apricot, Blackberry. *Pi* steht für Python Interpreter, eine wichtige Programmiersprache auf dem Raspberry Pi. Zusammen ergibt sich ein Name, der wie das englische Wort für Himbeerkuchen *raspberry pie,* klingt.

Was braucht man?

Wenn Sie diesen Text lesen, haben Sie sich sicher schon etwas mit dem Raspberry Pi beschäftigt. Deshalb fassen wir die Systemvoraussetzungen für das MakerKit nur kurz zusammen.

Raspberry Pi

Natürlich brauchen Sie einen Raspberry Pi, und zwar mindestens das Modell B+, besser den aktuellen Raspberry Pi 3. Das Modell A+ hat nur weniger Speicher und keinen Netzwerkanschluss, funktioniert für viele der Experimente aber ebenfalls.

Micro-USB-Handyladegerät

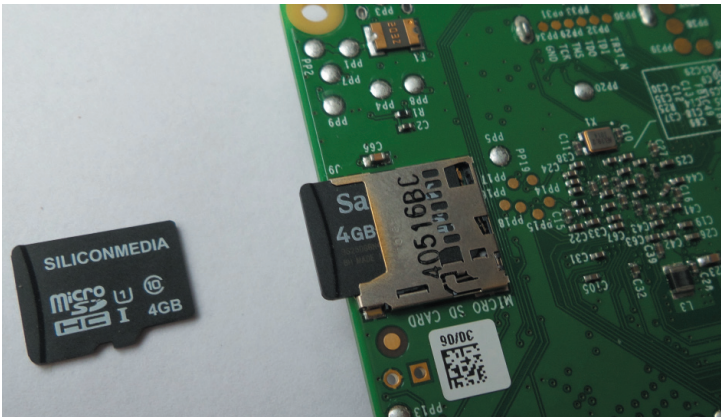
Für den Raspberry Pi reicht jedes moderne Handynetzteil. Das Netzteil muss 5 V und mindestens 1.500 mA liefern, besonders beim Raspberry Pi 3 sollten es besser 2.500 mA sein. Ältere Ladegeräte aus den Anfangszeiten der USB-Ladetechnik sind noch zu schwach. Schließt man leistungshungrige USB-Geräte wie externe Festplatten ohne eigene Stromversorgung an, ist ein stärkeres Netzteil erforderlich.

So äußert sich ein zu schwaches Netzteil

Wenn der Raspberry Pi bootet, sich dann aber die Maus nicht bewegen lässt oder das System nicht auf Tastatureingaben reagiert, deutet dies auf eine zu schwache Stromversorgung hin. Auch wenn der Zugriff auf angeschlossene USB-Sticks oder Festplatten nicht möglich ist, sollten Sie ein stärkeres Netzteil verwenden.

Speicherkarte

Die Speicherkarte dient im Raspberry Pi als Festplatte. Sie enthält das Betriebssystem. Eigene Daten und installierte Programme werden ebenfalls darauf gespeichert. Die Speicherkarte sollte mindestens 4 GByte groß sein und nach Herstellerangaben des Raspberry Pi mindestens den Class-4-Standard unterstützen. Dieser Standard gibt die Geschwindigkeit der Speicherkarte an. Eine aktuelle Class-10-Speicherkarte macht sich in der Performance deutlich bemerkbar. Alle aktuellen Raspberry-Pi-Modelle verwenden die aus Smartphones bekannten MicroSD-Speicherkarten.



Der MicroSD-Speicherkartensteckplatz auf der Rückseite des Raspberry Pi. Die Zahl im Kreis gibt die Klasse der Speicherkarte an.

Tastatur und Maus

Jede gängige Tastatur mit USB-Anschluss kann genutzt werden. Kabellose Tastaturen funktionieren manchmal nicht, da sie zu viel Strom oder spezielle Treiber benötigen.

Einige USB-Tastaturen besitzen einen weiteren USB-Anschluss für die Maus. Dadurch sparen Sie sich am Raspberry Pi einen USB-Anschluss, was besonders beim Raspberry Pi A+ wichtig ist, der nur einen USB-Anschluss hat. Die anderen aktuellen Raspberry-Pi-Modelle haben vier USB-Anschlüsse, die immer ausreichen sollten.

Netzwerkkabel

Zur Verbindung mit dem Router im lokalen Netzwerk wird ein Netzwerkkabel benötigt. Der Raspberry Pi 3 hat ein eingebautes WLAN-Modul.

Hier ist kein Netzwerkkabel zur Internetverbindung nötig. An die Modelle Raspberry Pi B+ und Pi 2 kann ein WLAN-USB-Stick angeschlossen werden. Zur Ersteinrichtung ist hier noch ein Netzwerkkabel erforderlich, beim Raspberry Pi 3 nicht. Ohne Internetzugang sind viele Funktionen des Raspberry Pi nicht sinnvoll nutzbar.

HDMI-Kabel

Der Raspberry Pi kann per HDMI-Kabel an einem Monitor oder an einem Fernseher angeschlossen werden. Zum Anschluss an Computermonitore mit DVI-Anschluss gibt es spezielle HDMI-Adapter. HDMI-Kabel sind im Elektronikhandel zu Preisen erhältlich, die fast dem Preis des Raspberry Pi selbst entsprechen. Bei Onlineversendern (z. B. `amzn.to/VGv05j`) bekommt man sie einschließlich Versand für wenige Euro – Unterschied: für die Verwendung am Raspberry Pi außer dem Preis keiner.

Audiokabel

Über ein Audiokabel mit 3,5-mm-Klinkensteckern können Kopfhörer oder PC-Lautsprecher am Raspberry Pi genutzt werden. Das Audiosignal ist auch über das HDMI-Kabel verfügbar. Bei HDMI-Fernsehern oder Monitoren ist kein Audiokabel nötig. Wird ein PC-Monitor über ein HDMI-Kabel mit DVI-Adapter angeschlossen, geht meist an dieser Stelle das Audiosignal verloren, sodass Sie den analogen Audioausgang wieder brauchen.

Raspbian-Betriebssystem

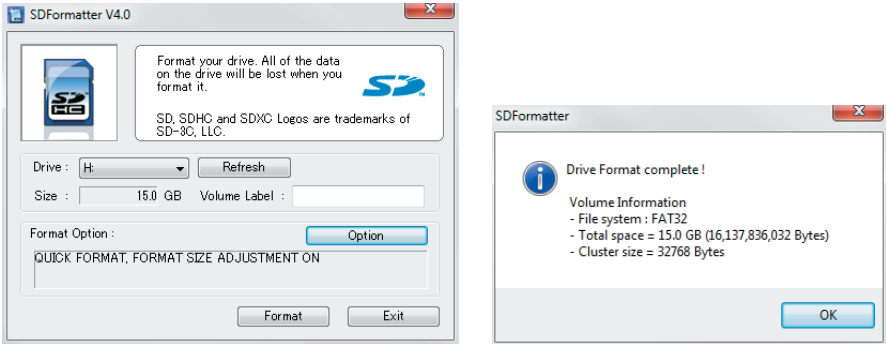
Wir gehen bei allen Projekten davon aus, dass die aktuellste Version des Raspbian-Jessie-Betriebssystems auf der Speicherkarte installiert ist. Ältere Versionen sind mit den aktuellen Raspberry-Pi-Modellen nur eingeschränkt kompatibel und bieten auch keine GPIO-Unterstützung für die Programmiersprache Scratch.

Speicherkarte im PC vorbereiten

Da der Raspberry Pi selbst noch nicht booten kann, bereiten wir die Speicherkarte auf dem PC vor. Dazu brauchen Sie einen Kartenleser am PC. Dieser kann fest eingebaut oder per USB angeschlossen werden.

Verwenden Sie am besten fabrikneue Speicherkarten, da diese vom Hersteller bereits optimal vorformatiert sind. Sie können aber auch eine Speicherkarte verwenden, die vorher bereits in einer Digitalkamera oder

einem Smartphone genutzt wurde. Diese Speicherkarten sollten vor der Verwendung für den Raspberry Pi neu formatiert werden. Theoretisch können Sie dazu die Formatierungsfunktionen von Windows verwenden. Deutlich besser ist die Software SDFormatter der SD Association. Damit werden die Speicherkarten für optimale Performance formatiert. Dieses Tool können Sie sich bei *www.sdcard.org/downloads/formatter_4* kostenlos herunterladen.



Das SDFormatter-Tool unter Windows in Aktion.

Sollte die Speicherkarte Partitionen aus einer früheren Betriebssysteminstallation für den Raspberry Pi enthalten, wird im SDFormatter nicht die vollständige Größe angezeigt. Verwenden Sie in diesem Fall die Formatierungsoption *FULL (Erase)* und schalten Sie die Option *Format Size Adjustment* ein. Damit wird die Partitionierung der Speicherkarte neu angelegt.

Speicherkarte wird gelöscht

Am besten verwenden Sie eine leere Speicherkarte für die Installation des Betriebssystems. Sollten sich auf der Speicherkarte Daten befinden, werden diese durch die Neuformatierung während der Betriebssysteminstallation unwiderruflich gelöscht.

Der Software-Installer NOOBS

New Out Of Box Software (NOOBS) ist ein besonders einfacher Installer für Raspberry-Pi-Betriebssysteme. Hier braucht sich der Benutzer nicht mehr wie früher selbst mit Image-Tools und Bootblöcken auseinanderzusetzen, um eine bootfähige Speicherkarte einzurichten.

NOOBS bietet verschiedene Betriebssysteme zur Auswahl, wobei man beim ersten Start direkt auf dem Raspberry Pi das gewünschte Betriebssystem auswählen kann, das dann bootfähig auf der Speicherkarte

installiert wird. Laden Sie sich das etwa 1,2 GByte große Installationsarchiv für NOOBS auf der offiziellen Downloadseite *www.raspberrypi.org/downloads* herunter und entpacken Sie es am PC auf eine Speicherkarte. Die Experimente in diesem Lernpaket wurden mit der NOOBS-Version 1.8.0 getestet.

Starten Sie jetzt den Raspberry Pi mit dieser Speicherkarte. Stecken Sie sie dazu in den Steckplatz des Raspberry Pi und schließen Sie Tastatur, Maus, Monitor und Netzwerkkabel an. Der USB-Stromanschluss kommt als Letztes. Damit wird der Raspberry Pi eingeschaltet. Einen separaten Einschaltknopf gibt es nicht.

Nach wenigen Sekunden erscheint ein Auswahlmenü, in dem Sie das gewünschte Betriebssystem wählen können. Wir verwenden das von der Raspberry-Pi-Stiftung empfohlene Betriebssystem Raspbian.

Wählen Sie ganz unten Deutsch als Installationssprache aus und markieren Sie das vorausgewählte Raspbian-Betriebssystem. Nach Bestätigen einer Sicherheitsmeldung, dass die Speicherkarte überschrieben wird, startet die Installation, die einige Minuten dauert. Während der Installation werden kurze Informationen zu Raspbian angezeigt.

Der erste Start auf dem Raspberry Pi

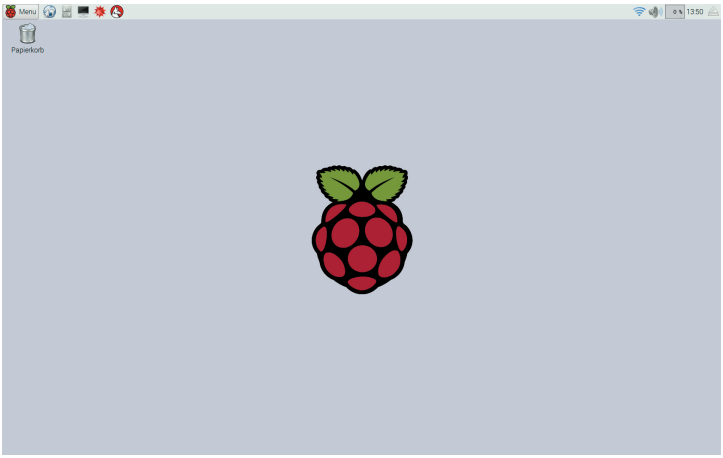
Nach abgeschlossener Installation bootet der Raspberry Pi neu und startet automatisch den grafischen Desktop LXDE. Die deutsche Sprache und die deutsche Tastaturbelegung sind wie auch weitere wichtige Grundeinstellungen bereits automatisch gewählt.

Fast wie Windows – die grafische Oberfläche LXDE

Viele schrecken bei dem Wort „Linux" erst einmal zurück, weil sie befürchten, kryptische Befehlsfolgen per Kommandozeile eingeben zu müssen wie vor 30 Jahren unter DOS. Weit gefehlt! Linux bietet als offenes Betriebssystem den Entwicklern freie Möglichkeiten, eigene grafische Oberflächen zu entwickeln. So ist man als Anwender des im Kern immer noch kommandozeilenorientierten Betriebssystems nicht auf eine Oberfläche festgelegt.

Das Raspbian-Linux für den Raspberry Pi verwendet die Oberfläche LXDE (Lightweight X11 Desktop Environment), die einerseits sehr wenig Sys-

temressourcen benötigt und andererseits mit ihrem Startmenü und dem Dateimanager der gewohnten Windows-Oberfläche sehr ähnelt.



Der LXDE-Desktop auf dem Raspberry Pi

Linux-Anmeldung

Selbst die bei Linux typische Benutzeranmeldung wird im Hintergrund erledigt. Falls Sie es doch einmal brauchen: Der Benutzername lautet `pi` und das Passwort `raspberry`.

Das Menüsymbol links oben öffnet das Startmenü, die Symbole daneben den Dateimanager und den Webbrowser. Das Startmenü ist wie unter Windows mehrstufig aufgebaut. Häufig verwendete Programme lassen sich mit einem Rechtsklick auf dem Desktop ablegen.

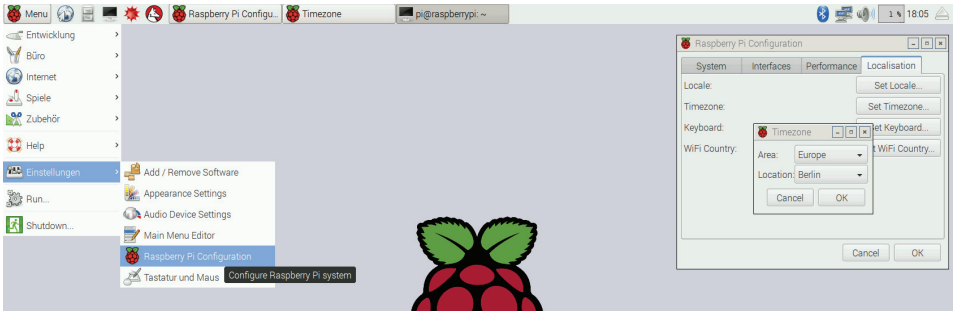
Raspberry Pi ausschalten

Theoretisch kann man bei dem Raspberry Pi einfach den Stecker ziehen und er schaltet sich ab. Besser ist es jedoch, ihn wie einen PC sauber herunterzufahren. Klicken Sie dazu auf dem Desktop doppelt auf das *Shutdown*-Symbol.

Uhrzeit einstellen

Der Raspberry Pi hat keine interne Echtzeituhr, sondern holt sich die aktuelle Uhrzeit aus dem Internet. Aber auch bei aktiver Internetverbindung wird die Uhr oben rechts in der Taskleiste zunächst eine falsche Zeit anzeigen. Das liegt an der standardmäßig eingestellten Zeitzone.

Wählen Sie im Menü *Einstellungen/Raspberry Pi Configuration*. Dieses Dialogfeld ersetzt das textbasierte Konfigurationstool früherer Raspbian-Versionen. Klicken Sie auf der Registerkarte *Localisation* auf den Button *Set Timezone* und wählen Sie die hierzulande verwendete Zeitzone *Europa/Berlin*.



Zeitzone einstellen

Haben Sie keine Internetverbindung, zeigt der Raspberry Pi eine ungültige Uhrzeit an. Sie können in solchen Fällen über das Symbol mit dem schwarzen Bildschirm in der Taskleiste ein Kommandozeilenfenster öffnen und die richtige Zeit einstellen, z. B.:

```
sudo date -s "2016-03-22 08:00:00 CET"
```

Die Einstellung wird anschließend mit einer Klartextanzeige von Datum und Uhrzeit quittiert:

```
Di 22. Mär 08:00:00 CET 2016
```

Diese Zeiteinstellung gilt nur bis zum nächsten Neustart. Es gibt keine batteriegepufferte Uhr.

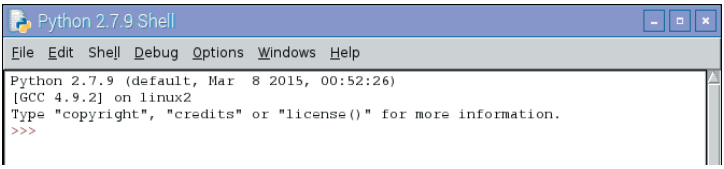
Das erste Programm mit Python

Zum Einstieg in die Programmierung ist auf dem Raspberry Pi die Programmiersprache Python vorinstalliert. Python überzeugt durch seine klare Struktur, die einen einfachen Einstieg in das Programmieren erlaubt, ist aber auch eine ideale Sprache, um „mal schnell" etwas zu automatisieren, was man sonst von Hand erledigen würde. Da keine Variablendeklarationen, Typen, Klassen oder komplizierte Regeln zu beachten sind, macht das Programmieren wirklich Spaß.

Python 2.7.9 wird über den Menüpunkt *Entwicklung/Python 2 (IDLE)* gestartet. Hier erscheint ein auf den ersten Blick simples Eingabefenster mit einem Befehlsprompt.

Python 2 oder Python 3?

Auf dem Raspberry Pi sind gleich zwei Versionen von Python vorinstalliert. Leider verwendet die neueste Python-Version 3.x teilweise eine andere Syntax als die bewährte Version 2.x, sodass Programme aus der einen Version nicht mit der anderen laufen. Einige wichtige Bibliotheken sind noch nicht für Python 3.x verfügbar. Deshalb, und weil auch die meisten im Internet verfügbaren Programme für Python 2.x geschrieben wurden, verwenden wir in diesem Lernpaket die bewährte Python-Version 2.7.9. Sollte auf Ihrem Raspberry Pi eine ältere Python-Version mit einer Versionsnummer 2.x installiert sein, funktionieren unsere Beispiele gleichermaßen damit.



Das Eingabefenster der Python-Shell

In diesem Fenster öffnen Sie vorhandene Python-Programme, schreiben neue oder können auch direkt Python-Kommandos interaktiv abarbeiten, ohne ein eigentliches Programm schreiben zu müssen. Geben Sie z. B. am Prompt Folgendes ein:

```
>>> 1+2
```

Es erscheint sofort die richtige Antwort:

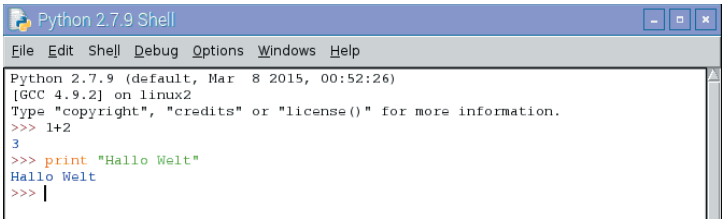
```
3
```

Auf diese Weise lässt sich Python als komfortabler Taschenrechner verwenden, was aber noch nichts mit Programmierung zu tun hat.

Üblicherweise fangen Programmierkurse mit einem *Hallo Welt*-Programm an, das auf den Bildschirm den Satz „Hallo Welt" schreibt. Das ist in Python derart einfach, dass es sich nicht einmal lohnt, dafür eine eigene Überschrift einzufügen. Tippen Sie im Python-Shell-Fenster einfach folgende Zeile:

```
>>> print "Hallo Welt"
```

Dieses erste „Programm" schreibt dann `Hallo Welt` in die nächste Zeile auf dem Bildschirm.



Hallo Welt in Python (oberhalb ist noch die Ausgabe der Berechnung zu sehen).

Hier sehen Sie auch gleich, dass die Python-Shell zur Verdeutlichung automatisch verschiedene Textfarben verwendet. Python-Kommandos sind orange, Zeichenketten grün und Ergebnisse blau. Später werden Sie noch weitere Farben entdecken.

Python-Flashcards

Python ist die ideale Programmiersprache, um den Einstieg in die Programmierung zu erlernen. Nur die Syntax und die Layoutregeln sind etwas gewöhnungsbedürftig. Zur Hilfestellung im Programmieralltag werden die wichtigsten Syntaxelemente der Sprache Python in Form kleiner „Spickzettel" kurz beschrieben. Diese basieren auf den Python-Flashcards von David Whale. Was es damit genau auf sich hat, finden Sie bei *bit.ly/pythonflashcards*. Diese Flashcards erklären nicht die technischen Hintergründe, sondern beschreiben nur anhand ganz kurzer Beispiele die Syntax, also wie etwas gemacht wird.

BEDINGUNGEN 8

```
a=1
if a==1:
     print "gleich"
if a!=1:
     print "nicht gleich"
if a<1:
     print "kleiner"
if a>1:
     print "größer"
if a<=1:
     print "kleiner oder gleich"
if a>=1:
     print "größer oder gleich"
```

python(1) V2 (deutsch) - softwarehandbuch.de

IF ELSE 9

```
alter=10
if alter>17:
     print "Du darfst Auto fahren"
else:
     print "Du bist nicht alt genug"
```

python(1) V2 (deutsch) - softwarehandbuch.de

IF ELIF ELSE 10

```
alter=10
if alter<4:
     print "Du bist in der Kinderkrippe"
elif alter<6:
     print "Du bist im Kindergarten"
elif alter<10:
     print "Du bist in der Grundschule"
elif alter<19:
     print "Du bist im Gymnasium"
else:
     print "Du hast die Schule verlassen"
```

python(1) V2 (deutsch) - softwarehandbuch.de

AND/OR BEDINGUNGEN 11

```
a=1
b=2
if a>0 and b>0:
     print "Beide sind nicht Null"

if a>0 or b>0:
     print "Mindestens eine ist nicht Null"
```

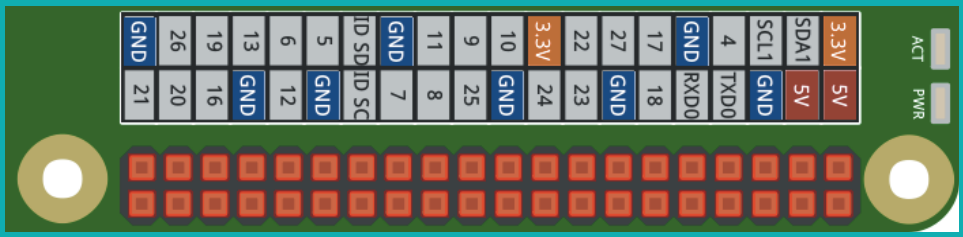
python(1) V2 (deutsch) - softwarehandbuch.de

Ausschnitt aus den Python-Flashcards

1 DIE ERSTE LED LEUCHTET AM RASPBERRY PI

Die 40-polige Stiftleiste in der Ecke des Raspberry Pi bietet die Möglichkeit, direkt Hardware anzuschließen, um z. B. über Taster Eingaben zu machen oder programmgesteuert LEDs leuchten zu lassen. Diese Stiftleiste wird als GPIO bezeichnet. Die englische Abkürzung „General Purpose Input Output" bedeutet auf Deutsch einfach „Allgemeine Ein- und Ausgabe".

Von diesen 40 Pins lassen sich die 22, die nur mit Nummern bezeichnet sind, wahlweise als Eingang oder Ausgang programmieren und so für vielfältige Hardwareerweiterungen nutzen. Die übrigen sind für die Stromversorgung und andere Zwecke fest eingerichtet.



Belegung der GPIO-Schnittstelle. Die abgerundete Ecke unten rechts entspricht der Ecke der Raspberry-Pi-Platine.

Vorsicht

Verbinden Sie auf keinen Fall irgendwelche GPIO-Pins miteinander und warten ab, was passiert, sondern beachten Sie unbedingt folgende Hinweise:

Einige GPIO-Pins sind direkt mit Anschlüssen des Prozessors verbunden, ein Kurzschluss kann den Raspberry Pi komplett zerstören. Verbindet man über einen Schalter oder eine LED zwei Pins miteinander, muss immer ein Vorwiderstand dazwischengeschaltet werden.

Verwenden Sie für Logiksignale immer Pin 1, der +3,3 V liefert und bis 50 mA belastet werden kann. Alle mit *GND* bezeichneten Pins sind Masseleitungen für Logiksignale.

Jeder GPIO-Pin kann als Ausgang (z. B. für LEDs) oder als Eingang (z. B. für Taster) geschaltet werden. GPIO-Ausgänge liefern im Logikzustand *1* eine Spannung von +3,3 V, im Logikzustand *0* 0 Volt. GPIO-Eingänge liefern bei einer Spannung bis +1,7 V das Logiksignal *0*, bei einer Spannung zwischen +1,7 V und +3,3 V das Logiksignal *1*.

Pin 2 und Pin 4 liefern +5 V zur Stromversorgung externer Hardware. Hier kann so viel Strom entnommen werden, wie das USB-Netzteil des Raspberry Pi liefert. Diese Pins dürfen aber nicht mit einem GPIO-Eingang verbunden werden.

1.1 | Bauteile im Paket

Das Paket enthält diverse elektronische Bauteile, mit denen sich die beschriebenen Experimente (und natürlich auch eigene) aufbauen lassen. An dieser Stelle werden die Bauteile nur kurz vorgestellt. Die notwendige praktische Erfahrung im Umgang damit bringen dann die wirklichen Experimente.

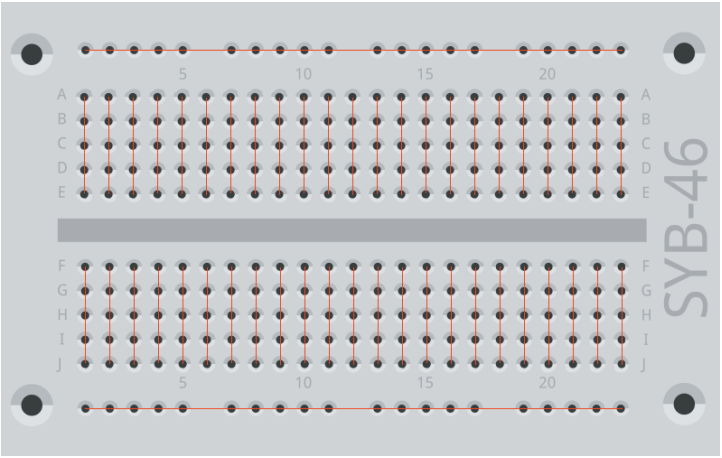
- 3x Steckplatine
- 1x LCD-Anzeige HD44780
- 1x Pfostenverbinder (16-polig)
- 1x Portexpander MCP 23017
- 10x LED
- 4x Taster
- 10x Widerstand 220 Ohm (Rot-Rot-Braun)
- 1x Widerstand 10 kOhm (Braun-Schwarz-Orange)
- 1x Widerstand 560 Ohm (Grün-Blau-Braun)
- 6x Widerstand 20 MOhm (Rot-Schwarz-Blau)
- 1x Potenziometer 15 kOhm
- 12x Verbindungskabel (Steckplatine – Raspberry Pi)

1

- Schaltdraht
- 2x Kabel mit Krokodilklemmen

1.1.1 | Steckplatinen

Für den schnellen Aufbau elektronischer Schaltungen, ohne löten zu müssen, sind drei Steckplatinen im Paket enthalten. Hier können elektronische Bauteile direkt in ein Lochraster mit Standardabständen eingesteckt werden. Bei diesen Platinen sind die äußeren Längsreihen mit Kontakten (X und Y) alle miteinander verbunden. Diese Kontaktreihen werden oft als Plus- und Minuspol zur Stromversorgung der Schaltungen genutzt. In den anderen Kontaktreihen sind jeweils fünf Kontakte (A bis E und F bis J) quer miteinander verbunden, wobei in der Mitte der Platine eine Lücke ist. So können hier in der Mitte größere Bauelemente wie z. B. der Portexpander eingesteckt und nach außen hin verdrahtet werden.



Die Steckplatine aus dem Paket mit ihren eingebauten Verbindungen

1.1.2 | Verbindungskabel

Die farbigen Verbindungskabel haben alle auf einer Seite einen dünnen Drahtstecker, mit dem sie sich auf der Steckplatine einstecken lassen. Auf der anderen Seite befindet sich eine Steckbuchse, die auf einen GPIO-Pin des Raspberry Pi passt.

Weiterhin ist Schaltdraht im Paket enthalten. Damit stellen Sie kurze Verbindungsbrücken her, mit denen Kontaktreihen auf der Steckplati-

ne verbunden werden. Schneiden Sie den Draht mit einem kleinen Seitenschneider je nach Experiment auf die passenden Längen ab. Um die Drähte besser in die Steckplatine stecken zu können, empfiehlt es sich, die Drähte leicht schräg abzuschneiden, sodass eine Art Keil entsteht. Entfernen Sie an beiden Enden auf einer Länge von etwa einem halben Zentimeter die Isolierung.

1.1.3 | Widerstände und ihre Farbcodes

Widerstände werden in der Digitalelektronik im Wesentlichen zur Strombegrenzung an den Ports eines Mikrocontrollers sowie als Vorwiderstände für LEDs verwendet. Die Maßeinheit für Widerstände ist Ohm. 1.000 Ohm entsprechen einem Kiloohm, abgekürzt kOhm, 1.000 kOhm entsprechen einem Megaohm, abgekürzt MOhm.

Die Widerstandswerte werden auf den Widerständen durch farbige Ringe angegeben. Die meisten Widerstände haben vier solcher Farbringe. Die ersten beiden Farbringe bezeichnen die Ziffern, der dritte einen Multiplikator und der vierte die Toleranz. Dieser Toleranzring ist meistens gold- oder silberfarben – das sind Farben, die auf den ersten Ringen nicht vorkommen, sodass die Leserichtung eindeutig ist. Der Toleranzwert selbst spielt in der Digitalelektronik kaum eine Rolle.

Farbe	Widerstandswert in Ohm			
	1. Ring (Zehner)	2. Ring (Einer)	3. Ring (Multiplikator)	4. Ring (Toleranz)
Silber			$10^{-2} = 0{,}01$	±10 %
Gold			$10^{-1} = 0{,}1$	±5 %
Schwarz		0	$10^{0} = 1$	
Braun	1	1	$10^{1} = 10$	±1 %
Rot	2	2	$10^{2} = 100$	±2 %
Orange	3	3	$10^{3} = 1.000$	
Gelb	4	4	$10^{4} = 10.000$	
Grün	5	5	$10^{5} = 100.000$	±0,5 %
Blau	6	6	$10^{6} = 1.000.000$	±0,25 %
Violett	7	7	$10^{7} = 10.000.000$	±0,1 %
Grau	8	8	$10^{8} = 100.000.000$	±0,05 %
Weiß	9	9	$10^{9} = 1.000.000.000$	

Die Tabelle zeigt die Bedeutung der farbigen Ringe auf Widerständen.

Im Paket sind Widerstände in vier verschiedenen Werten enthalten:

Wert	1. Ring (Zehner)	2. Ring (Einer)	3. Ring (Multipl.)	4. Ring (Toleranz)	Verwendung
220 Ohm	Rot	Rot	Braun	Gold	Vorwiderstand für LEDs
560 Ohm	Grün	Blau	Braun	Gold	Vorwiderstand für Stromversorgung der LCD-Anzeige
10 kOhm	Braun	Schwarz	Orange	Gold	Pulldown-Widerstand für GPIO-Eingänge
20 MOhm	Rot	Schwarz	Blau	Gold	Widerstand für Pikey Pikey

Farbcodes der Widerstände im Lernpaket

1.1.4 | LEDs

An die GPIO-Ports können für Lichtsignale und Lichteffekte LEDs (LED = Light Emitting Diode, zu Deutsch Leuchtdiode) angeschlossen werden. Dabei muss zwischen dem verwendeten GPIO-Pin und der Anode der LED ein 220-Ohm-Vorwiderstand (Rot-Rot-Braun) eingebaut werden, um den Durchflussstrom zu begrenzen und damit ein Durchbrennen der LED zu verhindern. Zusätzlich schützt der Vorwiderstand auch den GPIO-Ausgang des Raspberry Pi, da die LED in Durchflussrichtung fast keinen Widerstand bietet und deshalb der GPIO-Port bei Verbindung mit Masse schnell überlastet werden könnte. Die Kathode der LED verbindet man mit der Masseleitung auf Pin 6 des Raspberry Pi.

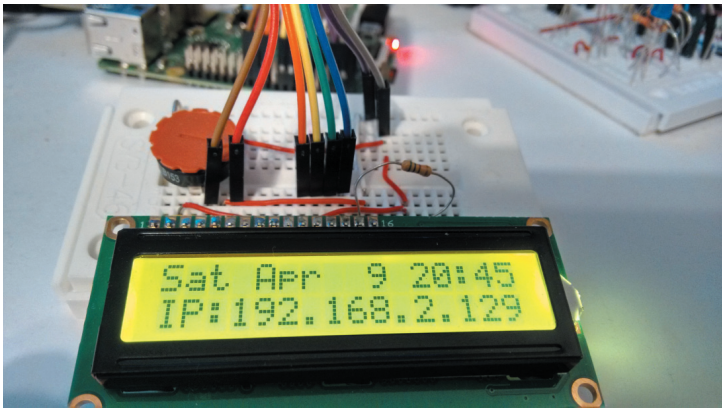
LED in welcher Richtung anschließen?

Die beiden Anschlussdrähte einer LED sind unterschiedlich lang. Der längere von beiden ist der Pluspol, die Anode, der kürzere die Kathode. Einfach zu merken: Das Pluszeichen hat einen Strich mehr als das Minuszeichen und macht damit den Draht etwas länger. Außerdem sind die meisten LEDs auf der Minusseite abgeflacht wie eben ein Minuszeichen. Leicht zu merken: Kathode = kurz = Kante.

1.1.5 | LC-Display

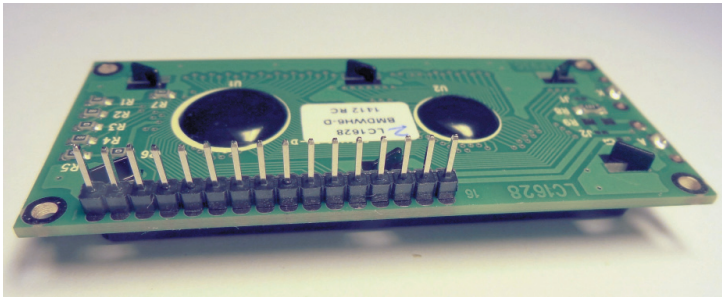
Im Paket ist ein zweizeiliges LC-Display enthalten, das wie die meisten derartigen Displays mit dem Quasi-Standard HD44780 kompatibel ist. Dabei handelt es sich um die Bezeichnung des in den Displays eingebau-

ten Controllers, der Zeichen in ein bis vier Zeilen darstellt, ohne dass sich der Benutzer um die Ansteuerung der einzelnen Pixel zu kümmern braucht.



Zweizeiliges LC-Display auf einer Steckplatine am Raspberry Pi

Das Display hat eine 16-polige Anschlussleiste. Löten Sie hier den mitgelieferten Pfostenverbinder mit den kürzeren Pins so an, dass die längeren Pins nach unten frei herausstehen. Damit wird das Display später auf die Steckplatine gesteckt.



Display mit angelötetem Pfostenverbinder

Löten – einfach und richtig

Wer sich mit Hardwarebasteleien rund um den Raspberry Pi beschäftigt, wird ab und an auch mal etwas löten müssen. Für den Profi ist das kein Problem, für den Anfänger eigentlich auch nicht, wenn er ein paar wichtige Tricks beachtet. *Löten ist einfach* ist ein unterhaltsamer Comic mit Basiswissen für Hobbylöter: *bit.ly/178qobA*.

1

1.2 | GPIO mit Python

Um GPIO-Ports über Python-Programme zu nutzen, muss die Python-GPIO-Bibliothek installiert sein. In ganz alten Raspbian-Versionen musste man diese Bibliothek noch manuell installieren. Dies ist heute nicht mehr nötig, auch der früher benötigte sudo-Zugriff ist inzwischen Geschichte.

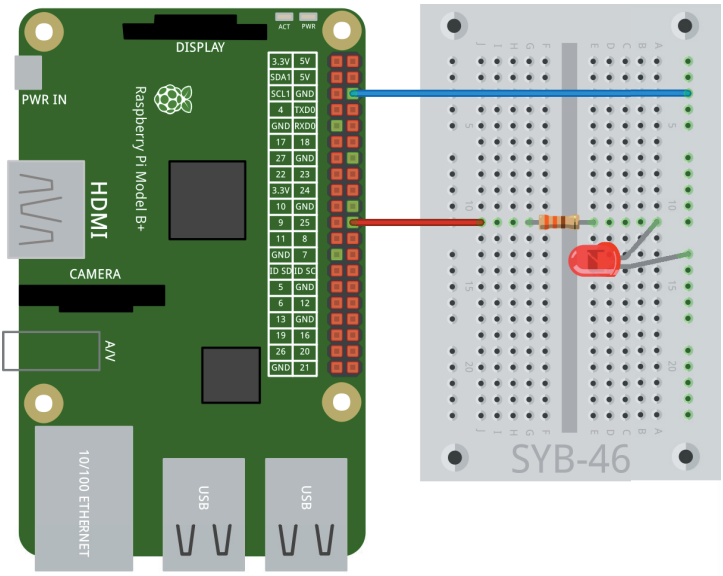
Programm-datei:
led.py

1.3 | LED mit Python ein-/ausschalten

Schließen Sie wie auf dem Bild eine LED über einen 220-Ohm-Vorwiderstand (Rot-Rot-Braun) am GPIO-Port 25 (Pin 22) an und verbinden Sie den Minuspol der LED über die Masseschiene der Steckplatine mit der Masseleitung des Raspberry Pi (Pin 6).

Benötigte Bauteile

1x Steckplatine, 1x LED, 1x 220-Ohm-Widerstand (Rot-Rot-Braun), 2x Verbindungskabel



Eine LED am GPIO-Port 25

Das Programm `led.py` lässt die LED zehnmal blinken:

```
001 #!/usr/bin/python
002 import RPi.GPIO as GPIO
003 import time
004
005 GPIO.setmode(GPIO.BCM)
006 GPIO.setup(25, GPIO.OUT)
007
008 for i in range(10):
009     GPIO.output(25, 1)
010     time.sleep(0.2)
011     GPIO.output(25, 0)
012     time.sleep(0.2)
013 GPIO.cleanup()
```

1.3.1 | So funktioniert es

Das Beispiel zeigt die grundlegenden Funktionen der `RPi.GPIO`-Bibliothek.

```
001 #!/usr/bin/python
```

Diese Zeile steht am Anfang (fast) jedes Python-Skripts, um die Datei als solches zu identifizieren. Sie wäre zwar in diesem Fall nicht unbedingt nötig, aber gewöhnen Sie sich an, sie immer als erste Zeile in ein Python-Skript zu schreiben.

```
002 import RPi.GPIO as GPIO
```

Die Bibliothek `RPi.GPIO` muss in jedes Python-Programm importiert werden, in dem sie genutzt werden soll. Durch diese Schreibweise können alle Funktionen der Bibliothek über das Präfix `GPIO` angesprochen werden.

```
003 import time
```

Die Python-Bibliothek `time` hat nichts mit GPIO-Programmierung zu tun. Sie enthält Funktionen zur Zeit- und Datumsberechnung, unter anderem auch eine Funktion `time.sleep()`, mit der sich auf einfache Weise Wartezeiten realisieren lassen.

```
005 GPIO.setmode(GPIO.BCM)
```

Am Anfang jedes Programms muss definiert werden, wie die GPIO-Ports bezeichnet sind. Üblicherweise verwendet man die Standardnummerierung `BCM`.

1

Nummerierung der GPIO-Ports

Die Bibliothek `RPi.GPIO` unterstützt zwei verschiedene Methoden zur Bezeichnung der Ports. Im Modus `BCM` werden die bekannten GPIO-Portnummern genutzt, die auch auf Kommandozeilenebene oder in Shell-Skripten verwendet werden. Im alternativen Modus `BOARD` entsprechen die Bezeichnungen den Pin-Nummern von 1 bis 26 auf der Raspberry-Pi-Platine. Dieser Modus wird jedoch nur von der Python-Bibliothek unterstützt, aber nicht von Scratch und anderen Programmen.

```
006 GPIO.setup(25, GPIO.OUT)
```

Die Funktion `GPIO.setup()` initialisiert einen GPIO-Port als Ausgang oder als Eingang. Der erste Parameter bezeichnet den Port je nach vorgegebenem Modus `BCM` oder `BOARD` mit seiner GPIO-Nummer oder Pinnummer. Der zweite Parameter kann entweder `GPIO.OUT` für einen Ausgang oder `GPIO.IN` für einen Eingang sein.

```
008 for i in range(10):
```

Jetzt startet eine Schleife, die zehnmal durchläuft. `for`-Schleifen wie diese sind in vielen Programmiersprachen bekannt. Der Zähler `i` nimmt nacheinander, von `0` aufsteigend, ganze Zahlen an. Die Schleife wird beendet, wenn der bei `range` angegebene Wert erreicht ist. Der eingerückte Programmcode wird in jedem Schleifendurchlauf wiederholt.

```
009   GPIO.output(25, 1)
```

Auf dem Port `25` wird eine `1` ausgegeben. Die angeschlossene LED leuchtet. Statt der `1` können auch die vordefinierten Werte `True` oder `GPIO.HIGH` ausgegeben werden.

Einrückungen sind in Python wichtig

In den meisten Programmiersprachen werden Programmschleifen oder Entscheidungen eingerückt, um den Programmcode übersichtlicher zu machen. In Python dienen diese Einrückungen nicht nur der Übersichtlichkeit, sondern sind auch für die Programmlogik zwingend nötig. Dafür braucht man hier keine speziellen Satzzeichen, um Schleifen oder Entscheidungen zu beenden.

```
010   time.sleep(0.2)
```

Diese Funktion aus der am Anfang des Programms importierten `time`-Bibliothek bewirkt eine Wartezeit von 0,2 Sekunden, bevor das Programm weiterläuft.

```
011   GPIO.output(25, 0)
```

Zum Ausschalten der LED gibt man den Wert `0` bzw. `False` oder `GPIO.LOW` auf dem GPIO-Port aus.

```
012   time.sleep(0.2)
```

Auch bei ausgeschalteter LED wartet das Programm wiederum 0,2 Sekunden. Danach ist die Schleife beendet, da dies die letzte eingerückte Zeile ist. Die Schleife beginnt von Neuem und die LED schaltet sich wieder kurz ein.

```
013 GPIO.cleanup()
```

Am Ende eines Programms müssen alle GPIO-Ports zurückgesetzt werden. Diese Zeile erledigt das für alle vom Programm initialisierten GPIO-Ports auf einmal. Ports, die von anderen Programmen initialisiert wurden, bleiben unberührt. So wird der Ablauf dieser anderen, möglicherweise parallel laufenden Programme nicht gestört.

Python-Programme brauchen keine eigene Anweisung zum Beenden. Sie enden einfach nach dem letzten Befehl bzw. nach einer Schleife, die nicht mehr ausgeführt wird und der keine weiteren Befehle folgen.

GPIO-Warnungen abfangen

Soll ein GPIO-Port verwendet werden, der nicht zurückgesetzt, sondern möglicherweise von einem anderen oder abgebrochenen Programm noch geöffnet ist, kommt es zu Warnungen, die den Programmfluss nicht unterbrechen. Bei der Programmierung sind diese Warnungen nützlich, um Fehler zu finden. Im fertigen Programm können sie den Anwender verwirren. Die GPIO-Bibliothek bietet mit `GPIO.setwarnings(False)` die Möglichkeit, diese Warnungen zu unterdrücken.

Quellcode der einzelnen Projekte

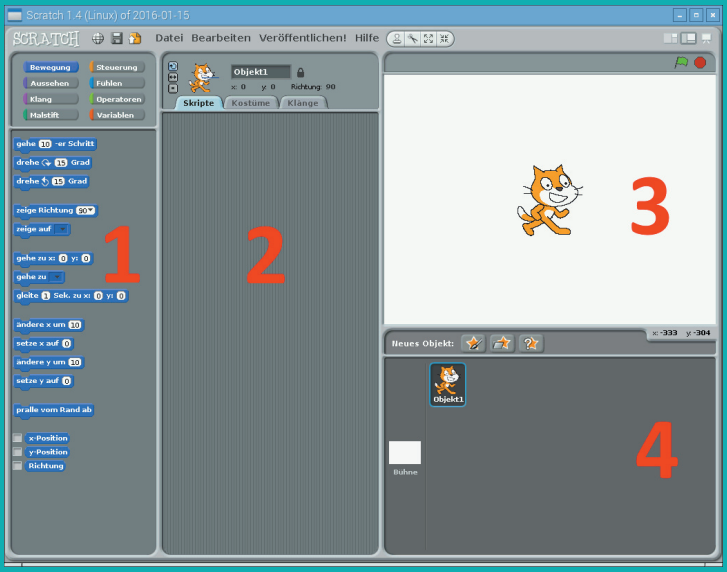
Den Quellcode zu den einzelnen Projekten können Sie unter Eingabe des Codes **65269-8** von *http://www.buch.cd* herunterladen.

Projektcode:
http://www.buch.cd

2 DAS ERSTE PROJEKT MIT SCRATCH

Scratch ist eine intuitive Programmierumgebung, mit der Kinder und Programmiereinsteiger schnell Ideen umsetzen können, ohne sich zuerst mit Programmiertheorie beschäftigen zu müssen. Scratch ist ein Projekt der Lifelong Kindergarten Group am Media-Lab des MIT und auf dem Raspberry Pi vorinstalliert. Passend für die Zielgruppe der Kinder und Jugendlichen eignet sich Scratch besonders für interaktive Animationen und Spiele. Eine grafische Oberfläche gibt bereits alle Grundlagen vor, sodass man sich um die Programmierung der Benutzeroberfläche des eigenen Programms keine Gedanken zu machen braucht. Die Scratch-Skripte werden nicht als Text geschrieben, sondern aus vorgefertigten Elementen zusammengeklickt.

Starten Sie Scratch über das Menü *Entwicklung*. Scratch startet mit einem viergeteilten Programmfenster:



Scratch starten

1. Links befindet sich die Blockpalette mit allen Elementen, aus denen ein Scratch-Skript zusammengesetzt werden kann. Da ausschließlich vordefinierte Blöcke verwendet werden, kann der Benutzer keine Syntaxfehler machen, die beim Einstieg in andere Programmiersprachen sehr ärgerlich und frustrierend sind.
2. Das mittlere Feld ist am Anfang noch leer. Hier entsteht später das Skript.
3. Rechts oben ist die sogenannte Bühne, die Oberfläche, auf der das Skript abläuft. Die Katze, die dort zu sehen ist, stellt beispielhaft ein Objekt dar, das mit Scratch animiert werden kann.
4. Das Objektfenster unten rechts zeigt die im Skript verwendeten Objekte. Hier können Sie später auch selbst eigene Objekte gestalten.

In einem allerersten einfachen Skript soll die Katze einmal im Kreis herum laufen und dabei ihre Farbe verändern.

1. Klicken Sie im Objektfenster unten rechts auf die Katze. Diese wird hervorgehoben und erscheint als *Objekt1* oberhalb des Skriptfensters. Alle Befehle im Skriptfenster beziehen sich dann auf dieses Objekt.
2. Klicken Sie in der Blockpalette oben auf das gelbe Symbol *Steuerung*. Damit werden die Blöcke zur Programmsteuerung angezeigt.
3. Ziehen Sie den abgebildeten Block aus der Blockpalette in das Skriptfenster. Auf der Scratch-Bühne ist oben rechts ein grünes Fähnchen. Dieses dient üblicherweise dazu, ein Programm zu starten. Das abgebildete Element bewirkt, dass die folgenden Skriptelemente ausgeführt werden, wenn der Benutzer auf das grüne Fähnchen klickt.

4. Die kreisförmige Bewegung wird aus einzelnen Gehen- und Drehen-Schritten zusammengesetzt. Diese werden so oft wiederholt, bis die Katze einen ganzen Kreis gegangen ist. Ziehen Sie für die Wiederholungsschleife den abgebildeten Block in das Skriptfenster und dokken ihn unten an das dort bereits vorhandene Element an.

5. In jedem Bewegungsschritt soll sich die Katze um 15 Grad drehen. Dabei dreht sie sich in 24 Schritten genau einmal. Klicken Sie in das weiße Zahlenfeld des *Wiederhole*-Blocks und ändern Sie den vorgegebenen Wert auf *24*.

2

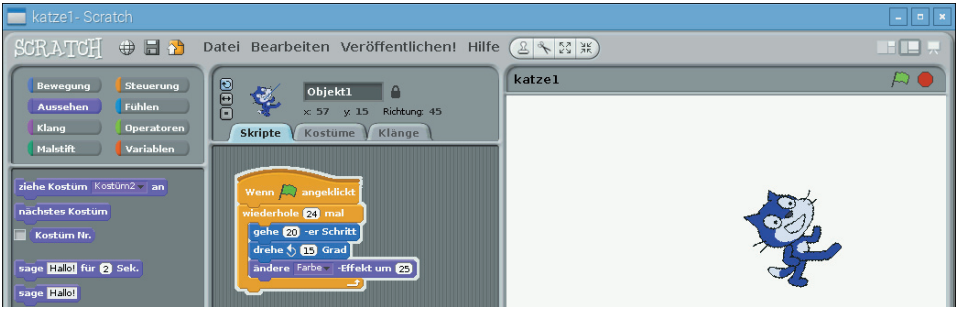
6 Damit die Katze eine Kreisbewegung ausführt, muss sie zunächst einen Schritt gehen, sich danach um 15 Grad drehen, wieder einen Schritt gehen usw. Klikken Sie oben in der Blockpalette auf das blaue Symbol *Bewegung* und ziehen Sie den abgebildeten Block in das Skriptfenster. Docken Sie ihn innerhalb der Schleife an. Ändern Sie dann noch die Schrittweite auf 20.

7 Ziehen Sie danach den Block für die Drehbewegung gegen den Uhrzeigersinn in das Skriptfenster. Platzieren Sie ihn so über den vorhandenen Blöcken, dass er sich innerhalb der Schleife nach der Gehbewegung einklinkt.

8 Das Skript sollte jetzt wie abgebildet aussehen. Probieren Sie aus, ob es auch wie erwartet funktioniert. Klicken Sie dazu rechts oberhalb der Bühne auf das grüne Fähnchen. Die Katze wird eine Kreisbewegung gehen.

9 Jetzt fehlt nur noch die gewünschte Veränderung der Farbe. Klicken Sie dazu oben in der Blockpalette auf das violette Symbol *Aussehen*. Jetzt werden Blöcke angeboten, die das Aussehen des aktiven Objekts beeinflussen. Ziehen Sie den abgebildeten Block in das Skriptfenster in die Wiederholung hinter den *Drehe*-Block.

10 Lassen Sie das Skript jetzt wieder ablaufen, ändert die Katze im Laufe ihrer Bewegung zyklisch ihre Farbe. Am Ende der 24 Wiederholungen steht die Katze wieder an ihrer ursprünglichen Position und hat auch wieder ihre ursprüngliche Farbe.



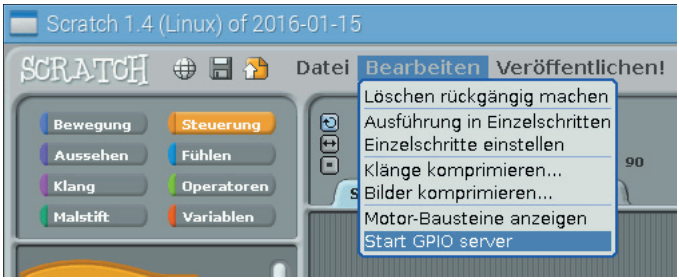
Gehen, drehen, Farbe ändern im Scratch-Programm »katze1«

SCRATCH UND GPIO

3

Scratch bietet in der Version, die im aktuellen Raspbian-Jessie-Betriebssystem vorinstalliert ist, eine eigene Unterstützung für die GPIO-Schnittstelle. Früher mussten dazu externe Funktionsbibliotheken eingebunden und spezielle Einstellungen vorgenommen werden.

Um die GPIO-Unterstützung in Scratch nutzen zu können, muss der GPIO-Server über den Menüpunkt *Bearbeiten/Start GPIO server* gestartet werden. Denken Sie bei jedem neuen Scratch-Programm daran, dies zu überprüfen. Wenn die GPIO-Funktionen bereits aktiv sind, ändert sich dieser Menüpunkt automatisch in *Stop GPIO server*.



GPIO-Server in Scratch starten

3.1 | Die erste LED blinkt in Scratch

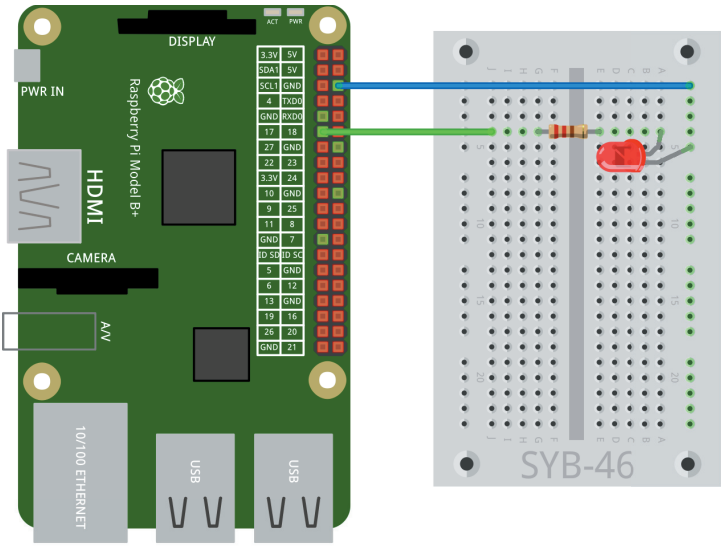
Verglichen mit dem Versuch, mit einem Windows PC eine extern angeschlossene LED zum Blinken zu bringen, ist Scratch wirklich ganz einfach. Bauen Sie die abgebildete Schaltung auf einer Steckplatine auf.

Projektcode:
led1.sb

Benötigte Bauteile

1x Steckplatine, 1x LED rot, 1x 220-Ohm-Widerstand (Rot-Rot-Braun), 2x Verbindungskabel

3



Die LED mit Vorwiderstand am Raspberry Pi

Öffnen Sie über den Menüpunkt *Datei/Öffnen* das Skript `led1` in Scratch und starten Sie es mit einem Klick auf das grüne Fähnchen. Die LED blinkt. Sie können das Programm natürlich auch selbst aus Scratch-Blöcken zusammensetzen.

3.1.1 | So funktioniert das Programm

Das Programm startet wie die meisten Scratch-Programme mit dem Block *Wenn grünes Fähnchen angeklickt*. Dieser ist in Scratch auf der Blockpalette *Steuerung* zu finden. Der Block ist oben rund, passt also unter keinen anderen Block. Er muss immer als Erstes gesetzt werden.

Scratch verwendet für die GPIO-Funktionen den Steuerungsblock *sende...an alle*.

Dieser Block enthält ein Feld, in dem freier Text eingegeben werden kann. Klicken Sie darauf, erscheint eine Liste der zuletzt verwendeten Eingaben. Ein Klick auf *Neu/edit...* öffnet ein Eingabefeld für neuen Text.

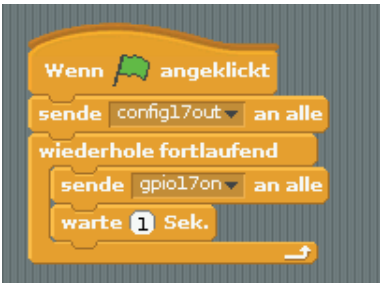
Schreiben Sie in dieses Feld `config17out`. Das definiert den GPIO-Pin 17 als Ausgang. Jeder GPIO-Pin muss als Ausgang (z. B. für LEDs) oder als Eingang (z. B. für Taster) definiert werden, bevor er verwendet werden kann.

Jetzt soll eine Endlosschleife starten, die die LED blinken lässt, bis Sie auf das rote Stoppsymbol klicken.

Innerhalb der Schleife wird als Erstes der GPIO-Pin 17 eingeschaltet. Ziehen Sie dazu wieder einen *sende...an alle*-Block in die Schleife hinein und schreiben Sie `gpio17on` in das Textfeld dieses Blocks. Damit wird der GPIO-Pin 17 eingeschaltet und die LED wird leuchten.

Danach soll das Programm eine Sekunde warten, bis die LED wieder ausgeschaltet wird. Fügen Sie dazu einen *warte...Sek*-Block ein und schreiben Sie `1` in das Textfeld.

Um die LED auszuschalten, fügen Sie wieder einen *sende...an alle*-Block hinter der Wartezeit in die Schleife ein. Schreiben Sie hier `gpio17off` in das Textfeld. Damit wird der GPIO-Pin 17 wieder ausgeschaltet.

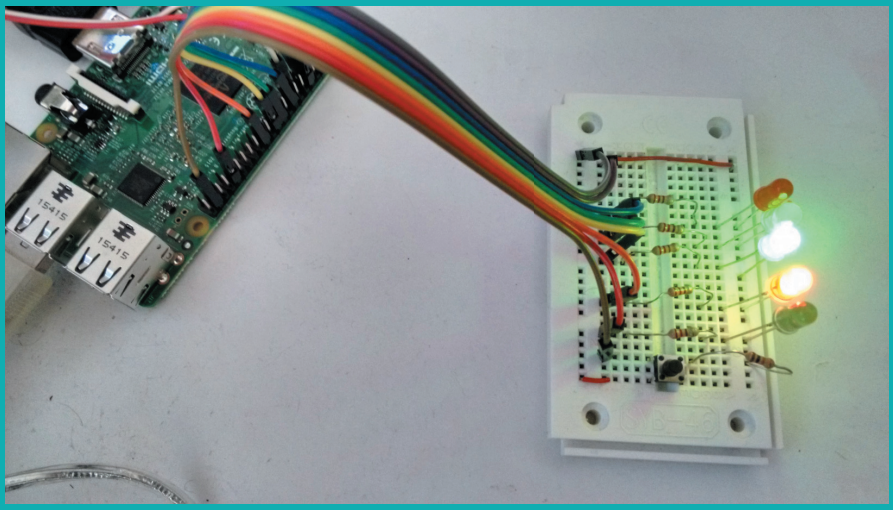


Zum Schluss kommt wieder ein *warte...Sek*-Block mit einer Sekunde. Das bewirkt, dass die LED auch eine Sekunde ausgeschaltet bleibt, bevor die Schleife wieder neu beginnt und die LED einschaltet.

4 FUSSGÄNGER-AMPEL

Eine einzelne LED ein- und wieder auszuschalten mag im ersten Moment ganz spannend sein, aber dafür braucht man eigentlich keinen Computer. Eine Verkehrsampel mit ihrem typischen Leuchtzyklus von Grün über Gelb nach Rot und dann über eine Lichtkombination Rot-Gelb wieder zu Grün zeigt weitere Programmiertechniken in Python.

Das nächste Experiment stellt eine einfache Ampelschaltung mit Fußgängerampel auf einem Steckbrett mit fünf LEDs dar, die während der Rotphase der Verkehrsampel eine Grünphase für Fußgänger anzeigt.

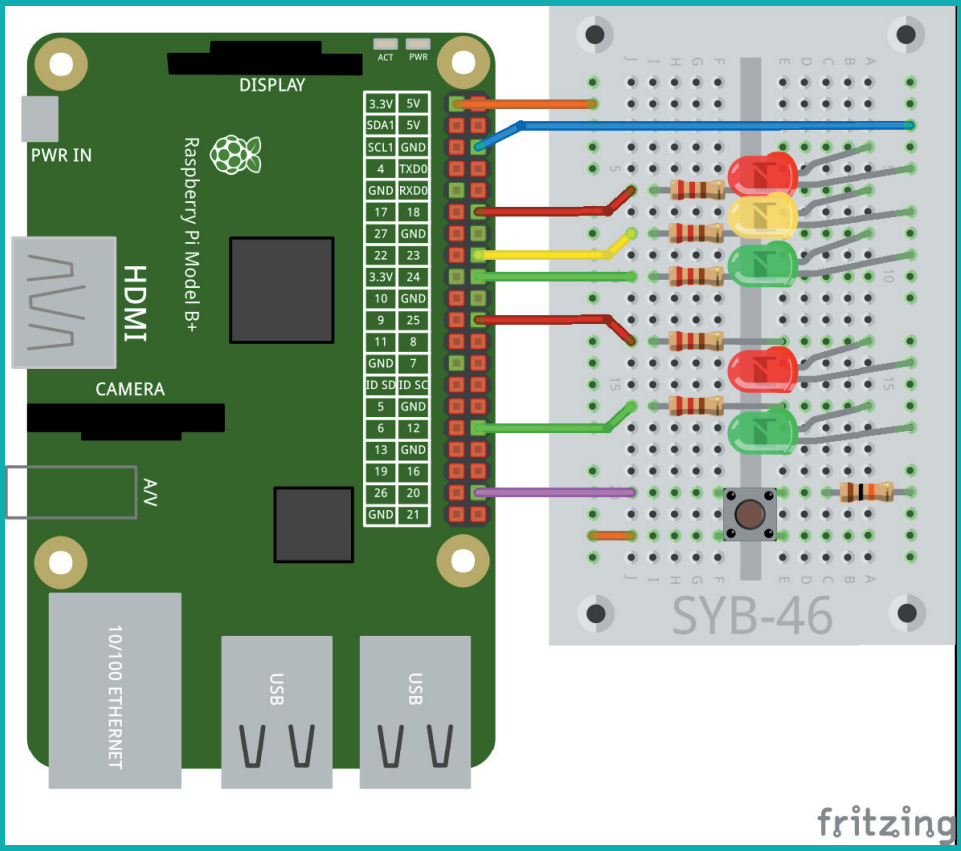


Die Ampel auf einem Steckbrett am Raspberry Pi

Bauen Sie für das folgende Experiment fünf LEDs mit Vorwiderständen und einen Taster wie abgebildet auf einer Steckplatine auf.

Benötigte Bauteile

1x Steckplatine, 2x LED rot, 1x LED gelb, 2x LED grün, 5x 220-Ohm-Widerstand (Rot-Rot-Braun), 1x Taster, 1x 10-kOhm-Widerstand (Braun-Schwarz-Orange), 8x Verbindungskabel, 1x kurze Drahtbrücke



Verkehrsampel mit Fußgängerampel

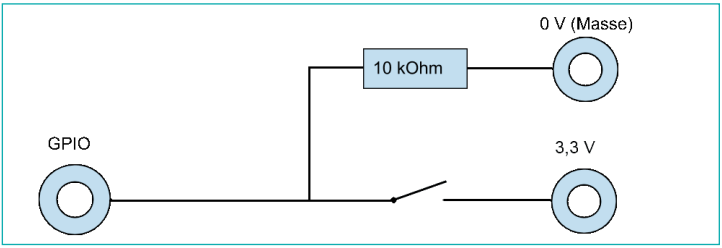
4

4.1 | Taster am GPIO-Anschluss

GPIO-Ports können nicht nur Daten ausgeben, z. B. über LEDs, sondern auch zur Dateneingabe verwendet werden. Dazu müssen sie im Programm als Eingang definiert werden. Zur Eingabe verwenden wir im nächsten Projekt einen Taster, der direkt auf die Steckplatine gesteckt wird. Der Taster hat vier Anschlusspins, wobei je zwei gegenüberliegende (großer Abstand) miteinander verbunden sind. Solange die Taste gedrückt ist, sind alle vier Anschlüsse miteinander verbunden. Im Gegensatz zu einem Schalter rastet ein Taster nicht ein. Die Verbindung wird beim Loslassen sofort wieder getrennt.

Liegt auf einem als Eingang definierten GPIO-Port ein +3,3-V-Signal an, wird dieses als logisch `True` bzw. `1` ausgewertet. Theoretisch könnten Sie also über einen Taster den jeweiligen GPIO-Port mit dem +3,3-V-Anschluss des Raspberry Pi verbinden.

In den meisten Fällen funktioniert diese simple Schaltung bereits, allerdings hätte der GPIO-Port bei offenem Taster keinen eindeutig definierten Zustand. Wenn ein Programm diesen Port abfragt, kann es zu zufälligen Ergebnissen kommen. Um das zu verhindern, schließt man einen vergleichsweise sehr hohen Widerstand – üblicherweise 10 kOhm – gegen Masse. Dieser sogenannte Pulldown-Widerstand zieht den Status des GPIO-Ports bei geöffnetem Taster wieder nach unten auf 0 V. Da der Widerstand sehr hoch ist, besteht, solange der Taster gedrückt ist, auch keine Kurzschlussgefahr. Im gedrückten Zustand des Tasters sind +3,3 V und die Masseleitung direkt über diesen Widerstand verbunden.



Taster mit Pulldown-Widerstand an einem GPIO-Eingang

Die in der Abbildung der aufgebauten Schaltung untere Kontaktleiste des Tasters ist über die Plusschiene der Steckplatine mit der +3,3-V-Leitung des Raspberry Pi (Pin 1) verbunden. Zur Verbindung des Tasters mit der Plusschiene verwenden wir, um die Zeichnung übersichtlich zu halten, eine kurze Drahtbrücke. Alternativ können Sie auch einen der unteren

Kontakte des Tasters direkt mit einem Verbindungskabel mit dem Pin 1 des Raspberry Pi verbinden.

Die in der Abbildung obere Kontaktleiste des Tasters ist mit dem GPIO-Port 20 verbunden und über einen 10-kOhm-Pulldown-Widerstand (Braun-Schwarz-Orange) mit 0 V Masse.

4.2 | Fußgängerampel mit Python

Programmdatei:
ampel1.py

Im Ruhezustand soll die Verkehrsampel Grün zeigen, die Fußgängerampel Rot. Drückt man auf die Taste, schaltet die Verkehrsampel nacheinander über Gelb auf Rot. Danach erst schaltet die Fußgängerampel auf Grün. Nach einer Grünphase von zwei Sekunden schaltet die Fußgängerampel wieder auf Rot. Dann beginnt der Schaltzyklus der Verkehrsampel über Rot-Gelb nach Grün. Danach wartet das Programm mit dieser Einstellung, bis der Benutzer erneut die Taste drückt. drückt. Das Programm ampel1.py steuert die Ampelanlage.

4.2.1 | So funktioniert das Programm

Am Anfang steht eine neue Zeile im Programm:

```
002 # -*- coding: utf-8 -*-
```

Damit die deutschen Umlaute im Wort Fußgängerampel in der Programmausgabe korrekt angezeigt werden – unabhängig davon, wie die IDLE-Oberfläche beim Benutzer eingestellt ist –, wird am Anfang eine Codierung zur Darstellung der Sonderzeichen definiert. Diese Zeile sollte in allen Programmen enthalten sein, die Texte ausgeben, in denen sich Umlaute oder andere landestypische Sonderzeichen befinden.

Die folgenden Zeilen sind bereits bekannt, sie importieren die Bibliotheken `RPi.GPIO` für die Ansteuerung der GPIO-Ports und `time` für Zeitverzögerungen.

```
006 rot    = 0
007 gelb   = 1
008 gruen  = 2
009 f_rot  = 3
010 f_gru  = 4
011 taste  = 5
```

Diese Zeilen definieren die drei Variablen `rot`, `gelb`, `gruen` für die drei LEDs der Verkehrsampel, `f_rot`, `f_gru` für die LEDs der Fußgängerampel sowie `taste` für den Taster. Damit braucht man sich im Programm keine Nummern der GPIO-Ports zu merken, sondern kann die LEDs einfach über ihre Farbe ansteuern.

```
012 Ampel=[18,23,24,25,12,20]
```

Zur Ansteuerung der drei LEDs wird eine Liste eingerichtet, die die GPIO-Nummern in der Reihenfolge enthält, in der sie zuvor definiert wurden. Da die GPIO-Ports nur an dieser einen Stelle im Programm auftauchen, können Sie das Programm ganz einfach umbauen, wenn Sie andere GPIO-Ports nutzen möchten. Danach wird die Nummerierung der GPIO-Ports auf BCM gesetzt.

```
012 GPIO.setmode(GPIO.BCM)
013 GPIO.setup(Ampel[rot], GPIO.OUT, initial=0)
014 GPIO.setup(Ampel[gelb], GPIO.OUT, initial=0)
015 GPIO.setup(Ampel[gruen], GPIO.OUT, initial=1)
016 GPIO.setup(Ampel[f_rot], GPIO.OUT, initial=1)
017 GPIO.setup(Ampel[f_gru], GPIO.OUT, initial=0)
018 GPIO.setup(Ampel[taste], GPIO.IN)
```

Nacheinander werden die verwendeten GPIO-Ports als Ausgänge initialisiert. Dabei verwenden wir keine GPIO-Portnummern, sondern die zuvor definierte Liste. Innerhalb einer Liste werden die einzelnen Elemente über Zahlen, mit 0 beginnend, indiziert.

Die Variablen `rot`, `gelb` und `gruen` enthalten die Zahlen `0`, `1` und `2`, die als Indizes für die Elemente der Liste benötigt werden. Auf diese Weise lassen sich die verwendeten GPIO-Ports über Farben adressieren:

- `Ampel[rot]` entspricht dem GPIO-Port `18` mit der roten LED.
- `Ampel[gelb]` entspricht dem GPIO-Port `23` mit der gelben LED.
- `Ampel[gruen]` entspricht dem GPIO-Port `24` mit der grünen LED.
- `Ampel[f_rot]` entspricht dem GPIO-Port `25` mit der roten LED der Fußgängerampel.
- `Ampel[f_gru]` entspricht dem GPIO-Port `12` mit der grünen LED der Fußgängerampel.
- `Ampel[taste]` entspricht dem GPIO-Port `20` mit dem Taster.

Der GPIO-Port des Tasters wird als Eingang definiert. Diese Definition erfolgt ebenfalls über `GPIO.setup`, diesmal aber mit dem Parameter `GPIO.IN`.

Die `GPIO.setup`-Anweisung kann einen optionalen Parameter `initial` enthalten, der dem GPIO-Port bereits beim Initialisieren einen logischen Status zuweist. Damit schalten wir in diesem Programm die grüne LED der Verkehrsampel sowie die rote LED der Fußgängerampel bereits von Anfang an ein. Die anderen LEDs beginnen das Programm im ausgeschalteten Zustand.

```
022 print ("Taster drücken, um Fußgängerampel
    einzuschalten, Strg+C beendet das Programm")
```

Jetzt wird eine kurze Bedienungsanleitung auf dem Bildschirm ausgegeben, die mitteilt, dass der Benutzer den Taster drücken soll. Das Programm läuft automatisch. Die Tastenkombination Strg+C soll es beenden.

Um abzufragen, ob der Benutzer mit Strg+C das Programm beendet, verwenden wir eine `try...except`-Abfrage. Dabei wird der unter `try:` eingetragene Programmcode zunächst normal ausgeführt. Wenn währenddessen eine Systemausnahme auftritt – das kann ein Fehler sein oder auch die Tastenkombination Strg+C –, wird abgebrochen und die `except`-Anweisung am Programmende wird ausgeführt.

```
045 except KeyboardInterrupt:
046   GPIO.cleanup()
```

Die Tastenkombination Strg+C löst einen `KeyboardInterrupt` aus, und die Schleife wird automatisch verlassen. Die letzte Zeile schließt die verwendeten GPIO-Ports und schaltet damit alle LEDs aus. Danach wird das Programm beendet.

Durch das kontrollierte Schließen der GPIO-Ports tauchen keine Systemwarnungen oder Abbruchmeldungen auf, die den Benutzer verwirren könnten.

Der eigentliche Ampelzyklus läuft in einer Endlosschleife:

```
025     while True:
```

Solche Endlosschleifen benötigen immer eine Abbruchbedingung, da das Programm sonst nie beendet werden würde.

```
026         if GPIO.input(Ampel[taste])==1:
```

Innerhalb der Endlosschleife ist eine Abfrage eingebaut. Die folgenden Anweisungen werden erst ausgeführt, wenn der GPIO-Port mit dem Taster den Wert `True` annimmt, der Benutzer also den Taster drückt. Bis zu diesem Zeitpunkt bleibt die Verkehrsampel in ihrer Grünphase stehen.

```
027                GPIO.output(Ampel[gruen],0)
028                GPIO.output(Ampel[gelb],1)
029                time.sleep(0.6)
```

Jetzt wird die grüne LED aus- und dafür die gelbe LED eingeschaltet. Diese leuchtet dann alleine für 0,6 Sekunden.

```
030                GPIO.output(Ampel[gelb],0)
031                GPIO.output(Ampel[rot],1)
032                time.sleep(0.6)
```

Jetzt wird die gelbe LED wieder ausgeschaltet und dafür die rote LED eingeschaltet und wieder 0,6 Sekunden gewartet.

```
033                GPIO.output(Ampel[f_rot],0)
034                GPIO.output(Ampel[f_gru],1)
035                time.sleep(2)
```

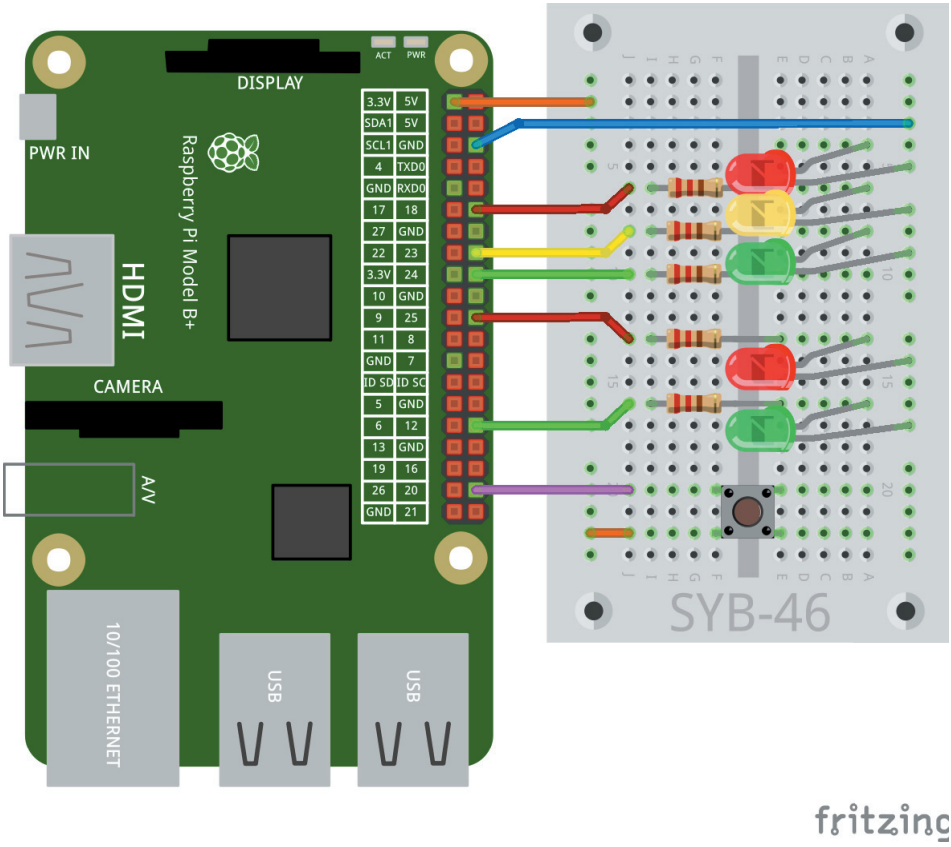
Erst danach schaltet die Fußgängerampel von Rot auf Grün um. Diese Phase dauert 2 Sekunden. Danach schaltet die Fußgängerampel auf Rot und mit jeweils 0,6 Sekunden Verzögerung schaltet die Verkehrsampel über Rot-Gelb auf Grün um.

Am Ende der Schleife leuchtet die Verkehrsampel wieder grün, die Fußgängerampel rot. Die Schleife beginnt in der Grünphase der Ampel von Neuem nach einer Wartezeit von 2 Sekunden. Um zu verhindern, dass die Grünphase fast ausfällt, wenn der Taster unmittelbar nach der Gelbphase wieder gedrückt wird, ist diese Verzögerung am Ende der Schleife eingebaut. Natürlich können Sie alle Zeiten beliebig anpassen. In der Realität hängen die Ampelphasen von den Maßen der Kreuzung und den Verkehrsströmen ab. Die Gelb- und die Rot-Gelb-Phase sind üblicherweise je 2 Sekunden lang.

4.2.2 | Internen Pulldown-Widerstand nutzen

Aktuelle Raspberry-Pi-Modelle verfügen über eingebaute Pulldown-Widerstände an den GPIO-Ports, die per Software eingeschaltet werden können. Auf diese Weise kann man sich externe Pulldown-Widerstände sparen.

Entfernen Sie den 10-kOhm-Pulldown-Widerstand aus der Schaltung. Der Ampelzyklus wird immer mal wieder zufällig starten, ohne dass Sie den Taster zu drücken brauchen.



Die Ampel ohne Pulldown-Widerstand

Der interne Pulldown-Widerstand wird bei der Initialisierung des GPIO-Eingangs über einen zusätzlichen Parameter in der Funktion `GPIO.setup()` eingeschaltet.

Programmdatei:
ampel2.py

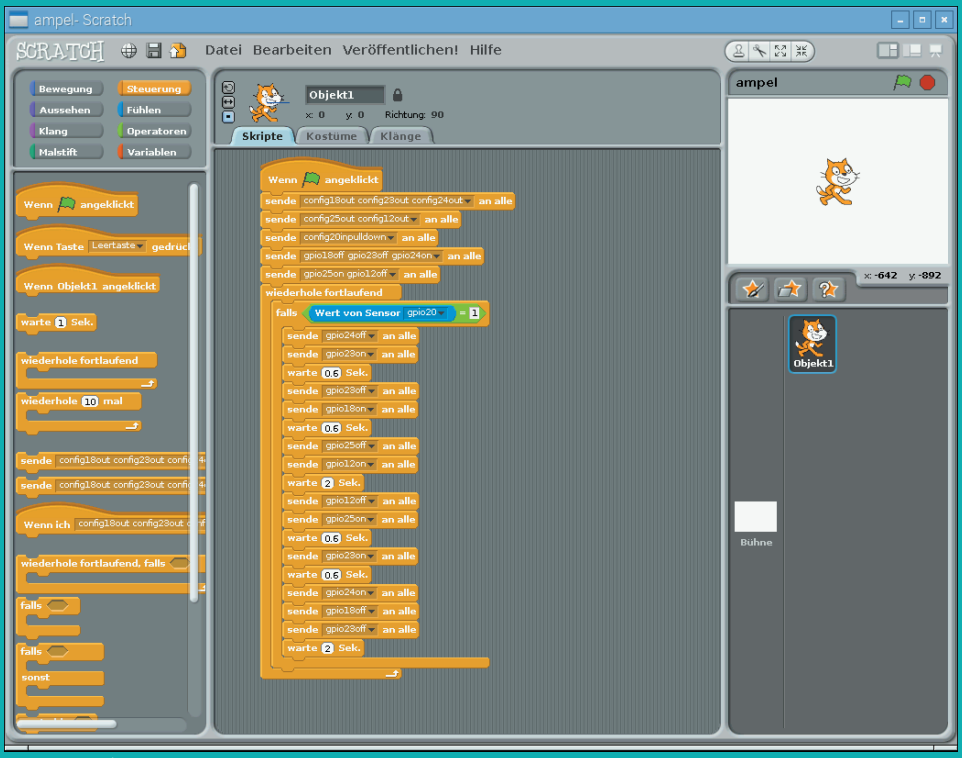
```
020 GPIO.setup(Ampel[taste], GPIO.IN, pull_up_down = GPIO.PUD_DOWN)
```

Das Programm `ampel2.py` enthält diese Änderung bereits. Starten Sie das Programm und die Ampel lässt sich mit dem Taster steuern, auch ohne dass ein externer Pulldown-Widerstand eingebaut ist.

5 FUSSGÄNGER-AMPEL MIT SCRATCH

Programm-datei:
ampel.sb

Die gleiche Fußgängerampel lässt sich auch mit Scratch programmieren. Da die Initialisierung der GPIO-Pins in Scratch einfacher ist, wirkt das Skript noch etwas übersichtlicher als die Python-Version.



Das Scratch-Skript für die Fußgängerampel in einem größeren Skript-Fenster

Um mehr Platz für das Skript zu haben, können Sie mit den Symbolen oben rechts im Scratch-Fenster den Skriptbereich verbreitern und die Bühne verkleinern.

5.1 | So funktioniert das Programm

Wenn der Benutzer auf das grüne Fähnchen klickt, werden als Erstes die sechs verwendeten GPO-Pins initialisiert. Wie in der Python-Version der Ampel verwenden wir die Ports 18, 23, 24, 25 und 12 als Ausgänge für die LEDs und den Port 20 mit eingeschaltetem Pulldown-Widerstand als Eingang für den Taster. Der Block *sende config20inpulldown an alle* schaltet am GPIO-Eingang 20 den internen Pulldown-Widerstand ein.

Danach werden die grüne LED der Verkehrsampel und die rote LED der Fußgängerampel eingeschaltet. Die anderen drei LEDs werden ausgeschaltet. Das Ausschalten ist hier am Anfang eigentlich nicht nötig. Es dient nur dazu, das Skript auf jeden Fall mit einem klar definierten Zustand zu starten.

Jetzt beginnt, wie im Python-Skript, eine Endlosschleife, die darauf wartet, dass der Benutzer die Taste drückt und dann den Ampelzyklus startet.

An dieser Stelle wird der Status des Tasters abgefragt. Dazu verwenden wir den Scratch-Block *falls*, der ähnlich wie eine `if`-Abfrage in Python funktioniert.

Innerhalb des Blocks werden die Blöcke eingefügt, die ausgeführt werden sollen, wenn die Abfrage wahr ist.

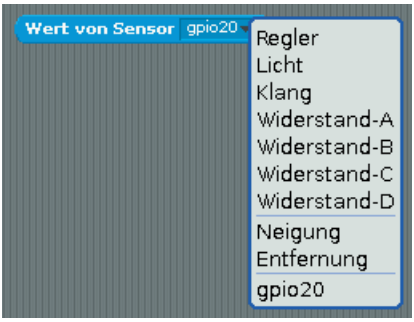
Für die Abfrage selbst ist im *falls*-Block ein längliches Feld mit spitzen Enden vorgesehen. Hier muss ein Block aus der grünen Blockpalette *Operatoren* eingefügt werden. Wählen Sie den Block mit dem Gleichheitszeichen und ziehen Sie ihn auf das Platzhalterfeld im Block *falls*.

Dieser Operator ist immer dann wahr, wenn die beiden Werte links und rechts des Gleichheitszeichens gleich sind.

In unserem Fall soll der Wert des GPIO-Pins 20 dem Wert `1` entsprechen. Ein gedrückter Taster gibt auch hier wie in Python, den Wert `1` zurück. Zur Abfrage von GPIO-Eingängen wird der Block *Wert von Sensor* aus der blauen Blockpalette *Fühlen* verwendet.

5

Ziehen Sie diesen Block auf das Platzhalterfeld links im grünen Gleichheitsoperator. Wählen Sie im Listenfeld des blauen Blocks den Sensor *gpio20* aus. Neben einigen vordefinierten Sensoren werden alle GPIO-Pins zur Auswahl angeboten, die als Eingang definiert sind. Tragen Sie dann noch rechts im grünen Gleichheitsoperator den Wert `1` ein.



GPIO-Eingang erscheint nicht

Sollte der GPIO-Eingang auch nach einigen Sekunden nicht in der Auswahlliste auftauchen, trennen Sie das Scratch-Programm hinter der Initialisierung des Eingangs mit dem Block *sende config20inpulldown an alle*. Starten Sie die Initialisierungsblöcke jetzt mit einem Klick auf das grüne Fähnchen. Danach erscheint der GPIO-Eingang in der Liste im Block *Wert von Sensor...*. Anschließend können Sie das Programm wieder zusammenfügen.

Wenn diese Abfrage wahr wird, der Benutzer also den Taster gedrückt hat, beginnt der Ampelzyklus. Im ersten Schritt wird die grüne LED der Verkehrsampel (GPIO 24) ausgeschaltet und dafür die gelbe (GPIO 23) eingeschaltet. Danach wartet das Skript 0,6 Sekunden.

Zum Umschalten der GPIO-Ports wird wieder der Block *sende...an alle* verwendet. Nach der Gelbphase schaltet die Verkehrsampel auf Rot. Dazu wird die gelbe LED (GPIO 23) ausgeschaltet und dafür die rote (GPIO 18) eingeschaltet. Auch diese Phase dauert 0,6 Sekunden.

Im nächsten Schritt schaltet die Fußgängerampel auf Grün, wobei die Verkehrsampel unverändert Rot anzeigt. Die grüne LED der Fußgängerampel (GPIO 12) wird eingeschaltet, die rote (GPIO 25) aus. Die Grünphase der Fußgängerampel dauert 2 Sekunden.

Am Ende der Grünphase wird die rote LED der Fußgängerampel (GPIO 25) wieder eingeschaltet, die grüne (GPIO 12) aus. Diese Schaltphase dauert 0,6 Sekunden.

Die rote LED der Verkehrsampel leuchtet an dieser Stelle bereits. Für die jetzt folgende Rot-Gelb-Phase braucht nur die gelbe LED (GPIO 23) zusätzlich eingeschaltet zu werden.

Nach der Rot-Gelb-Phase werden die rote LED (GPIO 25) und die gelbe LED (GPIO 23) der Verkehrsampel ausgeschaltet und die grüne LED (GPIO 24) eingeschaltet.

Nach dem Start der Grünphase wartet die Ampelschaltung zwei Sekunden, bevor die Befehlsfolge innerhalb der *falls*-Abfrage beendet wird. Damit wird verhindert, dass die Grünphase fast ausfällt, wenn der Taster unmittelbar nach der Gelbphase wieder gedrückt wird.

Danach startet die Endlosschleife neu und ruft je nach ermitteltem Sensorwert den Ampelzyklus auf.

5.2 | Die Katze bewegt sich zur Ampel

Im nächsten Schritt wird sich die Katze, die Symbolfigur von Scratch, passend zur Ampelschaltung bewegen und damit zeigen, dass sich klassische Scratch-Funktionen mit GPIO-Befehlen in einem Skript frei kombinieren lassen.

Das Skript `ampel_katze` enthält dazu ein paar zusätzliche Blöcke.

5

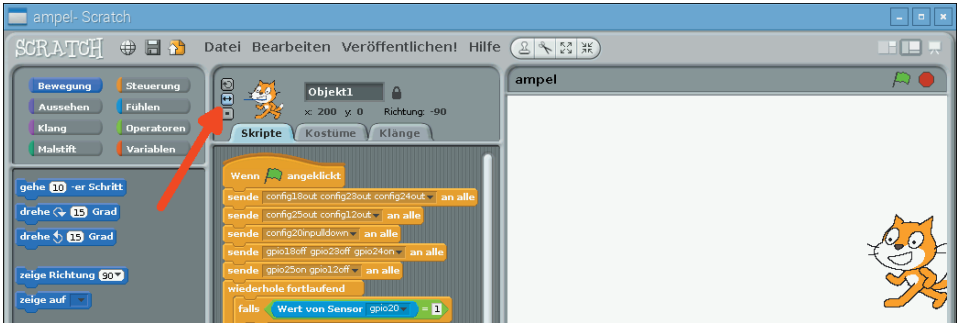

Wenn angeklickt
sende config18out config23out config24out an alle
sende config25out config12out an alle
sende config20inpulldown an alle
sende gpio18off gpio23off gpio24on an alle
sende gpio25on gpio12off an alle
wiederhole fortlaufend
falls Wert von Sensor gpio20 = 1
sende gpio24off an alle
sende gpio23on an alle
warte 0.6 Sek.
sende gpio23off an alle
sende gpio18on an alle
warte 0.6 Sek.
sende gpio25off an alle
sende gpio12on an alle
wiederhole 10 mal
gehe 20 -er Schritt
warte 0.1 Sek.
drehe 180 Grad
warte 2 Sek.
sende gpio12off an alle
sende gpio25on an alle
warte 0.6 Sek.
sende gpio23on an alle
warte 0.6 Sek.
sende gpio24on an alle
sende gpio18off an alle
sende gpio23off an alle
warte 2 Sek.

Die Ampelsteuerung läuft wie in der vorherigen Version des Skripts. Nur während der Grünphase der Fußgängerampel werden ein paar neue Blöcke eingefügt, um die Katze zu bewegen.

Nachdem die Fußgängerampel auf Grün schaltet, soll die Katze von ihrer aktuellen Position auf die andere Straßenseite gehen, sich dort um 180° drehen, um in der nächsten Grünphase der Fußgängerampel wieder zurückzugehen.

Anstatt einen 200er-Schritt auf einmal zu gehen oder eher zu springen, wird die Bewegung in zehn 20er-Schritte aufgeteilt, zwischen denen die Katze jeweils 0,1 Sekunden wartet. Durch diese Wiederholung erscheint die Bewegung flüssiger.

Nach zehn Wiederholungen soll sich die Katze um 180 Grad drehen. Damit sie nach der Drehung nicht auf dem Kopf steht, aktivieren Sie das Doppelpfeilsymbol bei der Katze oberhalb des Skriptbereichs.

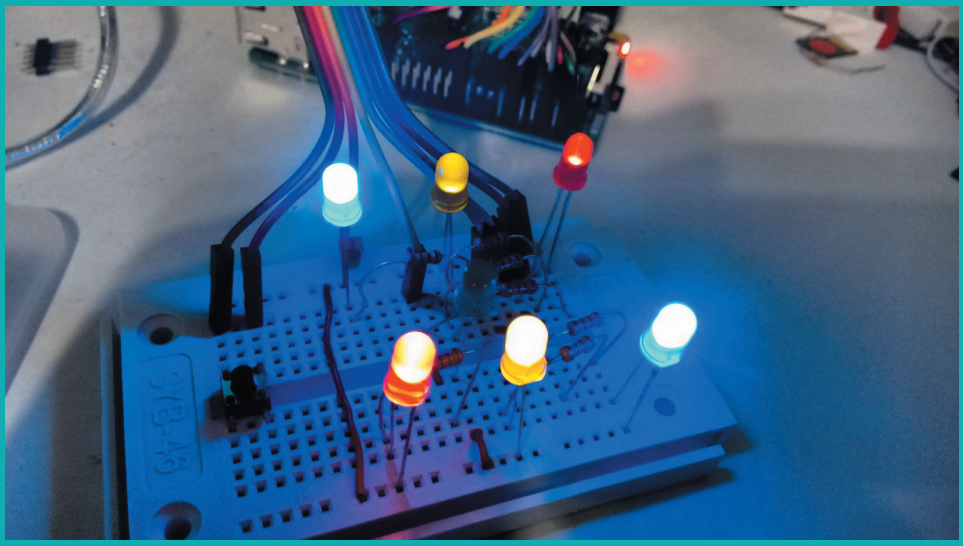


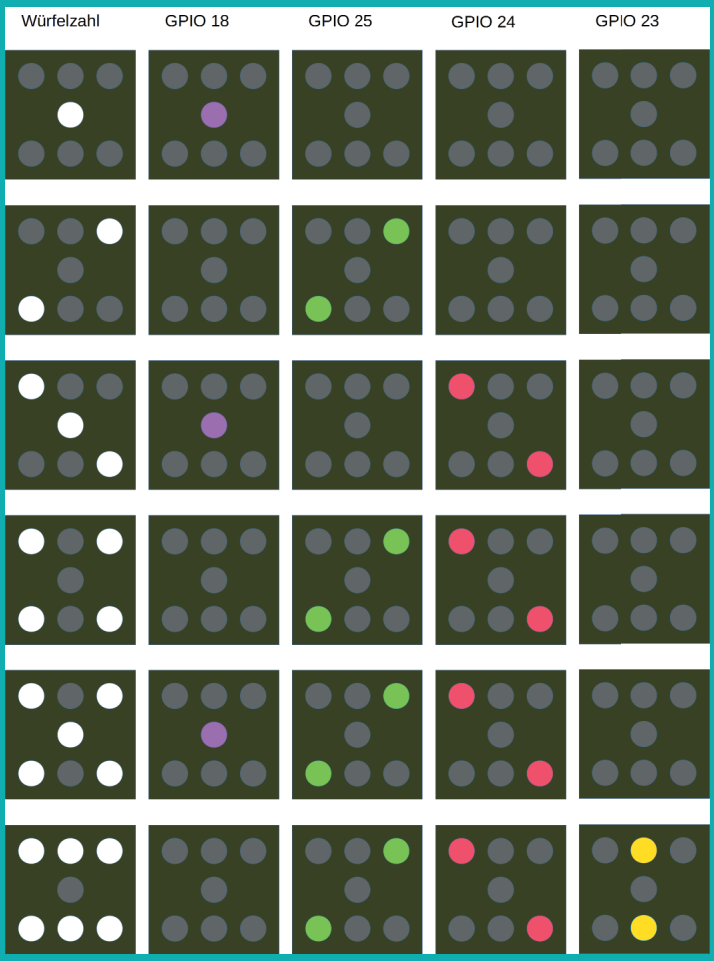
Dieses Symbol richtet die Katze nach rechts und links aus, anstatt sie frei zu drehen.

Wenn Sie jetzt das Skript starten und auf die Taste der Ampel drücken, läuft die Katze in der ersten Grünphase der Fußgängerampel nach rechts und dreht sich dort. Beim nächsten Drücken läuft sie wieder nach links auf ihre Startposition. Da nur relative Bewegungen verwendet werden, braucht man im Skript nicht zwischen dem Gang nach rechts und nach links zu unterscheiden.

6 SPIELWÜRFEL MIT LEDS

Die typischen Spielwürfel, die ein bis sechs Augen zeigen, kennt jeder und hat jeder zu Hause. Wesentlich cooler ist ein elektronisch gesteuerter Würfel, der auf Tastendruck die Augen leuchten lässt – und dann aber nicht einfach ein bis sechs LEDs in einer Reihe, sondern in der Anordnung eines Spielwürfels. Diese haben Augen in der typischen quadratischen Anordnung, wozu man sieben LEDs braucht.



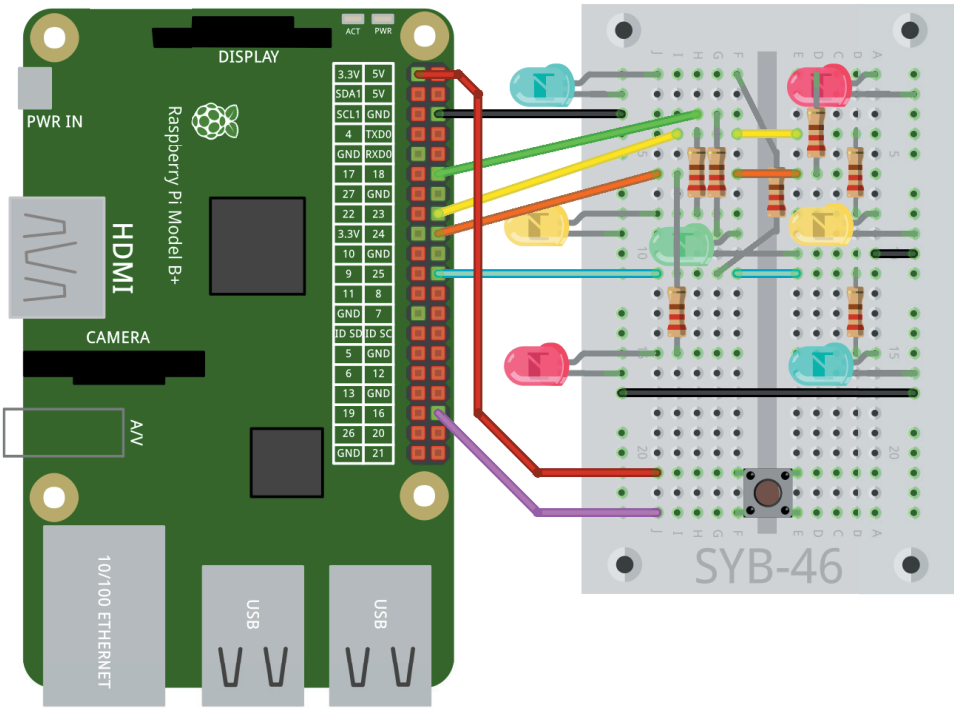


Die Würfelaugensymbole mit LED-Gruppen dargestellt

Bauen Sie für das folgende Experiment sieben LEDs mit Vorwiderständen und einen Taster wie abgebildet auf einer Steckplatine auf. Für die Ansteuerung der LEDs werden nur vier statt sieben GPIO-Pins benötigt, da ein Würfel zur Darstellung gerader Zahlen die Augen paarweise nutzt.

Benötigte Bauteile

1x Steckplatine, 2x LED rot, 2x LED gelb, 2x LED blau, 1x LED grün, 7x 220-Ohm-Widerstand (Rot-Rot-Braun), 1x Taster, 7x Verbindungskabel, 5x Drahtbrücke (unterschiedliche Längen)

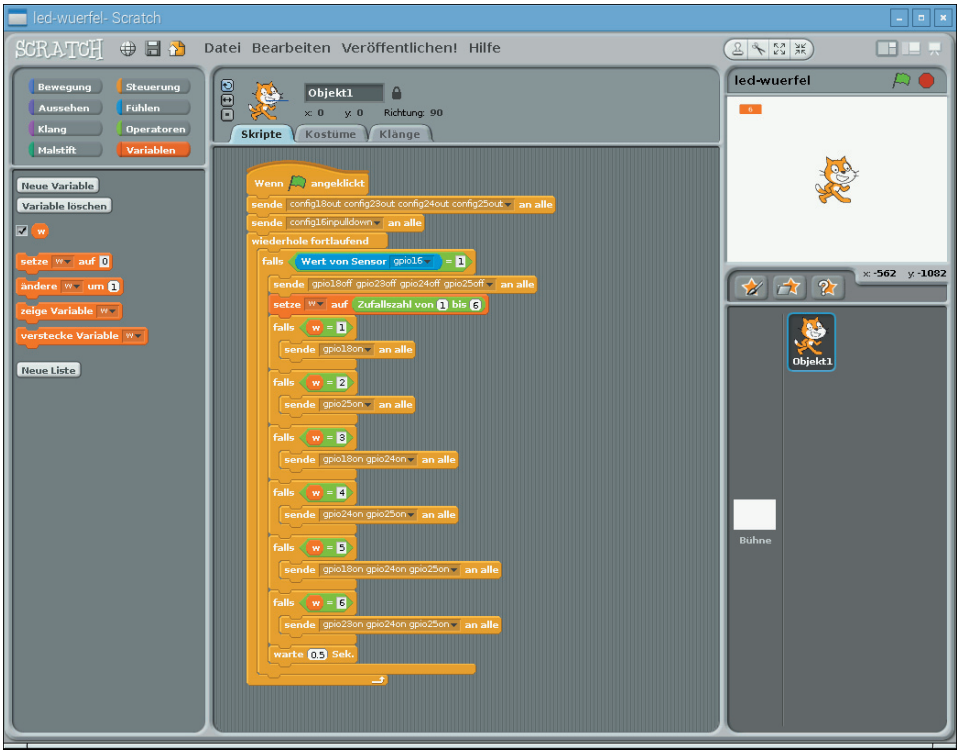


LED-Würfel mit Taster auf einer Steckplatine

Programmdatei:
led-wuerfel.sb

6.1 | Würfeln mit Scratch

Das Scratch-Skript `led-wuerfel` läuft insoweit ähnlich wie die Fußgängerampel, als dass ein Tastendruck verschiedene LEDs auslöst. Für das Erzeugen, Speichern und Abfragen der Zufallszahl werden jedoch neue Blöcke verwendet.



Das Scratch-Skript für den LED-Würfel

6.1.1 | So funktioniert das Programm

Wenn der Benutzer auf das grüne Fähnchen klickt, werden als Erstes die fünf verwendeten GPO-Pins initialisiert. Wir verwenden hier die Ports 18, 23, 24, 25 als Ausgänge für die LEDs und den Port 16 als Eingang für den Taster.

Danach beginnt die Endlosschleife, die darauf wartet, dass der Benutzer auf die Taste drückt.

Dazu wird ähnlich wie bei der Fußgängerampel der GPIO-Port 16 abgefragt.

Wenn dieser den Wert `1` hat, die Taste also gedrückt wurde, werden die Anweisungen innerhalb des *falls*-Blocks ausgeführt. Hier werden als Erstes die vier GPIO-Ports mit den LEDs ausgeschaltet. Am Anfang sind die LEDs alle aus, aber später bleibt ein angezeigtes Würfelergebnis so lange bestehen, bis der Benutzer wieder die Taste drückt.

Danach wird eine zufällige Zahl zwischen 1 und 6 erzeugt und in der Variablen `w` gespeichert.

Wie entstehen Zufallszahlen?

Man denkt, in einem Programm kann nichts zufällig passieren, alles ist geplant. Wie kann ein Programm dann in der Lage sein, zufällige Zahlen zu generieren? Teilt man eine große Primzahl durch irgendeinen Wert, ergeben sich ab der x-ten Nachkommastelle Zahlen, die kaum noch vorhersehbar sind. Diese ändern sich auch ohne jede Regelmäßigkeit, wenn man den Divisor regelmäßig erhöht. Dieses Ergebnis ist zwar scheinbar zufällig, lässt sich aber durch ein identisches Programm oder mehrfachen Aufruf desselben Programms jederzeit reproduzieren. Nimmt man jetzt aber eine aus einigen dieser Ziffern zusammengebaute Zahl und teilt sie wiederum durch eine Zahl, die sich aus der aktuellen Sekunde oder dem Inhalt einer beliebigen Speicherstelle des Rechners ergibt, kommt ein Ergebnis heraus, das sich nicht reproduzieren lässt und daher als Zufallszahl bezeichnet wird.

Variablen müssen in Scratch erst einmal angelegt werden. Klicken Sie in der Blockpalette oben auf das orangefarbene Symbol *Variablen* und dann auf *Neue Variable*. Geben Sie der Variablen einen Namen, in unserem Beispiel `w` (wie Würfel).

In der Blockpalette werden verschiedene Blöcke zur Arbeit mit Variablen angezeigt.

Schalten Sie den Schalter links neben der Variablen `w` ein, wird diese Variable automatisch auf der Bühne in einem kleinen orangefarbenen Feld angezeigt. So sehen Sie hier immer die gewürfelte Zahl und können leicht kontrollieren, ob die LEDs funktionieren.

Ziehen Sie jetzt den Block *setze...auf* in das Skript. Solange nur eine Variable definiert ist, ist diese im Block automatisch ausgewählt. Wenn Sie in einem Skript mehrere Variablen haben, müssen Sie im Listenfeld die richtige auswählen.

Ziehen Sie aus der Blockpalette der Operatoren den Block *Zufallszahl von...bis...* auf das Zahlenfeld im orangefarbenen Block *setze...auf*. Tragen Sie dann in die beiden Zahlenfelder eine 1 und eine 6 ein, da die Zufallszahl in diesem Bereich liegen soll.

Nachdem die Zahl gewürfelt wurde, folgen sechs *falls*-Blöcke für jeden möglichen Würfelwert.

Jeder dieser Blöcke schaltet, wenn eine bestimmte Zahl gewürfelt wurde, die entsprechenden LEDs ein.

Ziehen Sie den grünen Block = in das Abfragefeld des *falls*-Blocks.

Ziehen Sie dann den Block der Variablen *w* aus der Blockpalette *Variablen* in das erste Feld des Gleichheitsoperators. In das zweite Feld schreiben Sie eine 1. Jetzt wird der Block innerhalb der Klammer immer dann abgearbeitet, wenn das Würfelergebnis eine 1 war. Innerhalb des *falls*-Blocks setzen Sie einen Block *sende...an alle* und geben dort die entsprechenden GPIO-Ports der LEDs für dieses Würfelergebnis an.

Mit einem Rechtsklick auf den *falls*-Block können Sie diesen duplizieren und brauchen nur noch die Würfelergebnisse und die passenden LEDs dazu ändern.

Unabhängig vom Würfelergebnis wartet das Programm immer, wenn der Benutzer eine Taste gedrückt hat, noch eine halbe Sekunde, um zu vermeiden, dass durch Tastenprellen kurz hintereinander zwei verschiedene Würfelaktionen ausgelöst werden. Danach startet die Endlosschleife neu und ruft je nach ermitteltem Sensorwert die Würfelaktion auf.

6

6.2 | Würfeln mit Python

Das Python-Programm `led-wuerfel.py` stellt den gleichen Würfel in Python dar. Der Schaltungsaufbau und die verwendeten GPIO-Ports entsprechen denen aus dem Scratch-Programm.

Programmdatei:
led-wuerfel.py

6.2.1 | So funktioniert das Programm

Das Programm enthält viele bekannte Elemente. Wir gehen hier nur auf die neuen ein.

```
001 #!/usr/bin/python
002 # -*- coding: utf-8 -*-
003 import RPi.GPIO as GPIO
004 import time, random
```

Bei der Initialisierung wird zusätzlich das Modul `random` importiert, das die Möglichkeit bietet, Zufallszahlen zu erzeugen.

```
006 GPIO.setmode(GPIO.BCM)
007 LED = [18,25,24,23]
008 TASTE = 16
```

Die Nummerierung der GPIO-Ports wird wie in den vorherigen Beispielen auf BCM gesetzt und im Array `LED` die Nummern für die mittlere LED sowie die drei LED-Paare definiert. Der GPIO-Port des Tasters mit eingeschaltetem Pulldown-Widerstand wird in der Konstanten `TASTE` definiert.

```
010 for x in range(4):
011     GPIO.setup(LED[x], GPIO.OUT)
013 GPIO.setup(TASTE, GPIO.IN, pull_up_down = GPIO.PUD_DOWN)
014 print ("Knopf drücken, um zu würfeln. Strg+C beendet das Programm")
```

Die GPIO-Ports der LEDs werden als Ausgänge definiert, der GPIO-Port des Tasters als Eingang. Danach erscheint für den Benutzer ein kurzer Hinweis auf dem Bildschirm, wie das Programm funktioniert.

```
016 try:
017   while True :
018     if GPIO.input(TASTE)==1:
```

Jetzt startet die Endlosschleife, die darauf wartet, dass der Benutzer den Taster drückt.

```
019     for x in range(4):
020       GPIO.output(LED[x], 0)
021     time.sleep(0.5)
```

Hat er dies getan, werden alle LEDs ausgeschaltet und es wird 0,5 Sekunden gewartet.

```
022     w = random.randrange(1,7)
023     print "Gewürfelte Zahl:" + str(w)
```

Danach wird eine zufällige Zahl zwischen 1 und 6 erzeugt und auf dem Bildschirm angezeigt. Diese Anzeige kann auch weggelassen werden.

```
024     if w == 1:
025       GPIO.output(LED[0], 1)
026     if w == 2:
027       GPIO.output(LED[1], 1)
028     if w == 3:
029       GPIO.output(LED[0], 1)
030       GPIO.output(LED[2], 1)
```

Abhängig von der gewürfelten Zahl werden bestimmte LED-Gruppen eingeschaltet.

```
042 print ("Knopf drücken, um zu würfeln. Strg+C beendet das Programm")
```

Unabhängig vom Würfelergebnis wird der Benutzer wieder aufgefordert, die Taste zu drücken. Danach springt das Programm wieder zum Schleifenanfang. Die LEDs bleiben eingeschaltet, bis der Benutzer die Taste drückt.

7 SCRATCH-KATZE MIT GPIO-TASTEN STEUERN

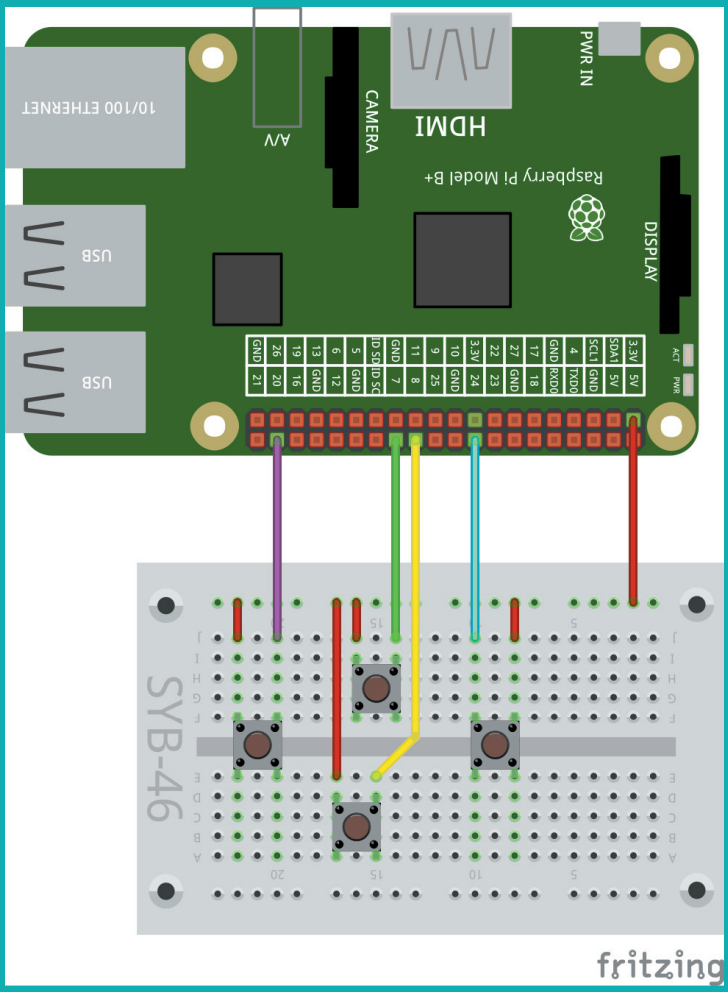
Die Katze, das Scratch-Symbol, kann auf der Bühne laufen und dabei Spuren hinterlassen. Die Katze steht dabei symbolisch für verschiedenartige Objekte, die in Scratch animiert werden können.

Das Scratch-Skript `tasten` zeigt, wie man Scratch-Animationen über GPIO-Tasten steuern kann.

Bauen Sie für das folgende Experiment vier Taster so auf, dass die vier Richtungstasten links, oben, rechts und unten darstellen.

Benötigte Bauteile

1x Steckplatine, 4x Taster, 4x Verbindungskabel, 5x Drahtbrücke (unterschiedliche Längen)

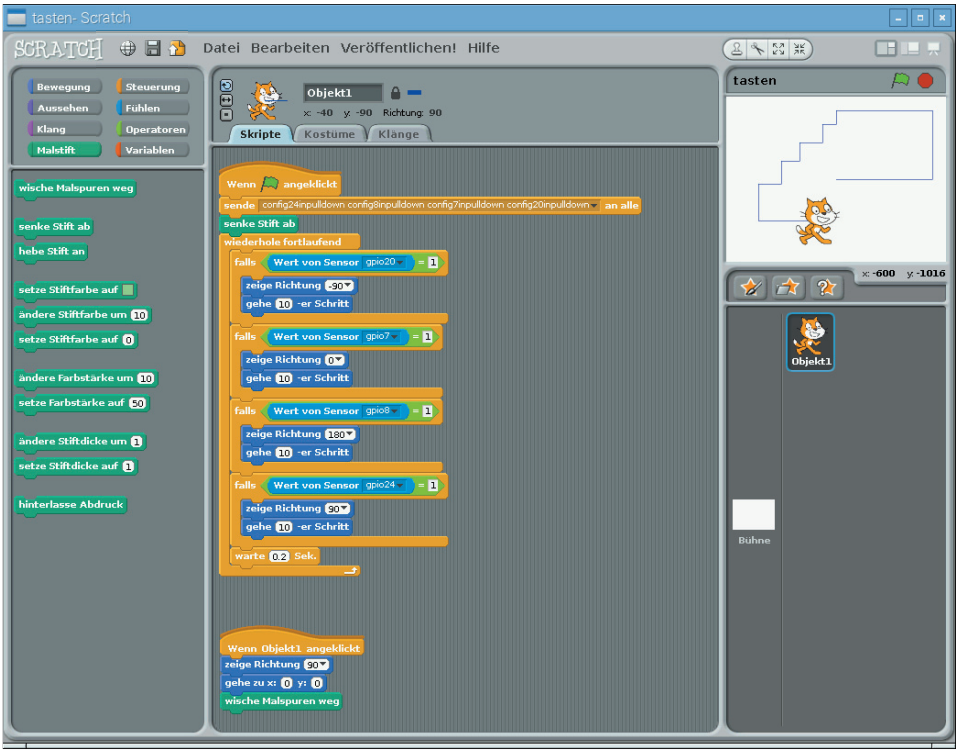


Vier Taster mit Widerständen auf einer Steckplatine

Die vier Tasten sollen die Katze in den vier Richtungen jeweils einen Schritt bewegen. Dabei sollen die Bewegungen als Linien aufgezeichnet werden. Auf diese Weise kann man verfolgen, wie sich die Katze bewegt hat, oder auch gezielt bestimmte Grafiken zeichnen.

Programmdatei:

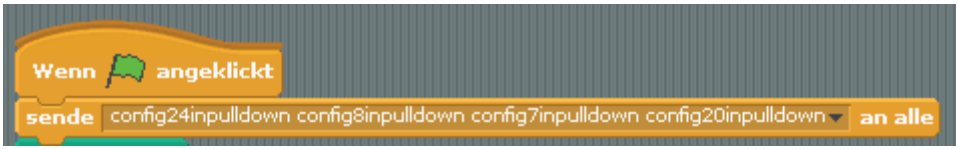
tasten.sb



Die Katze bewegt sich gesteuert von GPIO-Tasten.

7.1 | So funktioniert das Programm

Das Skript fragt in einer Endlosschleife die vier Tasten ab. Wurde eine davon gedrückt, dreht sich die Katze in die entsprechende Richtung und geht einen 10er-Schritt.

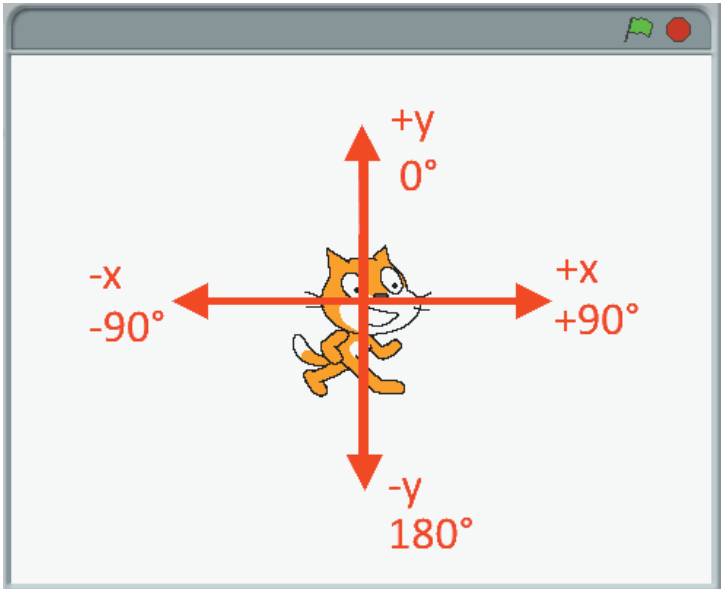


Wenn der Benutzer auf das grüne Fähnchen klickt, werden als Erstes die vier verwendeten GPO-Pins initialisiert. Wir verwenden hier die Ports 24, 8, 7 und 20 als Eingänge für die Taster mit aktivierten internen Pulldown-Widerständen.

Jetzt wird der Malstift abgesenkt, was bedeutet, dass ab jetzt alle Bewegungen eine Spur auf der Bühne hinterlassen. Klicken Sie dazu in der Blockpalette auf das türkisgrüne Symbol *Malstift* und ziehen Sie dann den Block *senke Stift ab* in das Programm.

Danach beginnt die Endlosschleife, die darauf wartet, dass der Benutzer auf eine der Tasten drückt. Jede Taste wird über einen *falls*-Block einzeln abgefragt.

Wurde die Taste am GPIO-Port 20 (nach links) gedrückt, dreht sich die Katze in die Richtung -90 Grad, was im Scratch-Koordinatensystem nach links bedeutet.



Hier werden absolute Richtungen angegeben, da die Ausrichtung der Katze vor dem Drücken der Taste nicht gespeichert wird. Nach der Drehung geht die Katze einen 10er-Schritt.

7

Auf die gleiche Weise werden die anderen drei Tasten abgefragt, die Katze entsprechend gedreht und bewegt.

Jetzt wartet das Programm noch 0,2 Sekunden, um zu vermeiden, dass durch Tastenprellen kurz hintereinander zwei Tastendrücke ausgelöst werden. Danach startet die Endlosschleife neu und bewegt die Katze je nach ermitteltem Sensorwert erneut um einen Schritt.

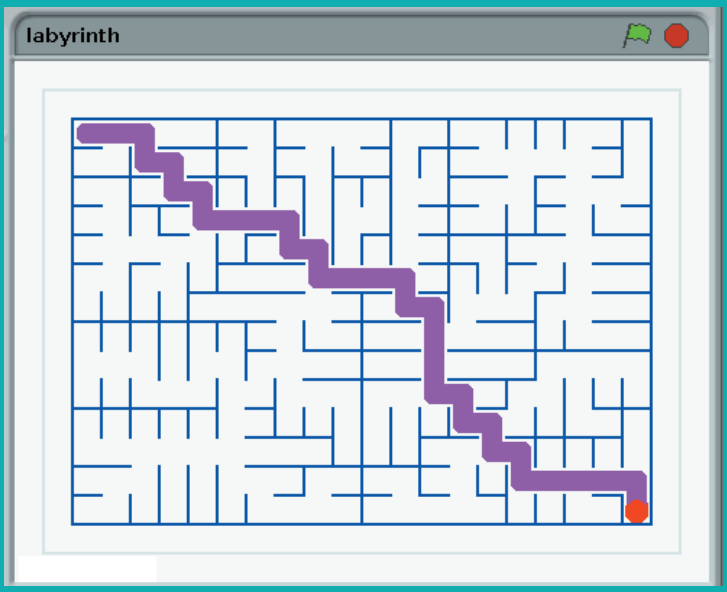
Irgendwann ist der Bildschirm voller Linien und die Katze möglicherweise dabei, am Rand der Bühne zu verschwinden. Für diesen Fall legen Sie im Skriptbereich noch einen zweiten Programmblock an, der mit dem Hauptprogramm nicht verbunden ist und beim Klick auf das grüne Fähnchen auch nicht automatisch gestartet wird.

Dieses Programm wird nicht durch das grüne Fähnchen, sondern durch Klick auf die als *Objekt1* bezeichnete Katze gestartet. Es stellt den Anfangszustand wieder her, indem es zunächst die Katze in die Standardrichtung nach rechts dreht und sie noch auf die Koordinaten *x:0, y:0* im Zentrum der Bühne zurücksetzt. Zuletzt werden alle Malspuren gelöscht.

Dieses Programm kann jederzeit durch Klick auf die Katze ausgelöst werden. Das Hauptprogramm braucht nicht beendet zu werden und läuft weiter. Sie können also nach dem Zurücksetzen der Katze sofort wieder eine Taste drücken und die Katze weiterlaufen lassen.

8 WEG DURCH EIN LABYRINTH

Seit Jahrtausenden faszinieren Labyrinthe und Irrgärten unterschiedlichster Formen die Menschen. Besonderes Interesse löst diese Form der Grafik bei Künstlern, aber auch bei Mathematikern und Logikern aus. Im Computerzeitalter stellen sowohl das Gestalten wie auch das Lösen solcher Labyrinthe immer wieder interessante Herausforderungen an Programmierer.

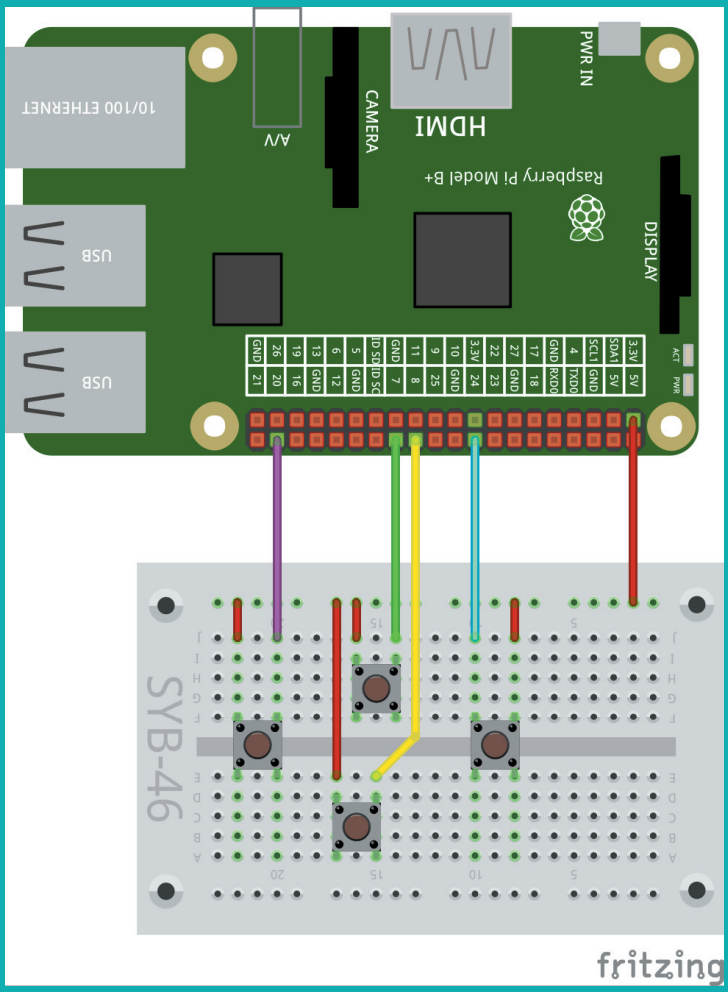


Zufällig generiertes Labyrinth und der Weg hindurch

Wir verwenden für dieses Experiment die gleiche Schaltung wie beim vorhergehenden Experiment.

Benötigte Bauteile

1x Steckplatine, 4x Taster, 6x Verbindungskabel, 6x Drahtbrücke (unterschiedliche Längen)



Vier Taster mit Widerständen auf einer Steckplatine

Ein Scratch-Projekt liefert einen ersten Einblick in die Geometrie und Logik von Labyrinthen. Das Programm generiert im ersten Schritt ein zufälliges Labyrinth. Anschließend kann man mit vier Tasten den roten Punkt durch das Labyrinth bewegen, ohne an den Wänden anzustoßen.

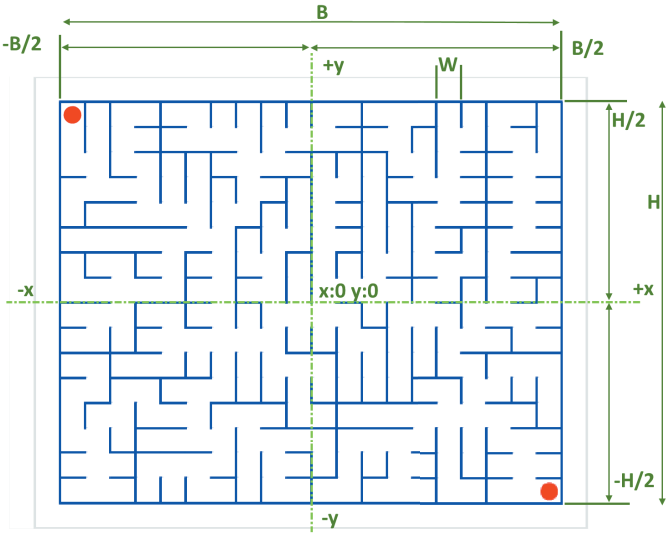
Programmdatei:
labyrinth.sb

8

Es gibt immer genau einen Weg durch das Labyrinth

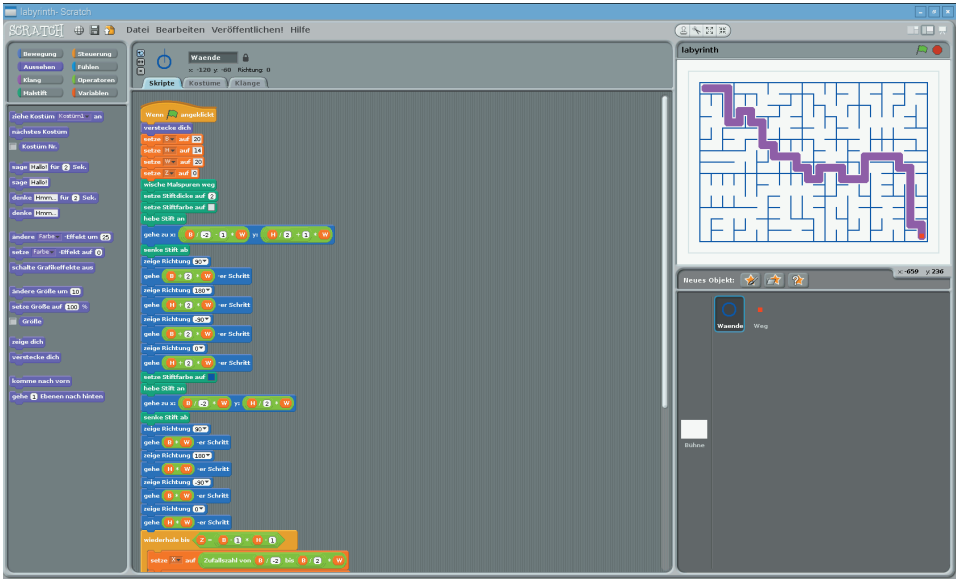
Jedes Labyrinth sieht etwas anders aus und durch jedes gibt es genau einen Weg von links oben nach rechts unten. Dieser garantierte Weg basiert auf einem einfachen Prinzip. Beim Generieren des Labyrinths wird an den Rändern angefangen. Von einem zufälligen Punkt auf einer Wand wird ein neues Wandsegment in den freien Raum gezogen. Dieses darf nichts an einer bestehenden Wand ändern. So wird nacheinander immer zufällig ein Wandpunkt (im Raster des Labyrinths) ausgewählt und von dort werden Wände auf angrenzende freie Rasterpunkte gezogen – so lange, bis alle Rasterpunkte belegt sind. Dadurch, dass nie eine Wand zwischen zwei Wänden gezogen wird, kann es keine durchgehende Sperre geben, die einen Lösungsweg verhindert. Umgekehrt fängt keine Wand an einem freien Punkt im Raum an, sodass es auch keine Inseln geben kann, woraus sich zwei unterschiedliche Wege ergeben würden.

Die Scratch-Bühne hat zwar eine genau definierte Größe, trotzdem ist das Programm `labyrinth` durch die Verwendung von Variablen so allgemein gehalten, dass es theoretisch auf unterschiedlichen Bühnengrößen laufen könnte. Außerdem verliert man bei der Berechnung mit Variablen nicht so schnell die Übersicht wie bei absoluten Zahlen, bei denen man irgendwann nicht mehr weiß, wie sie sich zusammensetzen.



Das Koordinatensystem des Labyrinths

Zur Beschreibung bestimmter Punkte im Labyrinth werden die Variablen *B* (= Breite), *H* (= Höhe) und *W* (= Wegbreite) verwendet. Die Wegbreite legt das Raster des Labyrinths fest.



Das Programm `labyrinth` *generiert ein Labyrinth in Scratch*

8.1 | So funktioniert das Programm

Im Programm verwenden wir zwei einfache Objekte: Eines zeichnet die Wände und ist selbst nicht sichtbar, mit dem anderen bewegt man sich später durch das Labyrinth und zeichnet dabei den Weg.

Beim Klick auf das grüne Fähnchen wird das Objekt *Waende* versteckt, da es beim Zeichnen nur stört und deshalb besser unsichtbar genutzt wird. Dann werden vier Variablen festgelegt:

- *B* bezeichnet die Breite des Labyrinths, gezählt in Wegbreiten, nicht in Koordinateneinheiten.
- *H* bezeichnet die Höhe des Labyrinths, ebenfalls gezählt in Wegbreiten.
- Die Wegbreite *W* legt das Raster des Labyrinths fest.
- Der Zähler *Z* zählt mit, wie viele Rasterpunkte bereits an das Labyrinth angeschlossen sind, und stellt so fest, wann es fertig gezeichnet ist.

8

Jedes Scratch-Objekt kann bei seiner Bewegung Linien zeichnen, wenn der sogenannte Malstift abgesenkt ist. Solange er angehoben ist, hinterlassen Objekte keine Malspuren.

Als Erstes zeichnen wir ein hellgraues Rechteck mit einem Abstand von einer Wegbreite um das eigentliche Labyrinth herum. Dies wird später verhindern, dass beim Erzeugen des Labyrinths Mauersegmente außen am Labyrinth angebaut werden. Dazu wird die Stiftfarbe auf Hellgrau gesetzt und der Stift zunächst ausgeschaltet, um die Figur an die linke obere Ecke dieses Rechtecks zu bringen.

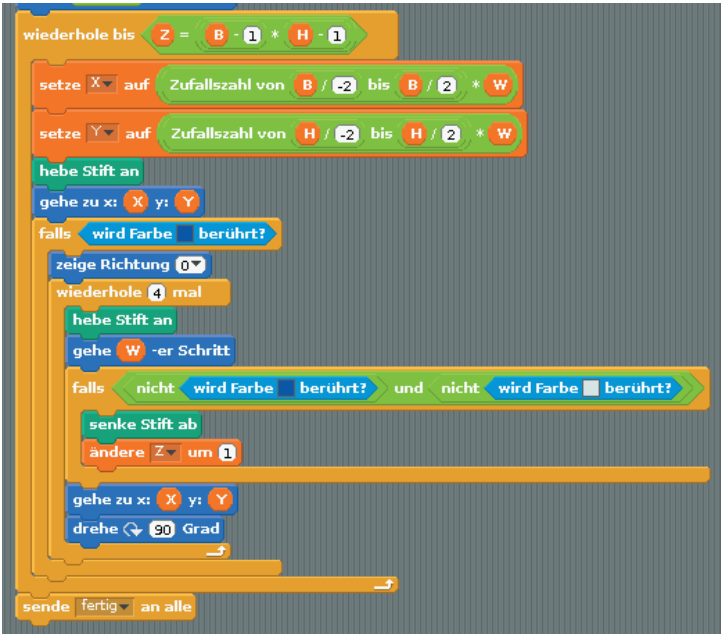
Anschließend werden die vier Seiten des Rechtecks gezeichnet.

Als Nächstes zeichnen wir den Umriss des Labyrinths in Blau. Dies funktioniert ganz ähnlich, da es auch ein Rechteck ist, dessen Mittelpunkt im Nullpunkt des Koordinatensystems liegt.

Jetzt kommt der eigentlich interessante Programmteil, der das Labyrinth aufbaut. Da wir, bedingt durch den Zufallsgenerator, vorher nicht wissen, wie lange es dauert, das Labyrinth aufzubauen, verwenden wir

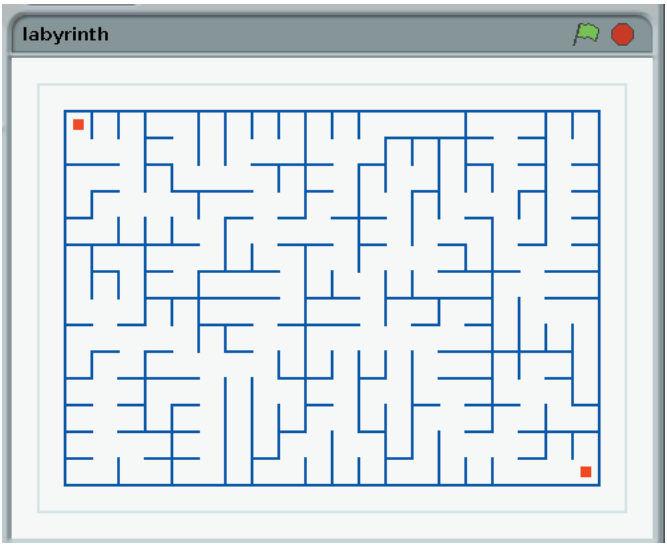
eine *wiederhole bis…*-Schleife, die so lange läuft, bis alle inneren Rasterpunkte des Rechtecks an das Netz der Labyrinthmauern angeschlossen sind. Es gibt `B-1 * H-1` Rasterpunkte innerhalb des Rechtecks, die die Variable *Z* zählt.

In jedem Schleifendurchlauf wird ein Punkt zufällig ermittelt, von dem aus neue Labyrinthmauern gebaut werden. Voraussetzung ist natürlich, dass der Punkt bereits an einer Mauer liegt, da Mauern nach den Baueregeln nicht frei im Raum beginnen können. Die Koordinaten dieses zufällig ermittelten Punktes werden in den Variablen *X* und *Y* gespeichert.



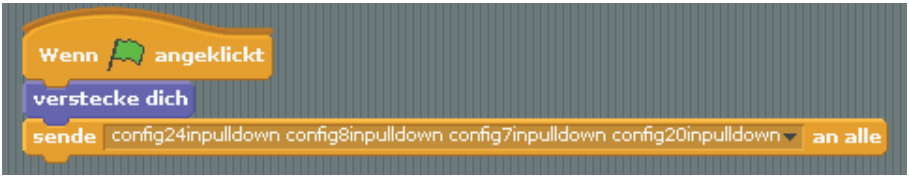
Liegt der zufällig ermittelte Punkt auf einer Mauer, können von dort neue Mauern gezeichnet werden. Um das zu überprüfen, verwenden wir den Block *wird Farbe… berührt*. Wenn das unsichtbare Objekt, das die Mauern zeichnet, die blaue Farbe berührt, mit der die Mauern gezeichnet werden, steht es auf einem Rasterpunkt mit einer Mauer. Das Objekt ist klein genug, um keine Nachbarmauer auf dem nächsten Rasterpunkt zu berühren. Es kann auch bei der hier verwendeten Programmlogik nie zwischen zwei Rasterpunkten stehen.

Trifft die Bedingung zu, versucht das Programm, innerhalb der Schleife möglichst viele Mauern von dem ermittelten Punkt aus in die vier Richtungen zu zeichnen. Es können aber höchstens drei Mauern neu gezeichnet werden, da ein Punkt nur als Anfangspunkt verwendet wird, wenn er bereits an einer Mauer liegt. Die vier möglichen Richtungen werden mit einer weiteren Schleife abgearbeitet.



Das fertig erstellte Labyrinth

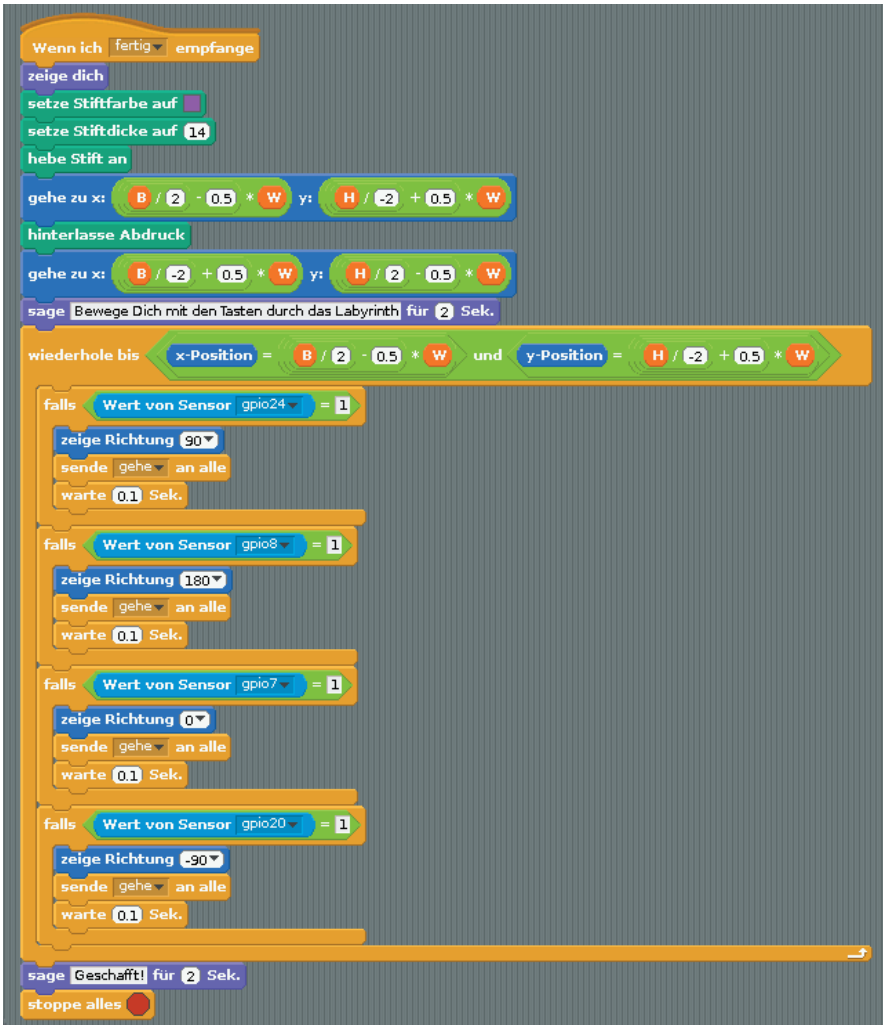
Um durch das Labyrinth zu laufen, verwenden wir ein zweites Objekt *Weg*, das mit vier Tastern in den vier Richtungen gesteuert wird. Beim Klick auf das grüne Fähnchen wird dieses Objekt zunächst versteckt, bis das Labyrinth fertig ist. Zusätzlich werden die vier GPIO-Eingänge für die Taster initialisiert.



Das Objekt soll erst auftauchen, wenn das Labyrinth fertig gezeichnet ist. Dazu brauchen wir Blöcke, die es einem Objekt möglich machen, an

ein anderes eine Nachricht zu senden. Bisher galten alle Programmblöcke immer nur für ein Objekt. Das Programm des Objekts *Waende* enthält dazu am Ende einen Block *sende fertig an alle* – der gleiche Blocktyp, der auch für die GPIO-Steuerung verwendet wird.

Das Hauptprogramm zum Bewegen des Objekts *Weg* durch das Labyrinth startet, wenn diese Nachricht empfangen wird.



8

Am Anfang zeigt sich das versteckte Objekt wieder, setzt Stiftfarbe und -dicke, hinterlässt in der unteren rechten Ecke einen Abdruck und startet dann in der linken oberen Ecke des Labyrinths. Dabei wird mit einem *sage... für... Sek*-Block eine kurze Erklärung in einer Sprechblase angezeigt.

Die Aktionen, die durch die Taster ausgelöst werden, unterscheiden sich nur in der Richtung, sind sonst aber gleich. Das Drücken eines Tasters legt eine Richtung fest und löst anschließend über einen *sende gehe an alle*-Block eine Bewegung um einen Rasterschritt aus.

Als Erstes geht das Objekt einen halben Schritt weit in die durch den Taster festgelegte Richtung. Würde das Objekt einen ganzen Schritt im Labyrinthraster gehen, würde es auf jeden Fall auf dem benachbarten Wegefeld landen. Durch den Trick mit dem halben Schritt gibt es zwei Möglichkeiten:

- Das Objekt kommt mitten in einer Wand zum Stehen.
- Das Objekt kommt auf einem Weg zwischen zwei Rasterfeldern zum Stehen.

Diese beiden möglichen Ergebnisse haben natürlich völlig unterschiedliche Folgen.

Berührt das Objekt nach dem halben Schritt die blaue Farbe einer Mauer, ist es auf dem falschen Weg. Es dreht sich um 180 Grad und geht den halben Schritt wieder zurück an seinen ursprünglichen Platz.

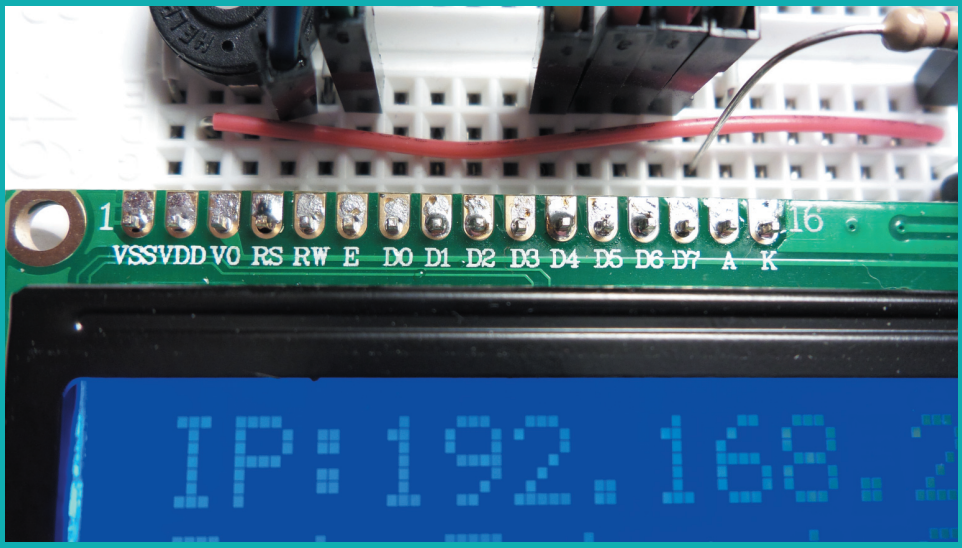
Im anderen Fall, wenn das Objekt keine blaue Farbe berührt, also auf dem Weg zwischen zwei Rasterfeldern steht, soll es auf das nächste Rasterfeld laufen und dabei eine Spur hinterlassen. Da es auf dem ersten halben Schritt keine Spur hinterlassen hat, geht es den halben Schritt wieder zurück, wie dies auch beim Berühren der Mauer passiert, und schaltet dann, am ursprünglichen Platz angekommen, den Stift ein. Dann dreht die Richtung wieder um 180 Grad und das Objekt bewegt sich mit eingeschaltetem Stift einen ganzen Schritt auf das nächste Rasterfeld.

Die Abfrage der vier Taster wird so lange wiederholt, bis das rote Objekt die untere rechte Ecke des Labyrinths erreicht hat. Diese Bedingung steht in der *wiederhole bis*-Schleife.

Zum Schluss liefert ein *sage... für... Sek*-Block eine Erfolgsmeldung, danach werden alle Programmaktivitäten beendet.

9 ERSTES EXPERIMENT MIT DEM LC-DISPLAY

Die nächsten Experimente zeigen, wie Sie mit Scratch und Python Texte auf dem LC-Display darstellen können. Zuerst müssen Sie den beiliegenden Pfostenverbinder, wie auf Seite 21 beschrieben, am Display anlöten.



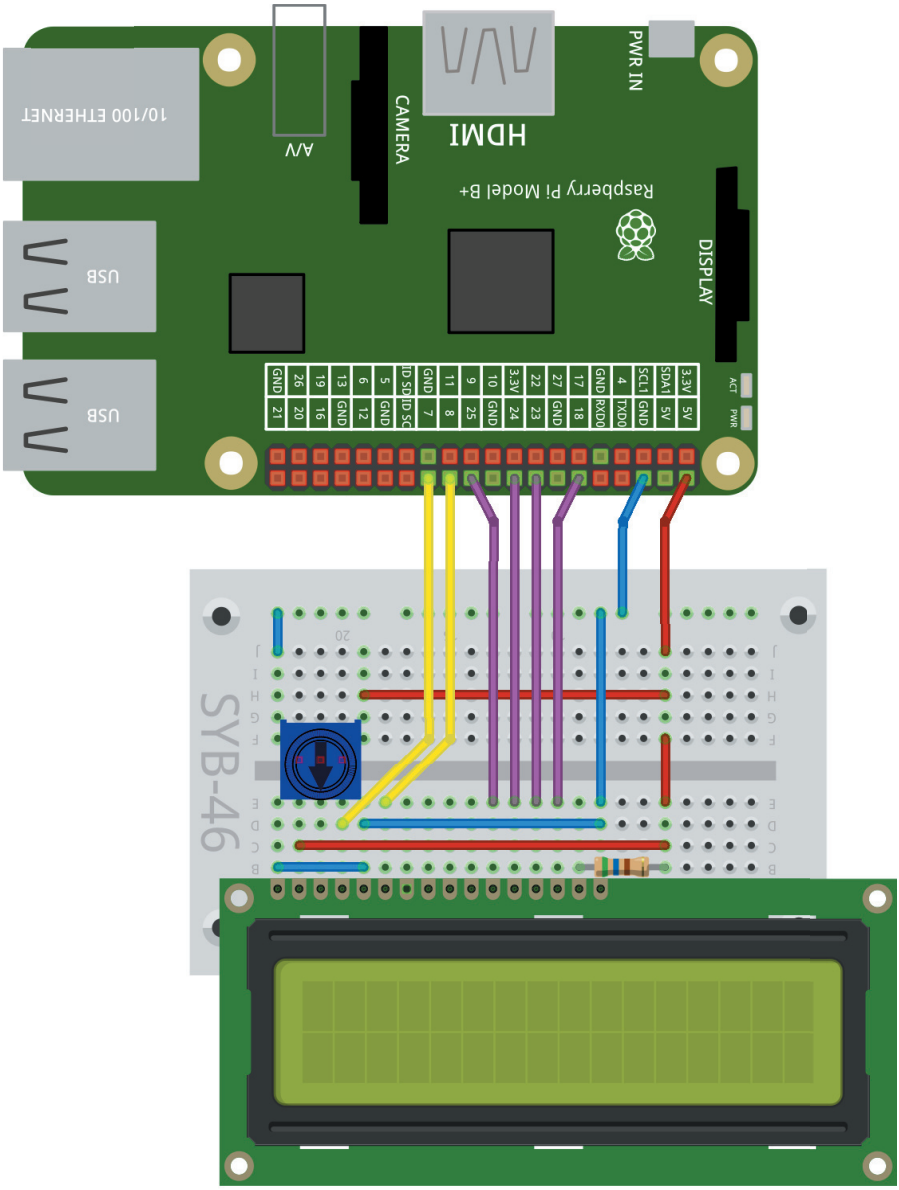
Die 16 Anschlusspins des LC-Displays

9.1 | Pinbelegung eines HD44780-kompatiblen Displays

Pin	Funktion	Beschreibung
1	VSS	Stromversorgung Masseleitung 0 V
2	VDD	Stromversorgung +5 V
3	V0	Kontrasteinstellung, 0 V ... 5 V
4	RS	Register Select
5	RW	Read/Write, wenn vom Display nichts ausgelesen wird, mit 0 V verbinden
6	E	Enable (Umschaltsignal)
7	D0	Datenbit 0 (im 4-Bit-Modus nicht benötigt)
8	D1	Datenbit 1 (im 4-Bit-Modus nicht benötigt)
9	D2	Datenbit 2 (im 4-Bit-Modus nicht benötigt)
10	D3	Datenbit 3 (im 4-Bit-Modus nicht benötigt)
11	D4	Datenbit 4
12	D5	Datenbit 5
13	D6	Datenbit 6
14	D7	Datenbit 7
15	A	Hintergrundbeleuchtung, 560-Ohm-Vorwiderstand erforderlich
16	K	Hintergrundbeleuchtung Masseleitung

Benötigte Bauteile

1x Steckplatine, 1x LC-Display, 1x 560-Ohm-Widerstand (Grün-Blau-Braun), 1x Potenziometer, 8x Verbindungskabel, 7x Drahtbrücke (unterschiedliche Längen)



Gelb: Steuerleitungen, violett: Datenleitungen, rot: +5 V, blau: Masse

Displaypin	Signal	Raspberry-Pi-GPIO
1	VSS	0 V
2	VDD	+5 V
3	V0	Potenziometer
4	RS	GPIO 25
5	RW	0 V
6	E	GPIO 24
7	D0	im 4-Bit-Modus nicht benötigt
8	D1	im 4-Bit-Modus nicht benötigt
9	D2	im 4-Bit-Modus nicht benötigt
10	D3	im 4-Bit-Modus nicht benötigt
11	D4	GPIO 23
12	D5	GPIO 17
13	D6	GPIO 27
14	D7	GPIO 22
15	A	Vorwiderstand 560 Ohm
16	K	0 V

Zusätzlich zu den Verbindungskabeln zum Raspberry Pi benötigen Sie Drahtbrücken unterschiedlicher Länge, mit denen mehrere Pins für die gemeinsame Stromversorgung und Masseleitung zusammengeführt werden. Schneiden Sie diese vom mitgelieferten Schaltdraht ab und entfernen Sie die Isolierung an beiden Enden auf einer Länge von etwa 5 mm.

Vor der LED der Hintergrundbeleuchtung (Pin 15) wird ein 560-Ohm-Widerstand als Schutz gegen Überlastung vorgeschaltet. Dieser muss höher sein als die üblichen LED-Vorwiderstände, da das LCD-Modul mit 5 V und nicht wie andere Schaltungen am Raspberry Pi mit 3,3 V betrieben wird. Zur Kontrastregelung bauen Sie ein 15-kOhm-Potenziometer ein, das dem Pin 3 eine Spannung zwischen +5 V und 0 V zuführt. Das Potenziometer muss so eingebaut werden, dass der mittlere Kontakt in der Reihe 21 in der unteren Hälfte des Steckbretts steckt, die beiden äußeren Kontakte in den Reihen 19 und 23 in der oberen Hälfte.

9

Schaltung genau überprüfen

Bevor Sie den Raspberry Pi und damit die Schaltung mit Strom versorgen, prüfen Sie alle Anschlussdrähte noch einmal genau. Eine falsch angeschlossene +5 V-Leitung kann das LCD-Modul und auch den Raspberry Pi beschädigen. Die maximale Spannung am Kontrast-Pin darf nur weniger als +5 V betragen. Drehen Sie also das Potenziometer nie ganz auf.

Das Display unterstützt zwei Modi zur Übertragung der Daten. Im 8-Bit-Modus kann ein komplettes Zeichen auf einmal übertragen werden. Meistens wird aber der 4-Bit-Modus verwendet, da dieser vier Ports am sendenden Gerät spart. Hier werden die Daten eines Zeichens in zwei Blöcken hintereinander übertragen.

9.2 | LC-Display mit Python ansteuern

Programmdatei:
display01.py

In Python wird das Display sehr hardwarenah, direkt über die GPIO-Ports und entsprechende High- oder Low-Signale angesteuert, was anfangs etwas umständlich wirkt. Dafür lernen Sie die Funktionsweise von LC-Displays genauer verstehen. Das Display wird auch bei diesem Projekt im 4-Bit-Modus betrieben.

Einfache Textdarstellung auf dem LC-Display

Das erste einfache Python-Beispiel `display01.py` zeigt zwei statische Zeichenfolgen auf dem Display. Das hier beschriebene Grundprogramm dient auch als Basis für die weiteren Programmbeispiele mit dem LC-Display.

9.2.1 | So funktioniert das Programm

Dass das Programm funktioniert, lässt sich einfach ausprobieren. Jetzt stellen sich natürlich die Fragen: Was passiert im Hintergrund? Was bedeuten die einzelnen Programmzeilen?

Am Anfang werden bereits bekannte Bibliotheken importiert, um die GPIO-Schnittstelle zu nutzen.

```
006 LCD_RS = 7
007 LCD_E  = 8
008 LCD_D4 = 25
009 LCD_D5 = 24
010 LCD_D6 = 23
011 LCD_D7 = 18
```

Diese Zeilen definieren die Konstanten, in denen die Nummern der GPIO-Ports für Steuerleitungen und Datenleitungen des Displays gespeichert sind.

```
013 GPIO.setmode(GPIO.BCM)
```

Danach wird die Nummerierung der GPIO-Pins auf BCM-Standard gesetzt.

```
014 GPIO.setup(LCD_E,  GPIO.OUT)
015 GPIO.setup(LCD_RS, GPIO.OUT)
016 GPIO.setup(LCD_D4, GPIO.OUT)
017 GPIO.setup(LCD_D5, GPIO.OUT)
018 GPIO.setup(LCD_D6, GPIO.OUT)
019 GPIO.setup(LCD_D7, GPIO.OUT)
```

Die für das Display verwendeten Ports werden alle als Ausgang definiert. Die Portnummern werden über die zuvor definierten Variablen angegeben. Diese Methode hat den Vorteil, dass die tatsächlichen GPIO-Ports nur an dieser einen Stelle im Programm auftauchen. So können Sie das Programm ganz einfach umbauen, wenn Sie andere GPIO-Ports nutzen möchten.

```
021 LCD_WIDTH = 16
022 LCD_LINE_1 = 0x80
023 LCD_LINE_2 = 0xC0
024 LCD_CHR = True
025 LCD_CMD = False
026 E_PULSE = 0.00005
027 E_DELAY = 0.00005
028 INIT = 0.01
```

Diese Zeilen definieren weitere Konstanten, die im Programm verwendet werden.

9

Konstante	Bedeutung
LCD_WIDTH	Breite des Displays in Zeichen
LCD_LINE_1	Speicheradresse der 1. Zeile
LCD_LINE_2	Speicheradresse der 2. Zeile
LCD_CHR	Schaltet das Display auf Pin RS in den Zeichenmodus
LCD_CMD	Schaltet das Display auf Pin RS in den Steuerungsmodus
E_PULSE	Dauer eines Steuerimpulses auf Pin E
E_DELAY	Wartezeit zwischen zwei Steuerimpulsen auf Pin E
INIT	Wartezeit zwischen zwei Zeichen des Initialisierungsstrings

```
030 def lcd_enable():
```

Jetzt wird eine Funktion definiert, die später im Programm aufgerufen wird. Diese sendet einen kurzen Steuerimpuls an den Pin E (*enable*). Damit liest das Display die Daten auf den GPIO-Ports aus, um sie anzuzeigen. Im 4-Bit-Modus wird der gleiche Impuls verwendet, um zwischen den oberen und den unteren Bits eines darzustellenden Bytes umzuschalten.

```
031     time.sleep(E_DELAY)
```

Zuerst wird eine kurze, in `E_DELAY` gespeicherte Zeit (0,05 ms) gewartet. Das Display hat eine gewisse Trägheit. Man kann nicht unmittelbar nach dem letzten Zeichen auf das andere Halbbyte umschalten.

```
032     GPIO.output(LCD_E, True)
```

Jetzt wird auf den Pin E das Logiksignal `True` ausgegeben.

```
033     time.sleep(E_PULSE)
034     GPIO.output(LCD_E, False)
```

Das Signal liegt über die in `E_PULSE` gespeicherte Zeit am Pin E an, danach wird der Pin wieder auf `False` zurückgesetzt. Diesen kurzen Impuls wertet das Display als Steuersignal aus.

```
035     time.sleep(E_DELAY)
```

Das Programm wartet noch die in `E_DELAY` gespeicherte Zeit ab, bevor die Funktion beendet wird, damit danach sofort wieder Daten an das Display gesendet werden können.

```
037 def lcd_byte(bits, mode):
```

Diese Funktion sendet ein Byte an das Display. Dieses Byte kann ein Zeichen oder ein Steuerbefehl sein. Im Parameter `bits` bekommt die Funktion das zu sendende Byte, der Parameter `mode` gibt an, ob es sich um ein Zeichen oder um einen Steuerungsbefehl handelt. Die beiden Werte, die `mode` annehmen kann, sind in den Konstanten `LCD_CHR` (Zeichen) und `LCD_CMD` (Steuerungsbefehl) festgelegt.

```
038     GPIO.output(LCD_RS, mode)
```

Der zu verwendende Modus wird auf den Pin RS (*Register Select*) ausgegeben. Damit wird das Display in den Zeichenmodus oder den Steuerungsmodus geschaltet.

```
039     GPIO.output(LCD_D4, bits&0x10==0x10)
040     GPIO.output(LCD_D5, bits&0x20==0x20)
041     GPIO.output(LCD_D6, bits&0x40==0x40)
042     GPIO.output(LCD_D7, bits&0x80==0x80)
```

Das Programm verwendet den 4-Bit-Modus des Displays. Zunächst werden die oberen vier Bits des zu sendenden Bytes auf den Datenleitungen ausgegeben. Dazu wird der Python-Operator `&` (bitweises UND) verwendet. Nacheinander wird für jedes der vier Bits geprüft, ob es `1` oder `0` ist. Das Ergebnis der Prüfung ist entsprechend `True` oder `False`. Dieser Wert wird dann auf der jeweiligen Datenleitung ausgegeben.

Bit-Nr.	4 high	3 high	2 high	1 high	4 low	3 low	2 low	1 low
Pin	D7	D6	D5	D4	D7	D6	D5	D4
Binär	1000 0000	0100 0000	0010 0000	0001 0000	0000 1000	0000 0100	0000 0010	0000 0001
Dezimal	128	64	32	16	8	4	2	1
Hex	0x80	0x40	0x20	0x10	0x08	0x04	0x02	0x01

```
043     lcd_enable()
```

Diese zuvor definierte Funktion liest die Daten von den GPIO-Ports und schaltet dann das Display auf das untere Halbbyte um.

```
044   GPIO.output(LCD_D4, bits&0x01==0x01)
045   GPIO.output(LCD_D5, bits&0x02==0x02)
046   GPIO.output(LCD_D6, bits&0x04==0x04)
047   GPIO.output(LCD_D7, bits&0x08==0x08)
```

Nach dem gleichen Schema werden danach die unteren vier Bits des zu sendenden Bytes auf den Datenleitungen ausgegeben ...

```
048   lcd_enable()
```

... und anschließend wieder auf das obere Halbbyte umgeschaltet.

```
050 def lcd_string(message):
```

Diese Funktion schreibt eine Zeichenfolge auf das Display. Diese Zeichenfolge wird vom aufrufenden Programm im Parameter `message` an die Funktion übergeben.

```
051   message = message.ljust(LCD_WIDTH," ")
```

Die Zeichenkette wird bis auf die Länge des Displays rechts mit Leerzeichen aufgefüllt. Die Länge des Displays, in unserem Fall 16 Zeichen, ist in der Konstanten `LCD_WIDTH` gespeichert. Die Methode `ljust` kann in jeder Zeichenkette angewendet werden, der zweite Parameter gibt das Zeichen an, mit dem kürzere Zeichenketten aufgefüllt werden. Längere Zeichenketten werden automatisch auf die angegebene Länge abgeschnitten.

```
052   for i in range(LCD_WIDTH):
053     lcd_byte(ord(message[i]),LCD_CHR)
```

Diese Schleife läuft so oft durch, wie das Display Zeichen hat. In jedem Durchlauf wird ein Zeichen der Zeichenkette `message` zunächst mit der Python-Funktion `ord()` in seinen ASCII-Zahlenwert umgewandelt und dann im Zeichenmodus `LCD_CHR` über die zuvor definierte Funktion `lcd_byte()` auf das Display ausgegeben.

Nachdem die drei Funktionen `lcd_enable()`, `lcd_byte()` und `lcd_string()` definiert sind, startet jetzt das eigentliche Programm.

```
055 LCD_INIT = [0x33, 0x32, 0x28, 0x0C, 0x06, 0x01]
```

Bevor das Display verwendet werden kann, muss es zunächst initialisiert werden. Eine Folge von Initialisierungskommandos bringt es in einen eindeutig definierten Zustand und setzt es auf 4-Bit-Modus.

```
056 for i in LCD_INIT:
057   lcd_byte(i, LCD_CMD)
058   time.sleep(INIT)
```

Diese Schleife schreibt nacheinander die in der Liste `LCD_INIT` eingetragenen Initialisierungskommandos im Steuerungsmodus `LCD_CMD` auf das Display. Zur Datenübertragung wird die weiter oben definierte Funktion `lcd_byte()` verwendet. Nach jedem übertragenen Byte wartet das Programm die in der Konstanten `INIT` gespeicherten 0,01 Sekunden, bevor das nächste Byte übertragen wird. Diese Wartezeit ist nur auf dem Raspberry Pi 3 notwendig. Ältere Modelle sind langsam genug, um die Zeichen nur mit einer Geschwindigkeit zu übertragen, die die Displayinitialisierung verarbeiten kann.

Nachdem alles definiert und initialisiert ist, kann es wirklich losgehen:

```
060 lcd_byte(LCD_LINE_1, LCD_CMD)
061 lcd_string("0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ")
```

Zuerst wird die Adresse der oberen Zeile des Displays im Steuerungsmodus `LCD_CMD` auf das Display ausgegeben. Damit wird festgelegt, dass die folgenden Ausgaben im Zeichenmodus auf der oberen Zeile erscheinen. Anschließend schreibt die Funktion `lcd_string()` die erste Zeichenkette auf das Display. Dabei spielt die Länge keine Rolle, da die Zeichenketten von der Funktion `lcd_string()` automatisch auf eine Länge von jeweils 16 Zeichen gebracht werden.

```
062 lcd_byte(LCD_LINE_2, LCD_CMD)
063 lcd_string("www.franzis.de")
```

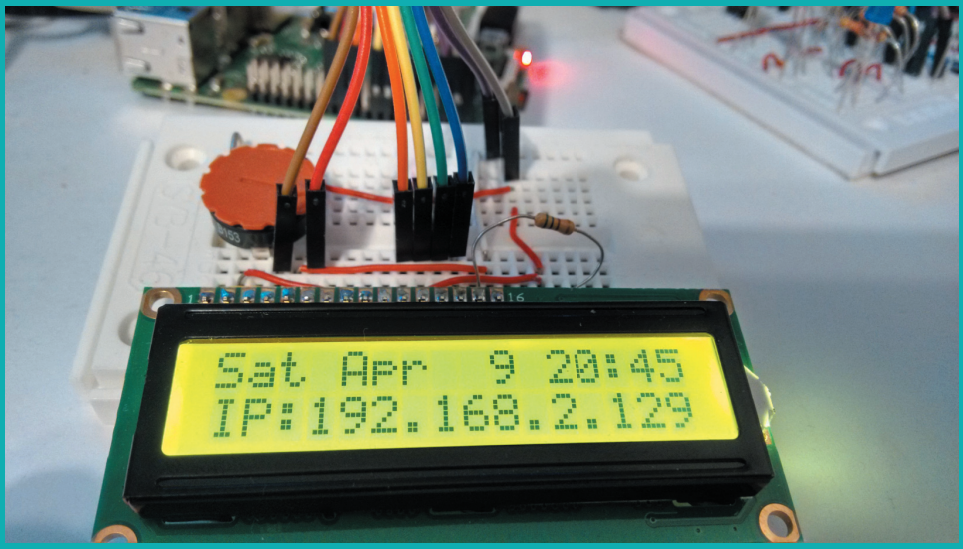
Auf die gleiche Weise wird anschließend die zweite Zeichenkette in die untere Zeile des Displays geschrieben.

```
064 GPIO.cleanup()
```

Danach ist das Programm auch schon zu Ende. Als Letztes muss, wie in allen Python-Programmen, die GPIO-Schnittstelle geschlossen werden, um Warnungen beim nächsten Start eines Programms zu verhindern. Nach Abschluss bleibt die Anzeige auf dem Display aber trotzdem bestehen. Das Display hat keinen ständigen Kontakt zu den GPIO-Ports, sondern liest die Daten nur aus, wenn das Enable-Signal umgeschaltet wird. Danach bleibt die Anzeige im Display wieder unverändert, unabhängig davon, ob Daten an den Datenleitungen anliegen.

10 IP-ADRESSE DES RASPBERRY PI ANZEIGEN

Betreibt man einen Raspberry Pi headless ohne Monitor, ist es nicht immer ganz einfach, die IP-Adresse herauszufinden, die man für eine SSH-Verbindung zur Administration des Servers benötigt. Mit dem LC-Display und einem Python-Skript lässt sich diese Aufgabe elegant lösen.



Das LC-Display zeigt über ein Python-Skript die aktuelle Zeit und die IP-Adresse an.

10.1 | So funktioniert das Programm

Das Programm `status.py` übernimmt die Funktionen zur Displaysteuerung aus dem weiter oben beschriebenen Programm `display01.py`. Auch der Schaltungsaufbau und die verwendeten GPIO-Ports sind die gleichen. Im Folgenden werden nur die neuen Programmelemente beschrieben.

Programmdatei:
status.py

```
001 #!/usr/bin/python
002 import RPi.GPIO as GPIO
003 import time, subprocess
```

Ganz am Anfang wird zusätzlich die Bibliothek `subprocess` importiert, mit der man Linux-Kommandos aus Python-Programmen heraus aufrufen und auswerten kann.

```
028 pause = 2
```

Bei den Variablendeklarationen wird zusätzlich eine Variable `pause` eingeführt, die das Aktualisierungsintervall der Uhrzeit festlegt. Der Wert `2` bedeutet, dass die Uhrzeit, die nur auf Minuten genau angezeigt wird, alle zwei Sekunden aktualisiert wird. Dies ist ein guter Kompromiss zwischen Genauigkeit und Systemauslastung.

Die Funktionen zum Zugriff auf das Display wie auch die Initialisierungssequenz werden unverändert übernommen.

Anstatt einer einfachen statischen Textausgabe wird jetzt eine Endlosschleife verwendet, die immer wieder die Uhrzeit und auch die IP-Adresse aktualisiert.

```
060 try:
061   while True:
```

Diese Endlosschleife wird so lange wiederholt, bis die weiter unten bei `except` festgelegte Abbruchbedingung eintritt.

```
062 lcd_byte(LCD_LINE_1, LCD_CMD)
063 lcd_string(time.asctime())
```

Zur Ausgabe von Datum und Uhrzeit wird die Funktion `time.asctime()` aus dem Modul `time` verwendet. Diese liefert Datum und Uhrzeit in einer Zeichenfolge, z. B. `Sat Apr 9 13:14:59 2016`. Die ersten 16 Stellen dieser Zeichenfolge zeigen Wochentag, Monat, Tag, Stunden und Minuten auf dem 16-steligen Display an. Die Ausgabe wird dann über die bereits im Grundprogramm definierte Funktion `lcd_string()` auf dem Display ausgegeben.

```
064 lcd_byte(LCD_LINE_2, LCD_CMD)
065 lcd_string("IP:" + subprocess.check_output(["hostname","-I"])[:-2])
```

10

In der zweiten Zeile wird die IP-Adresse angezeigt. Dafür wird die Funktion `check_output()` aus dem Modul `subprocess` verwendet, die die Ausgabe eines beliebigen Kommandozeilenbefehls als Zeichenkette liefert. Diese Zeichenkette wird bis auf die letzten zwei Zeichen verwendet, das Zeilenendezeichen `\n` der Ausgabe sowie ein Leerzeichen am Ende werden abgeschnitten.

Lange IP-Adressen

Sollten Sie für die Ausgabe der IP-Adressen in Ihrem lokalen Netzwerk mehr als 13 Stellen benötigen (z. B. `192.168.100.123`), lassen Sie die Zeichenfolge `"IP:"` am Anfang weg, weil das Display nur 16 Stellen hat.

Auch diese Zeichenfolge wird dann über die Funktion `lcd_string()` auf dem Display ausgegeben.

```
066     time.sleep(pause)
```

Nach der Ausgabe wartet das Programm die als Pause definierte Zeit (Standard: 2 Sekunden). Danach startet die Schleife neu und zeigt wieder die neue Zeit an.

```
067 except KeyboardInterrupt:
068   GPIO.cleanup()
```

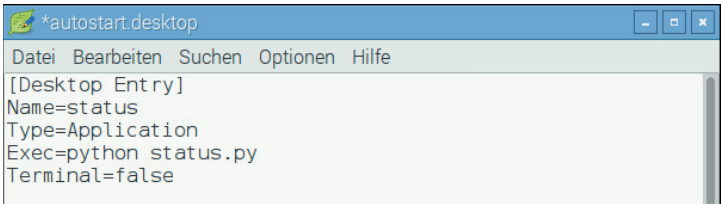
Drückt der Benutzer die Tastenkombination Strg + C, wird ein `KeyboardInterrupt` ausgelöst und die Schleife automatisch verlassen. Die letzte Zeile schließt die verwendeten GPIO-Ports. Danach wird das Programm beendet. Nach dem Beenden des Programms bleibt die Anzeige auf dem Display stehen, die Uhr läuft aber nicht mehr weiter. Die Anzeige selbst wird durch den im Display eingebauten Controller aufrechterhalten.

Durch das kontrollierte Schließen der GPIO-Ports tauchen beim nächsten Start eines Programms, das die GPIO-Schnittstelle nutzt, keine Systemwarnungen oder Abbruchmeldungen auf, die den Benutzer verwirren könnten.

10.2 | Programm automatisch starten

Dieses Programm ist besonders dann nützlich, wenn am Raspberry Pi kein Monitor angeschlossen ist. In diesem Fall kann es automatisch beim Booten gestartet werden.

Legen Sie dazu im Verzeichnis `/home/pi/.config/autostart` eine Textdatei mit Namen `autostart.desktop` an, mit dem abgebildeten Inhalt:



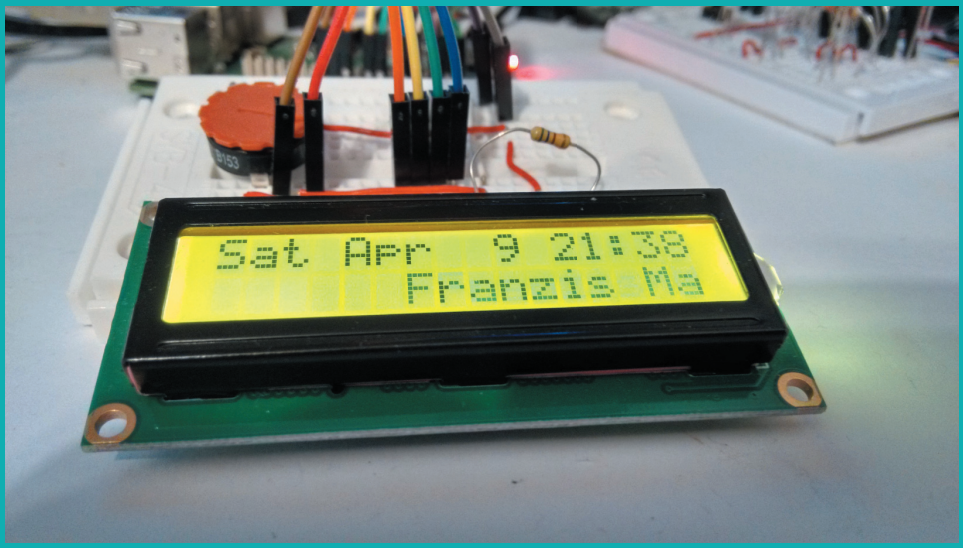
Damit wird kurz nach dem Booten des Raspberry Pi und der Anmeldung als Benutzer `pi` die aktuelle IP-Adresse automatisch auf dem LC-Display angezeigt.

Um das Verzeichnis `.config` im Dateimanager zu sehen, schalten Sie dort im Menü *Ansicht/Versteckte anzeigen* ein. Mit einem Rechtsklick in das Dateimanager-Fenster legen Sie über *Neu.../Leere Datei* eine leere Datei an. Diese können Sie wiederum mit einem Rechtsklick mit dem *Text Editor* öffnen und die abgebildeten Textzeilen eintragen.

11 LAUFSCHRIFT AUF DEM LC-DISPLAY

LC-Displays und ähnliche zeichenorientierte Anzeigeelemente werden gerne für Laufschriften verwendet. Auf diese Weise ist es möglich, auch längere Texte auf den relativ kurzen Zeilen der Anzeige darzustellen.

Dieses Experiment stellt einen beliebigen Text als Laufschrift in der unteren Zeile des Displays dar, in der oberen Zeile läuft die aus dem letzten Experiment bekannte Uhr.



Uhr und Laufschrift auf dem LC-Display

Programmdatei:
laufschrift.py

```
001 #!/usr/bin/python
002 import RPi.GPIO as GPIO
003 import time
004
005 LCD_RS = 7
006 LCD_E  = 8
```

```
LCD_D4 = 25
LCD_D5 = 24
LCD_D6 = 23
LCD_D7 = 18

GPIO.setmode(GPIO.BCM)
GPIO.setup(LCD_E,  GPIO.OUT)
GPIO.setup(LCD_RS, GPIO.OUT)
GPIO.setup(LCD_D4, GPIO.OUT)
GPIO.setup(LCD_D5, GPIO.OUT)
GPIO.setup(LCD_D6, GPIO.OUT)
GPIO.setup(LCD_D7, GPIO.OUT)

LCD_WIDTH = 16
LCD_LINE_1 = 0x80
LCD_LINE_2 = 0xC0
LCD_CHR = True
LCD_CMD = False
E_PULSE = 0.00005
E_DELAY = 0.00005
INIT  = 0.01
pause = 0.3

def lcd_enable():
  time.sleep(E_DELAY)
  GPIO.output(LCD_E, True)
  time.sleep(E_PULSE)
  GPIO.output(LCD_E, False)
  time.sleep(E_DELAY)

def lcd_byte(bits, mode):
  GPIO.output(LCD_RS, mode)
  GPIO.output(LCD_D4, bits&0x10==0x10)
  GPIO.output(LCD_D5, bits&0x20==0x20)
  GPIO.output(LCD_D6, bits&0x40==0x40)
  GPIO.output(LCD_D7, bits&0x80==0x80)
  lcd_enable()
  GPIO.output(LCD_D4, bits&0x01==0x01)
  GPIO.output(LCD_D5, bits&0x02==0x02)
  GPIO.output(LCD_D6, bits&0x04==0x04)
  GPIO.output(LCD_D7, bits&0x08==0x08)
  lcd_enable()

```

11

```
050 def lcd_string(message):
051   message = message.ljust(LCD_WIDTH," ")
052   for i in range(LCD_WIDTH):
053     lcd_byte(ord(message[i]),LCD_CHR)
054
055 LCD_INIT = [0x33, 0x32, 0x28, 0x0C, 0x06, 0x01]
056 for i in LCD_INIT:
057   lcd_byte(i,LCD_CMD)
058   time.sleep(INIT)
059
060 tx = " " * LCD_WIDTH + raw_input("Bitte Text eingeben: ")
061
062 try:
063   while True:
064     for j in range(len(tx)):
065       lcd_byte(LCD_LINE_1, LCD_CMD)
066       lcd_string(time.asctime())
067       lcd_byte(LCD_LINE_2, LCD_CMD)
068       lcd_string(tx[j:(j + LCD_WIDTH)])
069       time.sleep(pause)
070 except KeyboardInterrupt:
071   GPIO.cleanup()
```

11.1 | So funktioniert das Programm

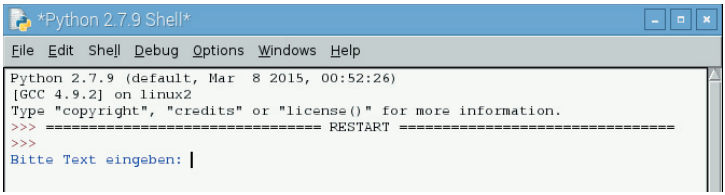
Das Programm `laufschrift.py` basiert auf dem Programm `status.py`. Für die Laufschrift braucht nur wenig verändert zu werden.

```
028 pause = 0.3
```

Die Pause wird auf 0,3 Sekunden verkürzt, sie gibt jetzt das Intervall an, in dem die Laufschrift um einen Buchstaben vorrückt. Verlängert man die Pause, läuft die Laufschrift langsamer.

```
060 tx = " " * LCD_WIDTH + raw_input("Bitte Text eingeben: ")
```

Diese Zeile fragt über die Funktion `raw_input()` nach einer Benutzereingabe.



Die Python-Shell wartet auf eine Benutzereingabe.

Der eingegebene Text wird in der Variablen `tx` gespeichert und davor werden so viele Leerzeichen gesetzt, wie das Display breit ist – in unserem Fall 16. In Python kann man einfach eine Zeichenkette mit einer Zahl multiplizieren und sie damit mehrfach wiederholen. Durch Verwendung der Konstante `LCD_WIDTH` anstelle einer Zahl ist das Programm auch auf breiteren Displays nutzbar.

```
062 try:
063   while True:
064     for j in range(len(tx)):
065       lcd_byte(LCD_LINE_1, LCD_CMD)
066       lcd_string(time.asctime())
067       lcd_byte(LCD_LINE_2, LCD_CMD)
068       lcd_string(tx[j:(j + LCD_WIDTH)])
```

Innerhalb der Hauptschleife des Programms läuft jetzt eine weitere Schleife über die Länge des Textes. In jedem Durchlauf wird in der oberen Zeile des Displays die aktuelle Uhrzeit angezeigt, in der unteren Zeile ein 16 Zeichen langer Ausschnitt von `tx`. Dieser beginnt jedes Mal ein Zeichen weiter hinten. Da `tx` am Anfang 16 Leerzeichen enthält, beginnt die Laufschrift mit einem leeren Display, der Text wandert von rechts Zeichen für Zeichen ins Bild. Am Ende des Textes sind keine zusätzlichen Leerzeichen erforderlich, diese werden von der bereits bekannten Funktion `lcd_string()` bei Bedarf automatisch am Ende angehängt.

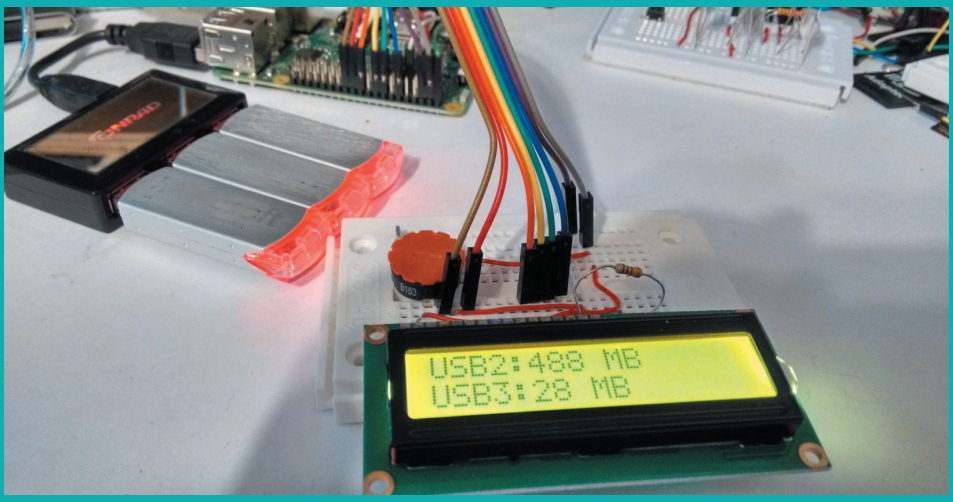
```
069       time.sleep(pause)
```

Nachdem der Textausschnitt dargestellt wurde, wartet das Programm 0,3 Sekunden. Danach erscheint der nächste, um ein Zeichen verschobene Textausschnitt.

Die innere Schleife endet, nachdem das letzte Zeichen von `tx` am linken Rand des Displays verschwunden ist und startet danach neu. Die äußere Endlosschleife läuft, bis der Benutzer mit Strg + C das Programm abbricht.

12 ERWEITERTE STATUSANZEIGE

Das nächste Programm erweitert die Statusanzeige um Anzeigen des freien Speicherplatzes auf der SD-Karte und auf bis zu drei angeschlossenen USB-Sticks. USB-Sticks benötigen so wenig Strom, dass man auch mehrere davon über einen Hub ohne eigene Stromversorgung am Raspberry Pi anschließen kann.



Drei USB-Sticks mit einem USB-Hub am Raspberry Pi angeschlossen

Die Schaltung entspricht der aus den vorhergehenden Projekten.

Dateisysteme auf USB-Sticks

USB-Sticks sind üblicherweise im vfat-Dateisystem formatiert. Linux kann dieses Dateisystem lesen und schreiben, nur die Rechteverwaltung gibt es unter vfat nicht. Lassen Sie dieses Dateisystem auf dem USB-Stick bestehen, können Sie leicht Daten von beliebigen PCs auf den Raspberry Pi übertragen, indem Sie nur den USB-Stick umstecken. Windows kann dagegen mit einem Linux-formatierten USB-Stick nichts anfangen.

Um einen USB-Stick in Linux zu mounten (= anzumelden), stellen Sie zuerst fest, unter welchem Namen der angeschlossene USB-Stick in der Linux-Verzeichnisstruktur automatisch eingetragen wurde.

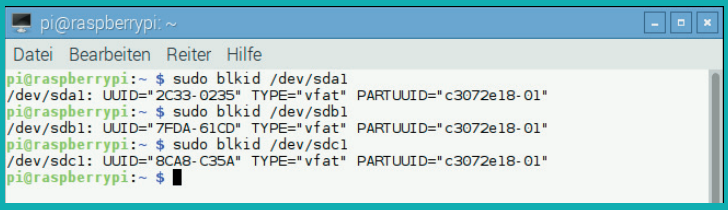
```
sudo fdisk -l
```

In den meisten Fällen wird ein USB-Stick wie eine USB-Festplatte physikalisch als `/dev/sda` bezeichnet, die erste darauf befindliche Datenpartition als `/dev/sda1`. Weitere USB-Sticks heißen dann meistens `/dev/sdb1` und `/dev/sdc1`.

Jeder USB-Stick hat eine eindeutige Geräte-ID. Um Verwirrungen zu vermeiden, wenn ein anderer USB-Stick angeschlossen wird, stellen Sie sicher, dass nur ein bestimmter USB-Stick im angegebenen Verzeichnis gemountet wird. Lesen Sie dazu zunächst die UUID des angeschlossenen USB-Sticks aus.

```
sudo blkid /dev/sda1
```

Hat der USB-Stick einen anderen physikalischen Namen als `/dev/sda1`, geben Sie diesen stattdessen an. Der Befehl zeigt die UUID des USB-Sticks sowie den Dateisystemtyp an.



UUIDs der angeschlossenen USB-Sticks anzeigen

Ermitteln Sie auf die gleiche Weise die UUIDs aller angeschlossenen USB-Sticks und tragen Sie diese dann in die Datei `/etc/fstab` ein.

```
sudo leafpad /etc/fstab
```

Ersetzen Sie im abgebildeten Beispiel die UUIDs durch die Ihrer USB-Sticks. Beachten Sie auch die anderen angezeigten Parameter. Diese sind nötig, damit der Benutzer `pi` den USB-Stick mounten kann und auch vollen Zugriff darauf hat.

12

Die Datei /etc/fstab

```
fstab
Datei  Bearbeiten  Suchen  Optionen  Hilfe
proc            /proc           proc    defaults          0       0
/dev/mmcblk0p1  /boot           vfat    defaults          0       2
/dev/mmcblk0p2  /               ext4    defaults,noatime  0       1
UUID=2C33-0235  /media/usb1     vfat    defaults,umask=000,users  0
UUID=7FDA-61CD  /media/usb2     vfat    defaults,umask=000,users  0
UUID=8CA8-C35A  /media/usb3     vfat    defaults,umask=000,users  0

# a swapfile is not a swap partition, no line here
#   use  dphys-swapfile swap[on|off]  for that
```

Starten Sie jetzt den Raspberry Pi neu. Die USB-Sticks werden automatisch gemountet und Sie können als Benutzer `pi` Daten darauf kopieren. Stecken Sie einen anderen USB-Stick an den Raspberry Pi, wird dieser nicht automatisch an einem der vorgegebenen Mountpunkte gemountet, sondern unter seiner eigenen UUID.

Programmdatei:
status_sdusb.py

12.1 | So funktioniert das Programm

Das Programm `status_sdusb.py` basiert auf dem Programm `status.py`, zeigt aber zusätzlich nach der Uhrzeit und der IP-Adresse im Wechsel von zwei Sekunden noch den freien Speicherplatz auf der SD-Karte und drei angeschlossenen USB-Sticks an. Dazu enthält es einige zusätzliche Programmzeilen:

```
002 import time, subprocess, os
```

Am Anfang wird darüber hinaus das Modul `os` importiert, das unter anderem Zugriff auf das Linux-Dateisystem bietet, um den freien Speicherplatz ermitteln zu können.

```
028 pause = 2
```

Die Pause bis zur Aktualisierung der Anzeige wird auf zwei Sekunden gesetzt. In dieser Zeit kann man die Werte auf der Anzeige bequem lesen und wartet nicht zu lange auf die nächste Anzeigeseite. Die wichtigsten Änderungen finden sich in der Hauptschleife des Programms, die jetzt aus drei Blöcken besteht, die nacheinander unterschiedliche Informationen anzeigen.

```
060 try:
061   while True:
062     sd1 = os.statvfs('/')
063     hd1 = os.statvfs('/media/usb1')
064     hd2 = os.statvfs('/media/usb2')
```

```
065     sd1x = sd1.f_bsize * sd1.f_bavail / 1048576
066     hd1x = hd1.f_bsize * hd1.f_bavail / 1048576
067     hd2x = hd2.f_bsize * hd2.f_bavail / 1048576
068     hd3x = hd3.f_bsize * hd3.f_bavail / 1048576
069     lcd_byte(LCD_LINE_1, LCD_CMD)
070     lcd_string(time.asctime())
071     lcd_byte(LCD_LINE_2, LCD_CMD)
072     lcd_string("IP:" + subprocess.check_output(["hostname","-I"])[:-2])
073     time.sleep(pause)
074     lcd_byte(LCD_LINE_1, LCD_CMD)
075     lcd_string("SD  :" + str(sd1x) + " MB")
076     lcd_byte(LCD_LINE_2, LCD_CMD)
077     lcd_string("USB1:" + str(hd1x) + " MB")
078     time.sleep(pause)
079     lcd_byte(LCD_LINE_1, LCD_CMD)
080     lcd_string("USB2:" + str(hd2x) + " MB")
081     lcd_byte(LCD_LINE_2, LCD_CMD)
082     lcd_string("USB3:" + str(hd3x) + " MB")
083     time.sleep(pause)
```

Am Anfang jedes Schleifendurchlaufs wird der freie Speicherplatz auf der SD-Karte und den angeschlossenen USB-Sticks ermittelt.

```
062     sd1 = os.statvfs('/')
063     hd1 = os.statvfs('/media/usb1')
064     hd2 = os.statvfs('/media/usb2')
065     hd3 = os.statvfs('/media/usb3')
```

Das Statistikmodul `os.statvfs()` aus der `os`-Bibliothek liefert diverse statistische Informationen zum Dateisystem, die hier für das Hauptverzeichnis bei jedem Schleifendurchlauf aufs Neue als Objekt in die Variable `sd1` geschrieben werden. Die Speicheranzeige für das Hauptverzeichnis ermittelt nicht den Speicherplatz auf gemounteten externen Datenträgern.

Auf die gleiche Weise werden die Daten der USB-Sticks ermittelt und in die Variablen `hd1`, `hd2` und `hd3` geschrieben.

Die Methode `sd1.f_bsize` liefert die Größe eines Speicherblocks in Byte aus den in der Variablen `s` gespeicherten Daten des Dateisystems. `sd1.f_bavail` gibt die Anzahl freier Blöcke an. Das Produkt aus beiden Werten gibt demnach die Anzahl freier Bytes an, die hier durch 1.048.576 geteilt wird, um die Anzahl freier Megabytes zu erhalten. Dieser Wert wird in der neuen Variablen `sd1x` gespeichert. Auf die gleiche Weise wird der

12

freie Speicherplatz auf den USB-Sticks errechnet und in die Variablen `hd1x`, `hd2x` und `hd3x` geschrieben.

Danach laufen drei Anzeigeblöcke ab, die jeweils zwei Werte in den beiden Zeilen des LC-Displays darstellen.

```
070     lcd_byte(LCD_LINE_1, LCD_CMD)
071     lcd_string(time.asctime())
072     lcd_byte(LCD_LINE_2, LCD_CMD)
073     lcd_string("IP:" + subprocess.check_output(["hostname","-I"])[:-2])
074     time.sleep(pause)
```

Im ersten Block zeigt das Display wie im Programm `status.py` Datum, Uhrzeit und IP-Adresse. Danach wartet das Programm die am Anfang festgelegte Pause von zwei Sekunden.

Der zweite Block zeigt den freien Speicherplatz auf der SD-Karte und dem ersten angeschlossenen USB-Stick. Die Textzeilen werden aus den drei Zeichenketten `"SD  :"` bzw. `"USB1:"`, dem errechneten Wert und `" MB"` zusammengesetzt.

```
075     lcd_byte(LCD_LINE_1, LCD_CMD)
076     lcd_string("SD  :" + str(sd1x) + " MB")
077     lcd_byte(LCD_LINE_2, LCD_CMD)
078     lcd_string("USB1:" + str(hd1x) + " MB")
079     time.sleep(pause)
```

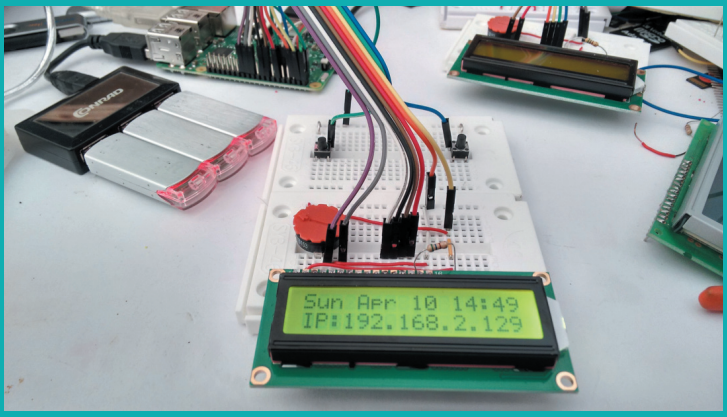
Diese Anzeige bleibt wieder zwei Sekunden lang stehen, danach werden die Daten des zweiten und dritten USB-Sticks angezeigt. Haben Sie weniger als drei USB-Sticks angeschlossen, entfernen Sie einfach die jeweiligen Programmzeilen.

```
080     lcd_byte(LCD_LINE_1, LCD_CMD)
081     lcd_string("USB2:" + str(hd2x) + " MB")
082     lcd_byte(LCD_LINE_2, LCD_CMD)
083     lcd_string("USB3:" + str(hd3x) + " MB")
084     time.sleep(pause)
```

Lassen Sie das Programm laufen und übertragen Sie große Dateien auf die Speicherkarte oder die USB-Sticks und Sie werden sehen, wie sich die Megabytezahlen bei jedem Anzeigezyklus ändern.

13 INTERAKTIVE STATUSANZEIGE MIT TASTEN

Die Statusanzeige blättert bisher automatisch zwischen den drei Seiten mit je zwei Anzeigezeilen. Zwei zusätzliche Taster machen die Anzeige interaktiv steuerbar. Mit der einen Taste blättert man nach oben, mit der anderen nach unten. So sind jeweils zwei Zeilen sichtbar, die auch automatisch aktualisiert werden.

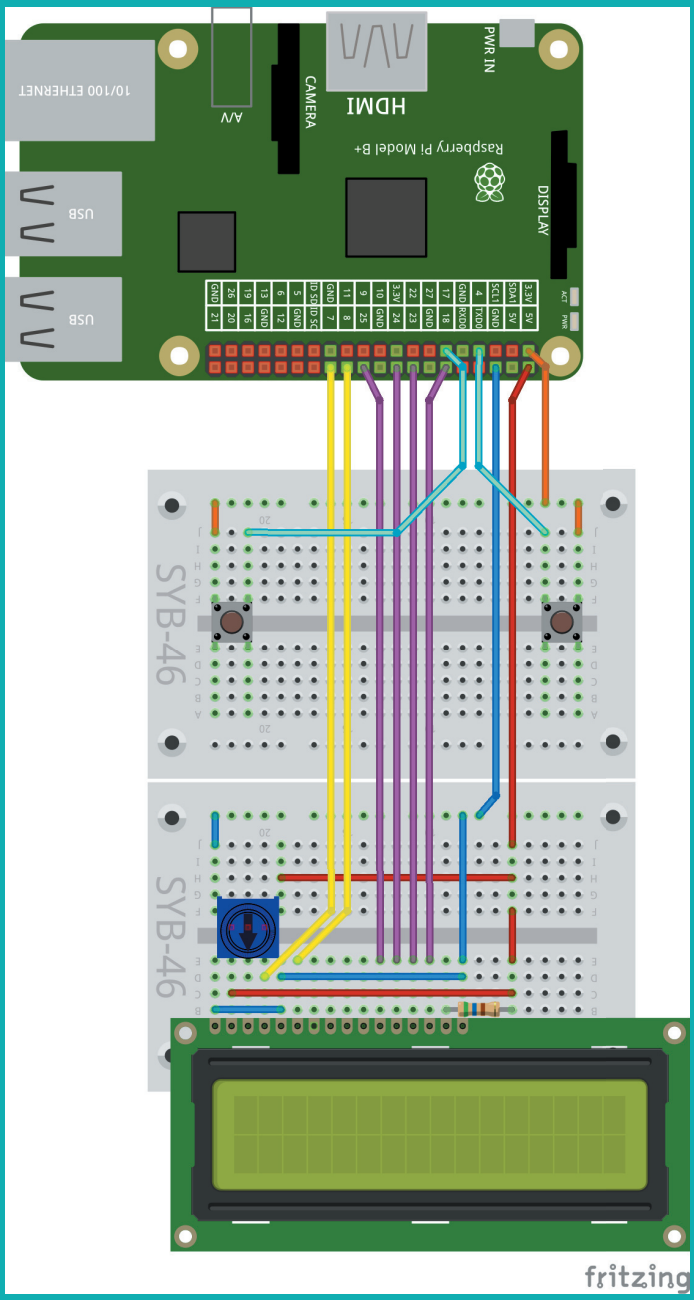


Interaktive Statusanzeige mit zwei Tastern zum Blättern

Bauen Sie auf einer zweiten Steckplatine zwei Taster mit Pulldown-Widerständen wie abgebildet auf. Die beiden Taster sind so auf dem Steckbrett angeordnet, dass sie trotz des breiten Kabelstrangs der Verbindungskabel zum Raspberry Pi gut erreichbar sind.

Benötigte Bauteile

2x Steckplatine, 1x LC-Display, 1x 560-Ohm-Widerstand (Grün-Blau-Braun), 1x Potenziometer, 2x Taster, 11x Verbindungskabel, 9x Drahtbrücke (unterschiedliche Längen)



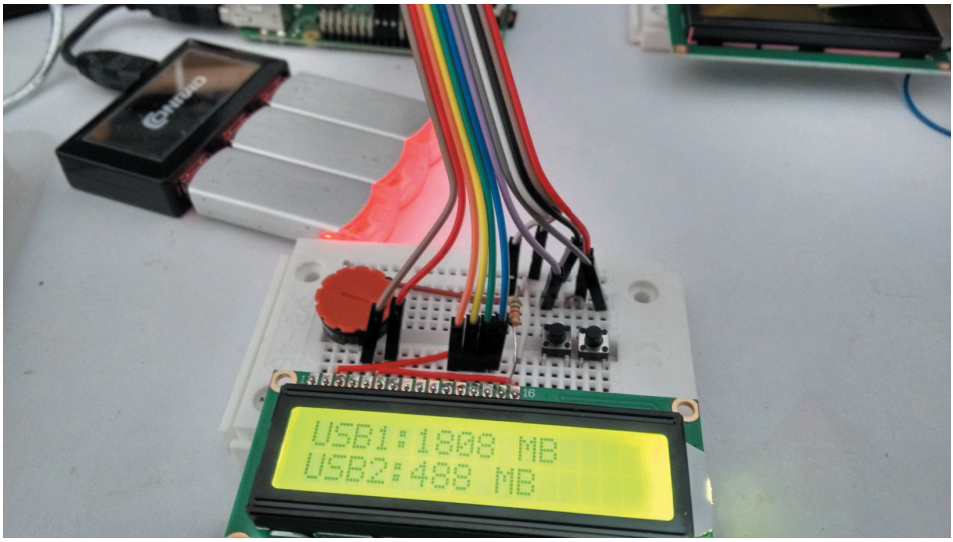
LC-Display mit zwei Tastern auf zwei Steckplatinen

13

Der Schaltungsaufbau entspricht weitgehend den vorhergehenden Experimenten mit dem LC-Display. Es werden auch die gleichen GPIO-Ports verwendet. Die beiden cyanfarben dargestellten Leitungen verbinden die Taster mit den GPIO-Ports 4 und 17, die als Eingänge verwendet werden.

Zwei verschiedene Stromversorgungen

Achten Sie beim Aufbau besonders genau auf den korrekten Anschluss der Stromversorgung, da die Schaltung beide Stromversorgungsleitungen des Raspberry Pi nutzt, +3,3 V für die GPIO-Ports der Taster und +5 V für das LC-Display. Ein +5-V-Signal darf keinesfalls zurück auf einen GPIO-Eingang gelangen.

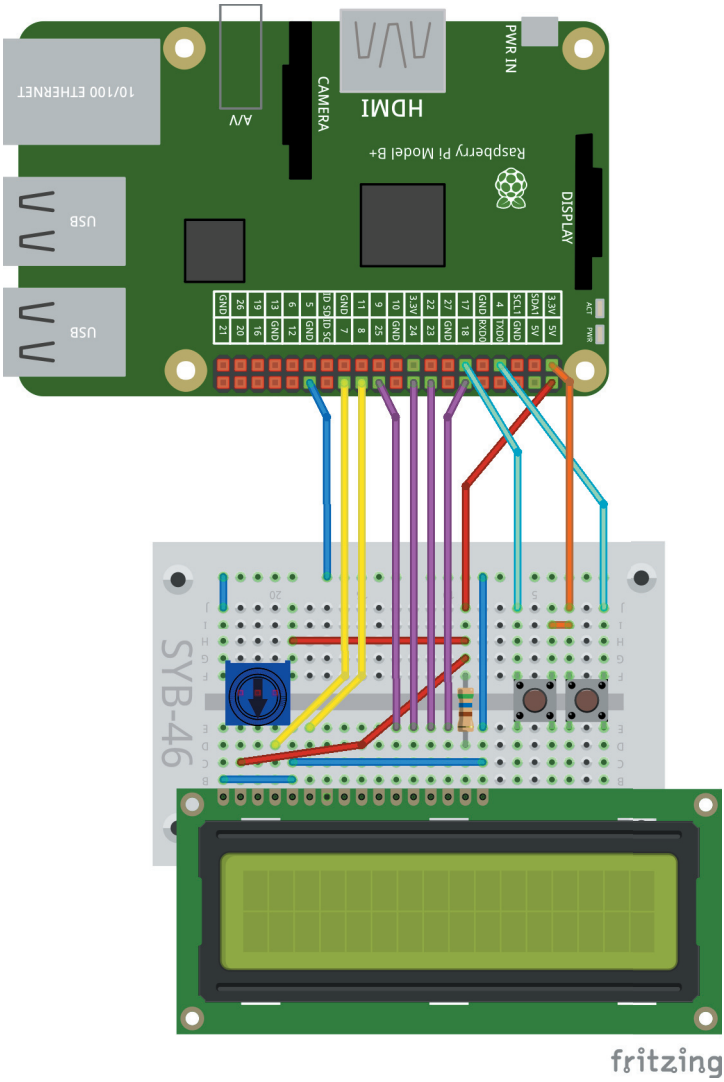


Interaktive Statusanzeige mit zwei Tastern auf einer Steckplatine

Wer es etwas handlicher haben möchte, kann die beiden Taster auch noch auf derselben Steckplatine wie die übrigen Bauteile unterbringen, wenn man alles etwas enger aufbaut.

Benötigte Bauteile:

1x Steckplatine, 1x LCD Display, 1x 560-Ohm-Widerstand (Grün-Blau-Braun), 1x Potenziometer, 2x Taster, 11x Verbindungskabel, 7x Drahtbrücke (unterschiedliche Längen)



LC-Display mit zwei Tastern auf einer Steckplatine

13.1 | So funktioniert das Programm

Programm-datei:
status_sdusb_taste.py

Das Programm `status_sdusb_taste.py` basiert auf dem Programm `status_sdusb.py` und ist um die Steuerung durch die beiden Taster erweitert.

13

```
011 T1 = 4
012 T2 = 17
```

Dazu werden am Anfang im Definitionsbereich zwei weitere GPIO-Ports für die beiden Taster definiert ...

```
021 GPIO.setup(T1, GPIO.IN, pull_up_down = GPIO.PUD_DOWN)
022 GPIO.setup(T2, GPIO.IN, pull_up_down = GPIO.PUD_DOWN)
```

... und später als Eingänge mit internen Pulldown-Widerständen initialisiert.

```
032 pause = 0.1
```

Die Pause wird auf 0,1 Sekunden verkürzt, da sie diesmal das Abfrageintervall der Taster festlegt. Bei zu langen Pausen müsste der Benutzer sehr lange auf die Taster drücken, damit der Tastendruck vom Programm registriert wird.

```
064 n = 0
065 z = ["", "", "", "", "", ""]
```

Jetzt werden noch eine Variable `n` und ein Array `z` mit sechs leeren Zeichenketten definiert, die später für die Auswahl und Darstellung der insgesamt sechs verschiedenen Anzeigezeilen benötigt werden.

In der Hauptschleife sind die wichtigsten Unterschiede gegenüber früheren Programmversionen zu finden.

```
067 try:
068   while True:
069     sd1 = os.statvfs('/')
070     hd1 = os.statvfs('/media/usb1')
071     hd2 = os.statvfs('/media/usb2')
072     hd3 = os.statvfs('/media/usb3')
073     sd1x = sd1.f_bsize * sd1.f_bavail / 1048576
074     hd1x = hd1.f_bsize * hd1.f_bavail / 1048576
075     hd2x = hd2.f_bsize * hd2.f_bavail / 1048576
076     hd3x = hd3.f_bsize * hd3.f_bavail / 1048576
077     z[0] = time.asctime()
078     z[1] = "IP:" + subprocess.check_output(["hostname","-I"])[:-2]
079     z[2] = "SD  :" + str(sd1x) + " MB"
080     z[3] = "USB1:" + str(hd1x) + " MB"
081     z[4] = "USB2:" + str(hd2x) + " MB"
082     z[5] = "USB3:" + str(hd3x) + " MB"
```

Die Texte für die sechs verschiedenen Anzeigezeilen werden aus den entsprechenden Werten zusammengesetzt und in den sechs Feldern des Arrays `z[]` abgelegt. Die Variable `n` gibt an, welche Zeilen dargestellt werden sollen. Am Anfang steht diese Variable auf `0`. Das bedeutet, `z[0]` und `z[1]` erscheinen in der Anzeige.

```
083     if GPIO.input(T1)==1:
084       n += 1
085       if n > 4:
086         n = 4
```

Drückt der Benutzer die Taste 1, wird `n` um 1 erhöht, die Anzeige blättert einen Schritt nach unten und zeigt jetzt `z[1]` und `z[2]`. Erreicht `n` durch mehrfaches Drücken der Taste den Wert 4, werden `z[4]` und `z[5]` angezeigt. Höhere Werte sind nicht möglich, da keine weiteren Textzeilen definiert sind. Deshalb wird `n` automatisch wieder auf 4 gesetzt, sollte der Benutzer ein weiteres Mal die Taste 1 drücken.

```
087     if GPIO.input(T2)==1:
088       n -= 1
089       if n < 0:
090         n = 0
```

Nach dem gleichen Prinzip blättert die andere Taste nach oben, indem der Wert `n` bei jedem Tastendruck um 1 reduziert wird. Würde der Wert kleiner als 0, wird er automatisch auf 0 gesetzt. In diesem Fall werden `z[0]` und `z[1]` angezeigt.

```
091     lcd_byte(LCD_LINE_1, LCD_CMD)
092     lcd_string(z[n])
093     lcd_byte(LCD_LINE_2, LCD_CMD)
094     lcd_string(z[n + 1])
095     time.sleep(pause)
```

Am Ende jedes Schleifendurchlaufs werden die in `n` definierte Zeile und die folgende auf dem LC-Display angezeigt. Nach einer kurzen Wartezeit startet die Schleife neu, aktualisiert die Anzeige und prüft, ob der Benutzer eine Taste gedrückt hat.

14 LC-DISPLAY IM 8-BIT-MODUS

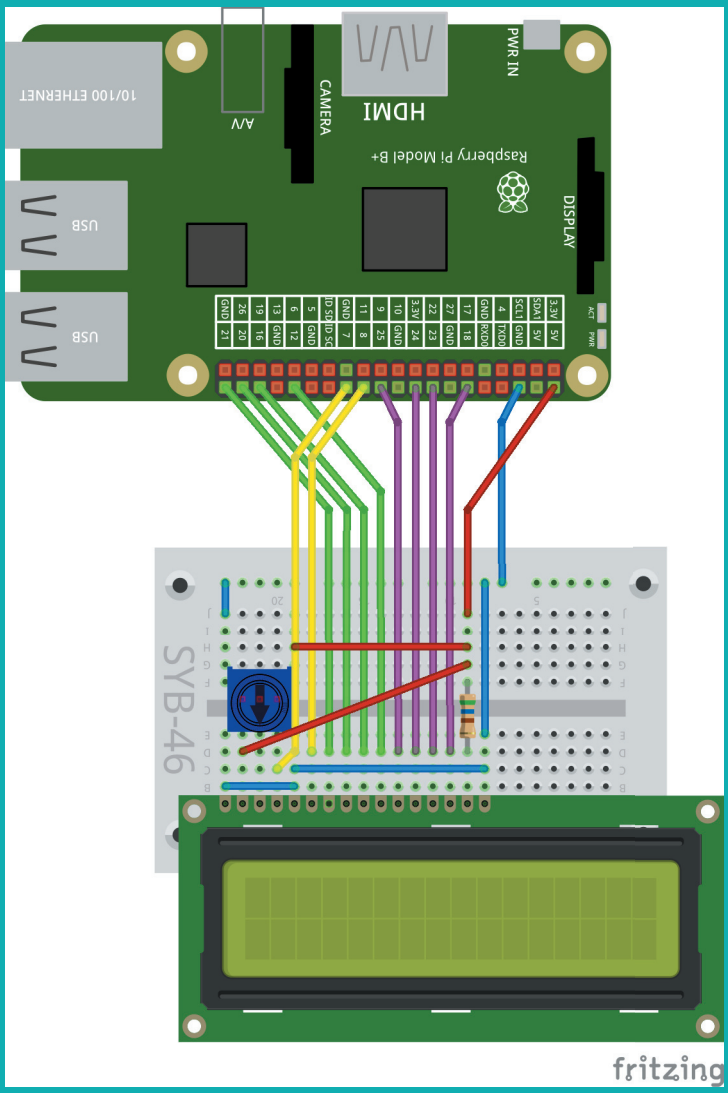
Das nächste Projekt zeigt, wie das LC-Display im 8-Bit-Modus verwendet wird. Verbinden Sie dazu mit vier weiteren Verbindungskabeln, wie in der Abbildung gezeigt, die Datenleitungen `D0...D3` des Displays mit den GPIO-Ports `21`, `20`, `16`, `12`.

Benötigte Bauteile

1x Steckplatine, 1x LC-Display, 1x 560-Ohm-Widerstand (Grün-Blau-Braun), 1x Potenziometer, 12x Verbindungskabel, 6x Drahtbrücke (unterschiedliche Längen)

Zur Übersicht auch hier noch einmal eine Tabelle der Datenleitungen des Displays und der verwendeten GPIO-Ports. Die Leitungen `D4...D7` sind gleich angeschlossen wie im 4-Bit-Modus in den vorhergehenden Projekten.

Displaypin	Signal	Raspberry-Pi-GPIO
4	RS	GPIO 7
6	E	GPIO 8
7	D0	GPIO 21
8	D1	GPIO 20
9	D2	GPIO 16
10	D3	GPIO 12
11	D4	GPIO 25
12	D5	GPIO 24
13	D6	GPIO 23
14	D7	GPIO 18



LC-Display im 8-Bit-Modus. Die zusätzlichen Leitungen sind grün dargestellt.

14.1 | So funktioniert das Programm

Das Programm `status08.py` basiert auf dem ersten Programm `status.py` zur Anzeige von Uhrzeit und IP-Adresse. Wir zeigen hier nur die Änderungen für den 8-Bit-Modus.

Programmdatei:
status08.py

14

```
005 LCD_RS = 7
006 LCD_E  = 8
007 LCD_D0 = 21
008 LCD_D1 = 20
009 LCD_D2 = 16
010 LCD_D3 = 12
011 LCD_D4 = 25
012 LCD_D5 = 24
013 LCD_D6 = 23
014 LCD_D7 = 18
```

Bei den Definitionen der GPIO-Portnummern werden vier weitere Variablen für die zusätzlichen vier Datenleitungen deklariert.

```
016 GPIO.setmode(GPIO.BCM)
017 GPIO.setup(LCD_E,  GPIO.OUT)
018 GPIO.setup(LCD_RS, GPIO.OUT)
019 GPIO.setup(LCD_D0, GPIO.OUT)
020 GPIO.setup(LCD_D1, GPIO.OUT)
021 GPIO.setup(LCD_D2, GPIO.OUT)
022 GPIO.setup(LCD_D3, GPIO.OUT)
023 GPIO.setup(LCD_D4, GPIO.OUT)
024 GPIO.setup(LCD_D5, GPIO.OUT)
025 GPIO.setup(LCD_D6, GPIO.OUT)
026 GPIO.setup(LCD_D7, GPIO.OUT)
```

Auch diese vier GPIO-Ports werden als Ausgänge initialisiert. Die weiteren Variablendeklarationen sowie die Funktion `lcd_enable()` bleiben unverändert.

```
045 def lcd_byte(bits, mode):
046   GPIO.output(LCD_RS, mode)
047   GPIO.output(LCD_D0, bits&0x01==0x01)
048   GPIO.output(LCD_D1, bits&0x02==0x02)
049   GPIO.output(LCD_D2, bits&0x04==0x04)
050   GPIO.output(LCD_D3, bits&0x08==0x08)
051   GPIO.output(LCD_D4, bits&0x10==0x10)
052   GPIO.output(LCD_D5, bits&0x20==0x20)
053   GPIO.output(LCD_D6, bits&0x40==0x40)
054   GPIO.output(LCD_D7, bits&0x80==0x80)
055   lcd_enable()
```

Die Funktion `lcd_byte()` errechnet weiterhin die 8 Bits jedes Zeichens, gibt diese jetzt aber nicht mehr in zwei Blöcken, sondern auf einmal auf acht Datenleitungen aus.

Die Funktion `lcd_string()` ist wieder unverändert, da diese sich nur mit der Umsetzung von Zeichenketten und nicht mit der direkten Ansteuerung der Display-Hardware befasst.

```
062 LCD_INIT = [0x33, 0x32, 0x38, 0x0C, 0x06, 0x01]
```

Um das Display im 8-Bit-Modus zu verwenden, muss sich der Initialisierungsstring im dritten Zeichen unterscheiden:

Drittes Zeichen des Initialisierungsstrings	
0x28	4-Bit-Modus
0x38	8-Bit-Modus

Die restlichen Programmanweisungen können wieder unverändert aus der Programmversion für den 4-Bit-Modus übernommen werden.

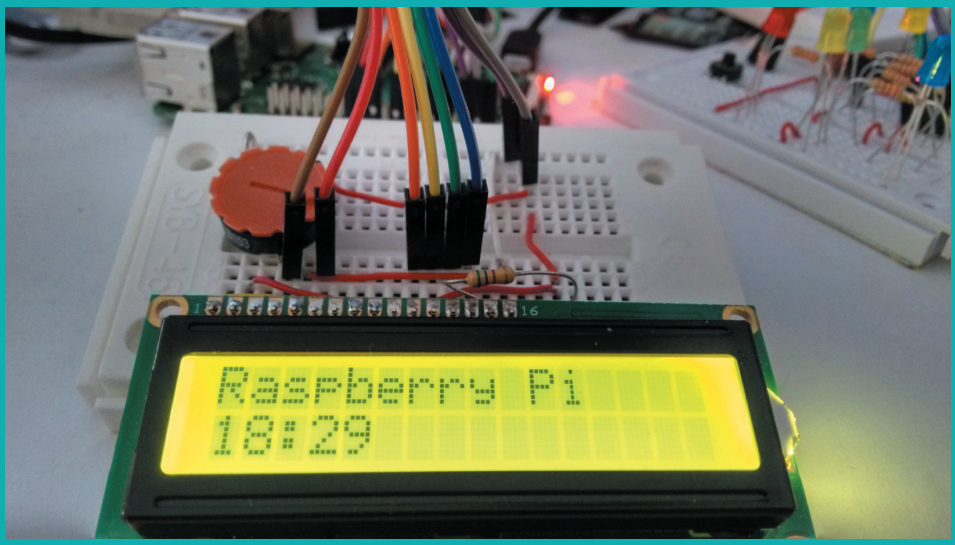
4 Bit oder 8 Bit – welcher Modus ist besser?

Vergleicht man den Verkabelungs- und Programmieraufwand für den 4-Bit-Modus und den 8-Bit-Modus, stellt man schnell fest, dass der 4-Bit-Modus die bessere Wahl ist. Im 4-Bit-Modus werden nur sechs statt zehn GPIO-Ports benötigt und das Programm ist sogar um ein paar Zeilen kürzer – zumindest bei Verwendung von Python. Aus diesen Gründen verwenden wir für die meisten Beispiele auch den 4-Bit-Modus.

Natürlich werden Sie sich fragen, warum es überhaupt noch einen 8-Bit-Modus gibt, wenn der 4-Bit-Modus doch offenbar nur Vorteile hat. Die Hersteller von LC-Displays könnten sich vier Anschlüsse sparen. Im Experiment „LCD-Display am Portexpander", sehen Sie, dass der 8-Bit-Modus in bestimmten Anwendungsfällen auch seine Stärken hat.

15 DIGITALUHR MIT SCRATCH AUF DEM LC-DISPLAY

LC-Displays werden zurzeit von Scratch noch nicht direkt unterstützt. Mithilfe eines Open-Source-Programmmoduls lassen sich Scratch und Python miteinander verbinden. Werden in Scratch Variablen gesetzt oder verändert, bekommt Python eine Nachricht in Form eines *dict*-Objekts, das dann vom Python-Programm ausgewertet werden kann – z. B. um Hardware zu steuern, die Scratch nicht direkt unterstützt.



Digitaluhr mit Scratch auf dem LC-Display

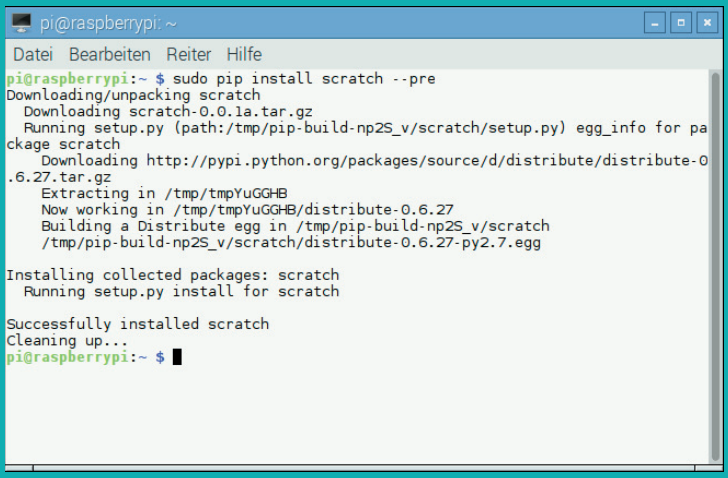
Der Schaltungsaufbau entspricht den vorherigen Projekten im 4-Bit-Modus. Das Scratch-Programm stellt die abgebildete Digitaluhr auf dem LC-Display dar.

Zuerst muss das Zusatzmodul *scratch* für Python installiert werden.

Öffnen Sie ein Terminalfenster. Geben Sie hier Folgendes ein:

```
sudo pip install scratch --pre
```

Beginnen Sie mit den nächsten Schritten erst, wenn die Installation erfolgreich durchgelaufen ist.

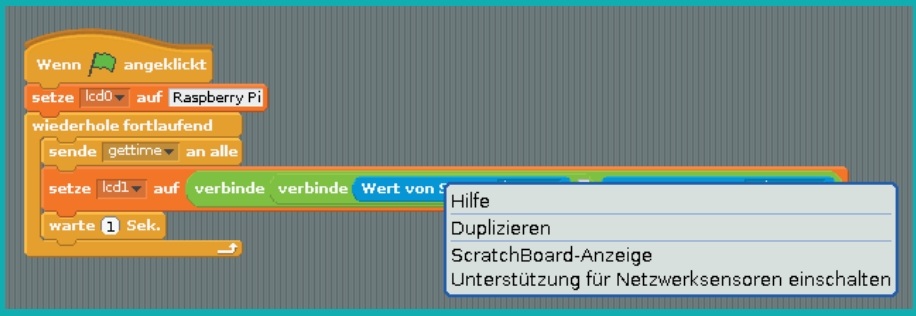


scratch-Modul für Python nachinstallieren

Öffnen Sie jetzt in Scratch das Programm `uhr`, starten Sie es aber noch nicht. Klicken Sie mit der rechten Maustaste auf einen der beiden Blöcke *Wert von Sensor* im Programm und wählen Sie dort *Unterstützung für Netzwerksensoren einschalten*. Danach erscheint eine Meldung *Unterstützung für Netzwerksensoren ist eingeschaltet*. Bestätigen Sie diese mit *OK*.

Programmdatei:

uhr.sb



Unterstützung für Netzwerksensoren einschalten

15

Starten Sie in einem Kommandozeilenfenster das Python-Programm zur Steuerung des LCD-Moduls mit Scratch:

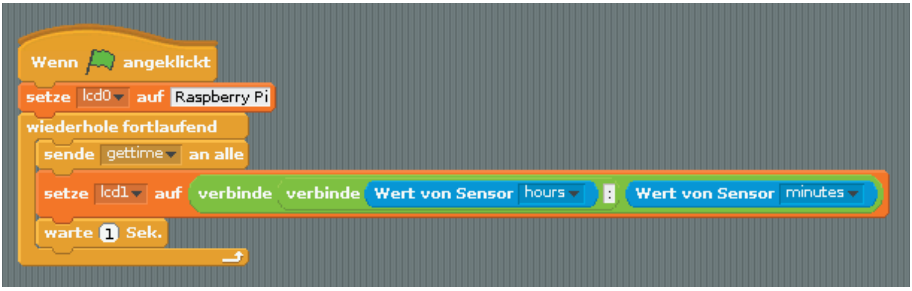
```
python display.py
```

Eventuelle GPIO-Warnungen können Sie dabei ignorieren. Alternativ können Sie das Programm auch über die Python-Entwicklungsumgebung öffnen und ausführen.

Klicken Sie jetzt auf das grüne Fähnchen in Scratch, wird die aktuelle Uhrzeit auf dem LCD-Modul angezeigt.

15.1 | So funktioniert das Scratch-Programm

Das Scratch-Programm schreibt im Sekundenrhythmus die aktuelle Zeit in eine Variable, die vom Python-Programm ausgewertet wird.



Der statische Text, der in der oberen Zeile des LC-Displays angezeigt werden soll, wird in der Variablen `lcd0` gespeichert. Danach startet eine Endlosschleife, die zunächst die Nachricht `gettime` sendet, die die aktuelle Systemzeit abfragt.

Anschließend wird der Text für die untere Zeile des Displays zusammengebaut und in der Variablen `lcd1` gespeichert. Dieser besteht aus drei Komponenten, die mit *verbinde*-Blöcken zu einer Zeichenkette verbunden werden.

Die aktuelle Stunde und Minute der Systemzeit wird über zwei Sensoren ausgelesen. Scratch aktiviert diese Sensoren automatisch, nachdem `gettime` gesendet wurde.

Nach einer Sekunde startet der nächste Schleifendurchlauf, der wieder die aktuelle Zeit ermittelt.

15.2 | So funktioniert das Python-Programm

Das Python-Programm `display.py` läuft im Hintergrund und wertet die beiden Scratch-Variablen `lcd0` und `lcd1` aus.

Programmdatei:
display.py

```
001 #!/usr/bin/python
002
003 import RPi.GPIO as GPIO
004 import time, scratch
005 s = scratch.Scratch()
006
007 LCD_RS = 7
008 LCD_E  = 8
009 LCD_D4 = 25
010 LCD_D5 = 24
011 LCD_D6 = 23
012 LCD_D7 = 18
013
014 GPIO.setmode(GPIO.BCM)
015 GPIO.setup(LCD_E,  GPIO.OUT)
016 GPIO.setup(LCD_RS, GPIO.OUT)
017 GPIO.setup(LCD_D4, GPIO.OUT)
018 GPIO.setup(LCD_D5, GPIO.OUT)
019 GPIO.setup(LCD_D6, GPIO.OUT)
020 GPIO.setup(LCD_D7, GPIO.OUT)
021
022 LCD_WIDTH = 16
023 LCD_LINE_1 = 0x80
024 LCD_LINE_2 = 0xC0
025 LCD_CHR = True
026 LCD_CMD = False
027 E_PULSE = 0.00005
028 E_DELAY = 0.00005
029 INIT  = 0.01
030 pause = 0.01
031
032 def lcd_enable():
033   time.sleep(E_DELAY)
034   GPIO.output(LCD_E, True)
035   time.sleep(E_PULSE)
036   GPIO.output(LCD_E, False)
037   time.sleep(E_DELAY)
038
```

15

```
039 def lcd_byte(bits, mode):
040   GPIO.output(LCD_RS, mode)
041   GPIO.output(LCD_D4, bits&0x10==0x10)
042   GPIO.output(LCD_D5, bits&0x20==0x20)
043   GPIO.output(LCD_D6, bits&0x40==0x40)
044   GPIO.output(LCD_D7, bits&0x80==0x80)
045   lcd_enable()
046   GPIO.output(LCD_D4, bits&0x01==0x01)
047   GPIO.output(LCD_D5, bits&0x02==0x02)
048   GPIO.output(LCD_D6, bits&0x04==0x04)
049   GPIO.output(LCD_D7, bits&0x08==0x08)
050   lcd_enable()
051
052 def lcd_string(message):
053   message = message.ljust(LCD_WIDTH," ")
054   for i in range(LCD_WIDTH):
055     lcd_byte(ord(message[i]),LCD_CHR)
056
057 LCD_INIT = [0x33, 0x32, 0x28, 0x0C, 0x06, 0x01]
058 for i in LCD_INIT:
059   lcd_byte(i,LCD_CMD)
060   time.sleep(INIT)
061
062 try:
063   while True:
064     z = (s.receive())['sensor-update']
065     lcd_byte(LCD_LINE_1, LCD_CMD)
066     if z.has_key('lcd0'):
067       lcd_string(z['lcd0'])
068     lcd_byte(LCD_LINE_2, LCD_CMD)
069     if z.has_key('lcd1'):
070       lcd_string(z['lcd1'])
071     time.sleep(pause)
072 except KeyboardInterrupt:
073   GPIO.cleanup()
```

Das Programm entspricht in großen Teilen den bereits bekannten Programmen zur Steuerung des LCD-Moduls.

Am Anfang wird das Modul `scratch` zusätzlich importiert und eine Variable `s` deklariert, die die Datenstruktur zur Kommunikation mit Scratch enthält.

```
003 import RPi.GPIO as GPIO
004 import time, scratch
005 s = scratch.Scratch()
```

In einer Endlosschleife werden mithilfe der `receive()`-Methode des Scratch-Objekts Daten aus Scratch nach Python übertragen. Diese Daten werden als *dict*-Objekt geliefert, das zwei Schlüssel, `broadcast` und `sensor-update`, enthält.

```
{
'broadcast': ['gettime'],
'sensor-update': {'lcd1': '22:22', 'lcd0': 'Raspberry Pi'}
}
```

Der Schlüssel `broadcast` enthält die letzte, über *sende ... an alle* gesendete Nachricht.

Der Schlüssel `sensor-update` enthält ein weiteres *dict*-Objekt, in dem alle seit der letzten Abfrage veränderten Scratch-Variablen enthalten sind.

```
    z = (s.receive())['sensor-update']
```

Diese Programmzeile speichert in der Variablen `z` das *dict*-Objekt aus dem Schlüssel `sensor-update`. Die Methode `has_key()` überprüft, ob in einem *dict*-Objekt ein bestimmter Schlüssel vorhanden ist. Enthält `z` einen Schlüssel `lcd0`, wird dessen Inhalt auf der oberen Zeile des LCD-Moduls angezeigt.

```
065     lcd_byte(LCD_LINE_1, LCD_CMD)
066     if z.has_key('lcd0'):
067       lcd_string(z['lcd0'])
```

Auf die gleiche Weise wird in der unteren Zeile der Inhalt des Schlüssels `lcd1` angezeigt, wenn dieser vorhanden ist.

```
068     lcd_byte(LCD_LINE_2, LCD_CMD)
069     if z.has_key('lcd1'):
070       lcd_string(z['lcd1'])
071     time.sleep(pause)
```

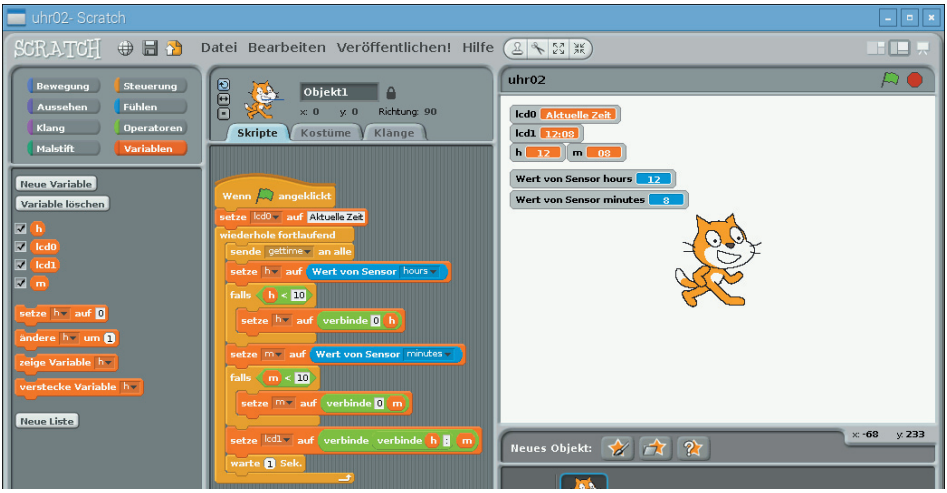
Zum Schluss wartet das Programm 0,01 Sekunden und übernimmt dann wieder die aktuellen Werte aus Scratch.

15

15.3 | Verbesserte Uhrzeitanzeige

Dieses einfache Uhrenprogramm ermittelt zwar zuverlässig die aktuelle Zeit, zeigt diese aber in den ersten zehn Minuten jeder Stunde sehr ungewöhnlich an, da die Minuten nur einstellig dargestellt werden, z. B. `12:8` statt `12:08`. Das Gleiche gilt auch für einstellige Stundenangaben, wobei dies aber nicht so negativ auffällt.

Die verbesserte Version `uhr02` verwendet zwei Variablen, in denen die aktuelle Stunde und Minute zwischengespeichert und bei Bedarf zweistellig formatiert werden. Das Programm zeigt alle Variablen und Sensorwerte auf der Bühne an.



Stunden und Minuten werden auf der Uhr immer zweistellig formatiert.

In jedem Schleifendurchlauf werden die aktuelle Stunde in der Variablen `h` und die aktuelle Minute in der Variablen `m` gespeichert. Falls einer dieser Werte kleiner als 10 ist, hängt ein *verbinde*-Block eine `0` davor. Zeichenketten und Zahlenwerte können in Scratch ohne Konvertierung gleichwertig miteinander verbunden werden.

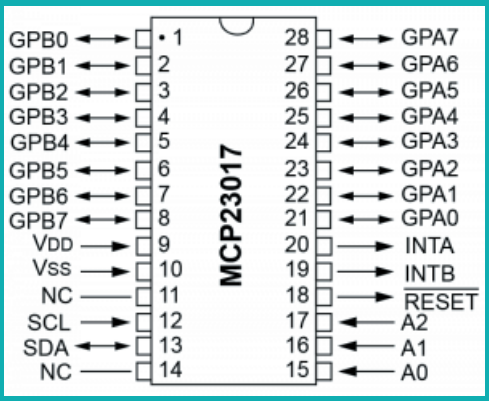
16 LAUFLICHT MIT DEM PORTEXPANDER

Lauflichter sind immer wieder beliebte Effekte, um Aufmerksamkeit zu erregen, sei es im Partykeller oder in professioneller Leuchtwerbung. Mit dem Raspberry Pi und ein paar LEDs lässt sich so etwas leicht realisieren. Das Problem ist, dass die GPIO-Ports schnell knapp werden.

16.1 | Der Portexpander MCP23017

Im Paket befindet sich ein integrierter Schaltkreis mit der Bezeichnung MCP23017. Dabei handelt es sich um einen Portexpander, der 16 programmierbare GPIO-Ports liefert, selbst aber nur über zwei Steuerleitungen vom Raspberry Pi angesteuert wird.

Der MCP23017-Chip hat auf jeder Seite 14 Anschlüsse, wovon auf jeder Seite 8 GPIO-Ports sind. Der Rest sind Steuerleitungen und Stromversorgung.



Pinbelegung des MCP23017

In der Linksammlung `www.softwarehandbuch.de/raspberry-pi` finden Sie das Datenblatt zu diesem Chip.

Kerbe beachten

Der MCP23017 hat wie alle integrierten Schaltkreise dieser Bauform an einer Schmalseite eine Kerbe. Diese dient zur Orientierung, um den Baustein in der richtigen Richtung einzubauen. Wenn Sie die Schaltungen auf den Steckplatinen entsprechend den Abbildungen in diesem Buch nachbauen, ist die Kerbe immer rechts.

Die Anschlüsse `GPA0...GPA7` und `GPB0...GPB7` sind zwei je 8 Bit breite GPIO-Schnittstellen `GPIOA` und `GPIOB`, die binärcodiert angesteuert werden können. Das bedeutet, man steuert nicht jeden Port einzeln an, sondern schreibt eine 8 Bit lange Zahl im Ganzen auf die `GPIOA`- oder `GPIOB`-Schnittstelle.

Die Anschlüsse `VDD` und `VSS` sind die Stromversorgung für den Portexpander. `VDD` muss an +3,3 V angeschlossen werden, `VSS` an 0 V.

Der `RESET`-Anschluss muss im Normalbetrieb auf `True` stehen, also mit +3,3 V verbunden sein.

Die Anschlüsse `A0`, `A1`, `A2` legen die Adresse des Portexpanders fest. Mit den drei Leitungen lassen sich binär die Zahlen von 0 bis 7 darstellen. Ist eine Leitung mit 0 V verbunden, hat sie den Wert 0, bei der Verbindung mit +3,3 V hat sie den Wert 1. Dazu wird `0x20` addiert, sodass sich die i2c-Adressen `0x20...0x27` verwenden lassen.

16.2 | Das i2c-Protokoll

i2c (auch als I²C „I-Quadrat-C" = „Inter Chip Communication" bezeichnet) ist ein standardisiertes serielles Kommunikationsprotokoll, mit dem Elektronikchips über eine Zweidrahtverbindung Daten untereinander austauschen können. Auf diesem sehr einfachen Datenbus braucht jedes Gerät eine eindeutige Adresse, damit die Daten entsprechend den Geräten zugeordnet werden können.

Der Raspberry Pi unterstützt das i2c-Protokoll, es muss aber erst aktiviert werden:

Rufen Sie über den Menüpunkt *Einstellungen* das Programm *Raspberry Pi Configuration* auf und schalten Sie dort auf der Registerkarte *Interfaces* den Schalter *I2C* auf *Enable*. Starten Sie dann den Raspberry Pi neu.

16

Installieren Sie anschließend noch das Paket zum Zugriff auf i2c über Python.

```
sudo apt-get update
sudo apt-get install python-smbus
```

Geben Sie danach folgenden Befehl ein, um die i2c-Unterstützung zu testen:

```
sudo i2cdetect -y 1
```

Wenn die i2c-Unterstützung aktiv ist, erscheint eine Tabelle der i2c-Geräte am Bus, die am Anfang leer ist, solange kein Gerät angeschlossen ist.

```
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

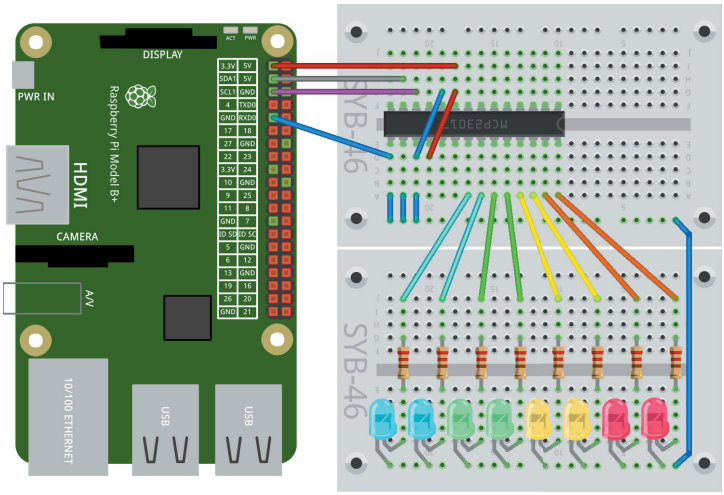
Erscheint eine Fehlermeldung, ist das i2c-Protokoll auf dem Raspberry Pi nicht korrekt aktiviert.

16.3 | LEDs am Portexpander

Das erste Experiment mit i2c und dem Portexpander lässt acht LEDs als Lauflicht nacheinander aufleuchten. Bauen Sie die Schaltung wie in der Abbildung auf.

Benötigte Bauteile

2x Steckplatine, 1x Portexpander MCP23017, 2x LED rot, 2x LED gelb, 2x LED grün, 2x LED blau, 8x 220-Ohm-Widerstand (Rot-Rot-Braun), 4x Verbindungskabel, 14x Drahtbrücke (unterschiedliche Längen)



Lauflicht mit Portexpander MCP23017

Achten Sie darauf, den MCP23017 mit der Kerbe nach rechts einzubauen. Die farbigen Drahtbrücken verbinden die farblich passenden LEDs mit den acht Anschlüssen der `GPIOA`-Schnittstelle. Alle blau dargestellten Drahtbrücken sind mit 0 V verbunden, die roten mit +3,3 V – hier außer der Stromversorgung auch der *RESET*-Pin.

Die drei kurzen blauen Drahtbrücken verbinden die drei Adressleitungen mit 0 V und geben dem Portexpander damit die i2c-Adresse `0x20`. Lassen Sie, nachdem Sie die Schaltung mit dem Raspberry Pi verbunden haben, noch einmal `i2cdetect` laufen.

Programmdatei:
i2c_01.py

```
sudo i2cdetect -y 1
```

Jetzt erscheint das neue Gerät mit der Adresse `0x20` in der i2c-Geräteliste.

```
001      0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
002 00:          -- -- -- -- -- -- -- -- -- -- -- -- --
003 10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
004 20: 20 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
005 30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
006 40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
007 50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
008 60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
009 70: -- -- -- -- -- -- -- --
```

Das Programm `i2c_01.py` lässt die LEDs als Lauflicht leuchten.

16

```
001 #!/usr/bin/python
002 import time, smbus
003
004 DEVICE = 0x20
005 IODIRA = 0x00
006 GPIOA  = 0x12
007
008 bus = smbus.SMBus(1)
009 bus.write_byte_data(DEVICE,IODIRA,0x00)
010 bus.write_byte_data(DEVICE,GPIOA,0x00)
011
012 while True:
013   j = 1
014   for i in range(8):
015     bus.write_byte_data(DEVICE,GPIOA,j)
016     j *= 2
017     time.sleep(0.1)
```

16.3.1 | So funktioniert das Programm

Programme, die ausschließlich i2c nutzen, brauchen die GPIO-Schnittstelle nicht zu initialisieren. Es ist nicht einmal die RPi.GPIO-Bibliothek nötig.

```
002 import time, smbus
```

Für i2c muss die Bibliothek `smbus` importiert werden. Die Bibliothek `time` wird für die Berechnung der Blinkdauer benötigt. Anschließend werden drei Hardware-Adressen als Konstanten definiert.

DEVICE = 0x20	i2c-Geräteadresse des Portexpanders
IODIRA = 0x00	8-Bit-Register, das die Richtung (Ausgang/Eingang) der GPIOA-Ports festlegt
GPIOA = 0x12	8-Bit-Datenregister der GPIOA-Ports

```
008 bus = smbus.SMBus(1)
```

Diese Zeile definiert ein Objekt namens `bus` vom Typ `smbus.SMBus()`. Darüber wird der i2c-Bus mit allen angeschlossenen Geräten angesprochen.

Die Methode `bus.write_byte_data()` schreibt ein Byte auf den i2c-Bus. Alle acht Bits eines der beiden GPIO-Ports müssen immer gleichzeitig

angesteuert werden. Der i2c-Bus verarbeitet immer ganze Bytes aus acht Bits. Ein Byte kann binär, dezimal oder hexadezimal angegeben werden.

```
009 bus.write_byte_data(DEVICE,IODIRA,0x00)
```

Diese Zeile schreibt auf das in der Konstanten `DEVICE` definierte Gerät (hier das einzige vorhandene i2c-Gerät) in alle Bits des Registers `IODIRA` den Wert `0`. Damit werden alle acht `GPIOA`-Ports als Ausgänge definiert. Stünde ein Bit auf `1`, würde der entsprechende Port als Eingang definiert.

```
010 bus.write_byte_data(DEVICE,GPIOA,0x00)
```

Diese Zeile schreibt auf das gleiche Gerät in alle Bits des Datenregisters `GPIOA` den Wert `0`. Damit werden alle acht `GPIOA`-Ports auf `0` gesetzt, die angeschlossenen LEDs also ausgeschaltet.

Für das Lauflicht werden jetzt nacheinander die acht LEDs einzeln eingeschaltet. Dazu muss das `GPIOA`-Datenregister nacheinander genau die Zahlenwerte annehmen, bei denen ein einzelnes Bit auf 1 steht.

LED leuchtet	Binär	Dezimal	Hex
1	0000 0001	1	0x01
2	0000 0010	2	0x02
3	0000 0100	4	0x04
4	0000 1000	8	0x08
5	0001 0000	16	0x10
6	0010 0000	32	0x20
7	0100 0000	62	0x40
8	1000 0000	128	0x80

In der Tabelle ist leicht zu erkennen, dass der Dezimalwert von 1 beginnend einfach bei jedem Schritt verdoppelt werden muss, um die passende Zahl zu erhalten.

```
012 while True:
```

Das Lauflicht wird in einer Endlosschleife wiederholt, bis der Benutzer das Programm mit Strg + C abbricht.

16

```
013   j = 1
```

Am Anfang jedes Schleifendurchlaufs wird der Wert für das Datenregister auf `1` gesetzt, damit nur die erste LED leuchtet.

```
014   for i in range(8):
```

Innerhalb der Endlosschleife läuft eine weitere Schleife je achtmal, wobei in jedem Durchlauf eine LED eingeschaltet wird.

```
015       bus.write_byte_data(DEVICE,GPIOA,j)
```

Dazu wird der aktuelle Wert `j` in das `GPIOA`-Datenregister geschrieben. Im ersten Durchlauf hat das erste Bit den Wert `1`, die anderen sieben Bit den Wert `0`.

```
016       j *= 2
```

Danach wird der Wert `j` verdoppelt und damit das nächsthöhere Bit auf `1` gesetzt.

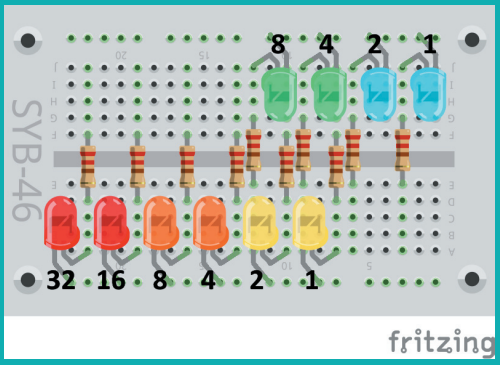
```
017       time.sleep(0.1)
```

Dieser Zustand wird für 0,1 Sekunden beibehalten. Dann startet der nächste Schleifendurchlauf und die nächste LED leuchtet für 0,1 Sekunden. Nach acht Durchläufen der inneren Schleife startet der nächste Durchlauf der äußeren Schleife, wobei der Wert `j` wieder mit `1` startet und wieder die erste LED der Reihe blinkt.

17 BINÄRUHR

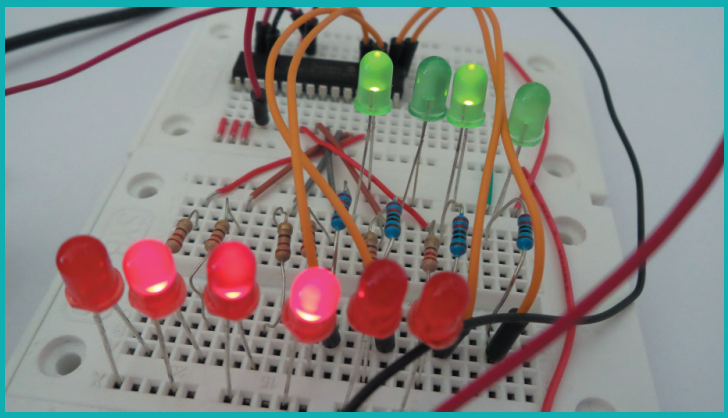
Binäre Uhren zeigen die Uhrzeit in binärcodierter Form anstatt in Ziffern an. Solche Uhren sehen einfach cool aus, obwohl oder gerade weil das Ablesen für Anfänger etwas gewöhnungsbedürftig ist. So bietet z. B. der Uhrenhersteller ‚01 the one' (*www.01theone.com*) solche Binäruhren als Armbanduhren an.

Das Prinzip der Zeitdarstellung ist einfach. Jede LED steht für ein Bit der Binärzahl. Zur Darstellung der maximal 12 Stunden werden vier LEDs benötigt (8, 4, 2, 1), zur Darstellung der maximal 59 Minuten sechs LEDs (32, 16, 8, 4, 2, 1). Jetzt braucht man nur noch die Werte der leuchtenden LEDs zusammenzuaddieren und man erhält die Werte für Stunden und Minuten. Die Zeit wird wie auf einer Analoguhr im 12-Stunden-Format angezeigt.



Aufbau einer Binäruhr mit zehn LEDs

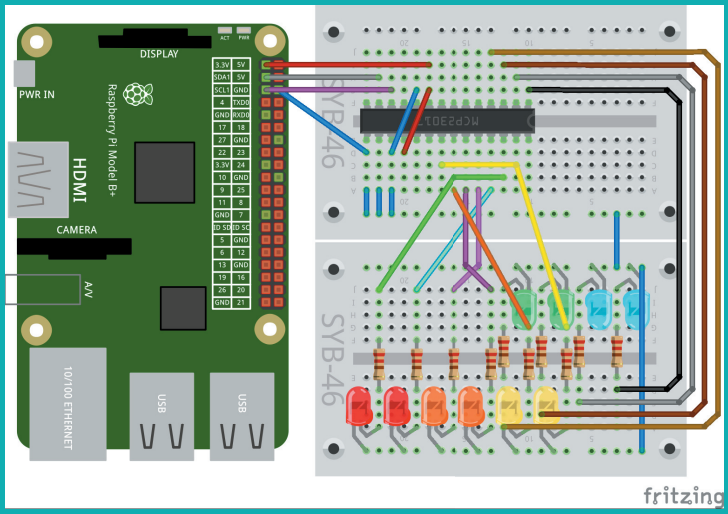
In diesem Experiment bauen wir eine solche Binäruhr aus zehn LEDs auf, die über den Portexpander gesteuert werden.



Auf der Uhr ist es 10:28. Die Widerstände haben alle den gleichen Wert, die vier Vorwiderstände der LEDs zur Stundenanzeige haben im Bild eine andere Farbe, nur um sie besser zu erkennen.

Benötigte Bauteile

2x Steckplatine, 1x Portexpander MCP23017, 2x LED rot, 2x LED orange, 2x LED gelb, 2x LED grün, 2x LED blau, 10x 220-Ohm-Widerstand (Rot-Rot-Braun), 4x Verbindungskabel, 17x Drahtbrücke (unterschiedliche Längen)



Binäruhr mit Portexpander MCP23017

Die Anschlussdrähte der LEDs verlaufen diesmal nicht parallel von den Vorwiderständen zu den Ports des Portexpanders, sondern genau über Kreuz. Dies hat den Vorteil, dass die LED ganz rechts mit dem niedrigsten Bitwert auch mit dem Port mit dem niedrigsten Bitwert verbunden ist.

17

Die weiteren LEDs folgen mit aufsteigenden Bitwerten, was die Programmierung deutlich vereinfacht. Die Tabelle zeigt, welche LED wo angeschlossen ist.

LED	Port am Portexpander
Stunde 8	GPIOB3
Stunde 4	GPIOB2
Stunde 2	GPIOB1
Stunde 1	GPIOB0
Minute 32	GPIOA5
Minute 16	GPIOA4
Minute 8	GPIOA3
Minute 4	GPIOA2
Minute 2	GPIOA1
Minute 1	GPIOA0

Die Anschlüsse für Adressleitungen und die des MCP23017 sind unverändert aus dem letzten Experiment übernommen.

Das Programm `binaeruhr.py` ist sehr einfach gehalten, da der Portexpander von sich aus bereits mit binären Daten arbeitet.

Programmdatei:
binaeruhr.py

```
001 #!/usr/bin/python
002 import time, smbus
003
004 DEVICE = 0x20
005 IODIRA = 0x00
006 IODIRB = 0x01
007 GPIOA  = 0x12
008 GPIOB  = 0x13
009
010 bus = smbus.SMBus(1)
011 bus.write_byte_data(DEVICE,IODIRA,0x00)
012 bus.write_byte_data(DEVICE,IODIRB,0x00)
013 bus.write_byte_data(DEVICE,GPIOA,0x00)
014 bus.write_byte_data(DEVICE,GPIOB,0x00)
015
016 m1 = 60
```

```
017
018 while True:
019   zeit = time.localtime()
020   m = zeit.tm_min
021   h = zeit.tm_hour
022   if h > 12:
023     h = h - 12
024   if m1 <> m:
025     bus.write_byte_data(DEVICE,GPIOB,h)
026     bus.write_byte_data(DEVICE,GPIOA,m)
027     m1 = m
028   time.sleep(1)
```

17.1 | So funktioniert das Programm

Die Initialisierung des i2c-Busses läuft weitgehend wie im vorherigen Experiment, mit dem Unterschied, dass diesmal beide GPIO-Ports des Portexpanders verwendet werden.

DEVICE = 0x20	i2c-Geräteadresse des Portexpanders
IODIRA = 0x00	8-Bit-Register, das die Richtung (Ausgang/Eingang) der GPIOA-Ports festlegt
IODIRB = 0x01	8-Bit-Register, das die Richtung (Ausgang/Eingang) der GPIOB-Ports festlegt
GPIOA = 0x12	8-Bit-Datenregister der GPIOA-Ports
GPIOB = 0x13	8-Bit-Datenregister der GPIOB-Ports

Das Programm läuft in einer Endlosschleife, die regelmäßig die aktuelle Uhrzeit des Raspberry Pi ausliest. Unterscheidet sich die aktuelle Minute von der zuletzt auf den LEDs dargestellten Minute, werden andere LEDs eingeschaltet.

```
018 while True:
019   zeit = time.localtime()
```

In jedem Durchlauf wird die aktuelle Zeit in das Objekt `zeit` geschrieben. Dazu wird die Funktion `time.localtime()` aus der time-Bibliothek verwendet. Das Ergebnis ist eine Datenstruktur, die aus verschiedenen Einzelwerten besteht.

17

```
020   m = zeit.tm_min
021   h = zeit.tm_hour
```

Die beiden für die Digitaluhr relevanten Werte, Minuten und Stunden werden aus der Struktur in die Variablen `m` und `h` geschrieben.

```
022   if h > 12:
023     h = h - 12
```

Ist die Stundenangabe im 24-Stunden-Format größer als 12, wird 12 subtrahiert, um die Zeit im 12-Stunden-Format in der Variablen `h` zu speichern.

```
024   if m1 <> m:
025     bus.write_byte_data(DEVICE,GPIOB,h)
026     bus.write_byte_data(DEVICE,GPIOA,m)
027     m1 = m
```

Hat die aktuelle Minute `m` einen anderen Wert als die zuletzt dargestellte Minute `m1`, wird die Stunde auf den Port `GPIOB` und die Minute auf den Port `GPIOA` geschrieben. Hier können direkt die Zahlenwerte verwendet werden, der Portexpander rechnet diese automatisch in Binärzahlen um und gibt sie entsprechend auf den Ports aus.

Danach wird `m1` auf die gerade dargestellte Minute gesetzt. Vor dem ersten Start der Schleife bekommt `m1` den Wert `60`, den die Minutenanzeige im laufenden Betrieb nie erreicht. Damit wird sichergestellt, dass unabhängig von der aktuellen Uhrzeit bereits im ersten Schleifendurchlauf eine neue Zeit ermittelt und über die LEDs angezeigt wird.

```
028   time.sleep(1)
```

Am Ende der Schleife wartet das Programm eine Sekunde. Damit wird verhindert, dass die Schleife in Intervallen von Sekundenbruchteilen immer wieder aufgerufen wird und damit den Prozessor so stark auslastet, dass der Raspberry Pi für nichts anderes als dieses Programm zu nutzen ist.

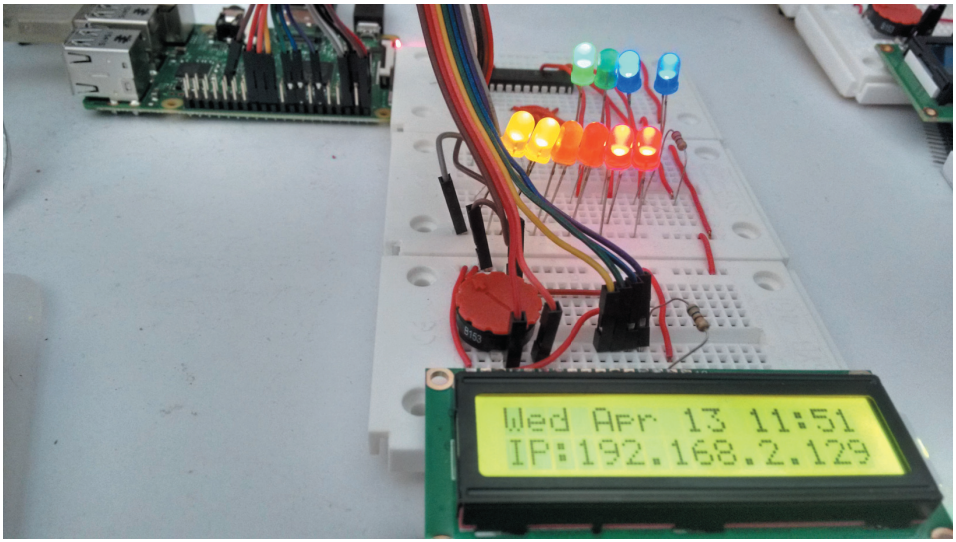
17.2 | Binäruhr + LCD-Uhr

Die Binäruhr benötigt auf dem Raspberry Pi nur die beiden GPIO-Ports `2` und `3` zur Ansteuerung des Portexpanders über den i2c-Bus. Alle anderen GPIO-Ports bleiben ungenutzt, auch die, die wir in früheren Experimenten zur Steuerung des LC-Displays verwendet haben. Daher lassen

sich beide Schaltungen zu einer kombinieren und beide Programme zu einem, das die aktuelle Zeit parallel auf der Binäruhr und dem LC-Display zeigt. Für den Schaltungsaufbau benötigen Sie fast alle Bauteile aus dem Paket.

Zwei verschiedene Stromversorgungen

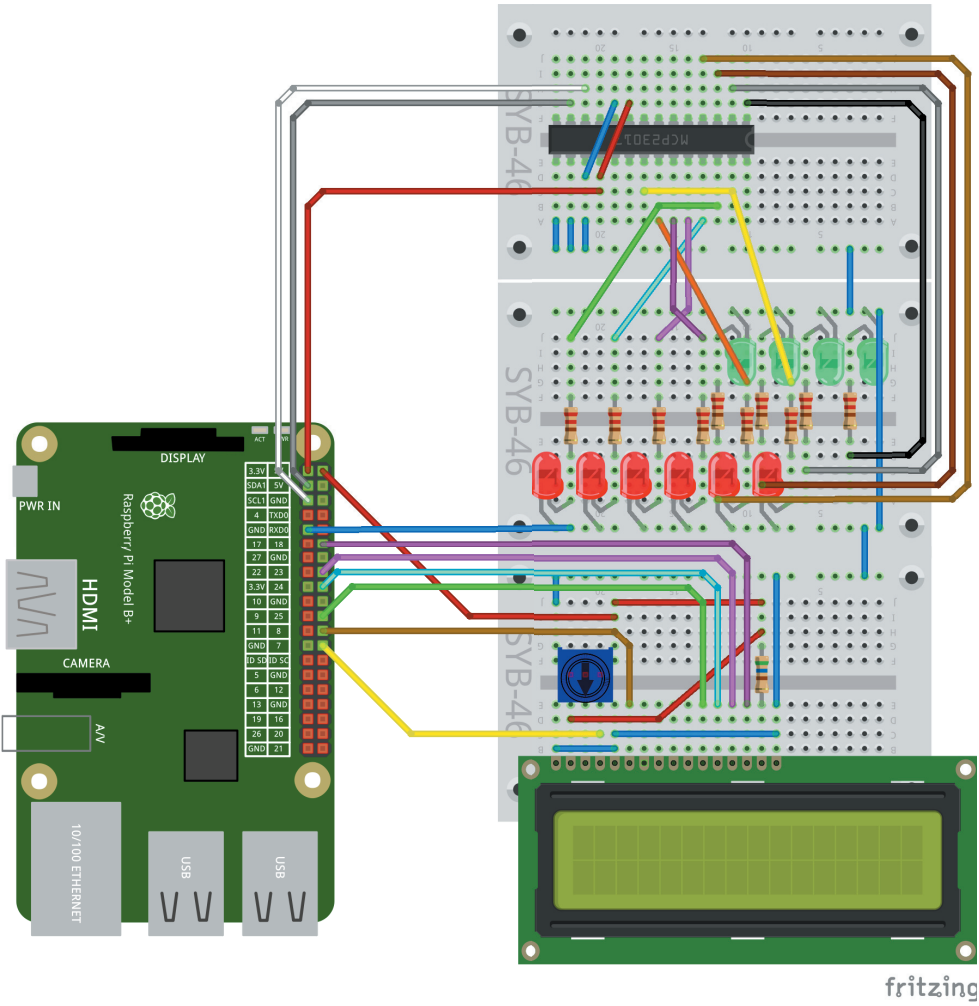
Achten Sie beim Aufbau besonders genau auf den korrekten Anschluss der Stromversorgung, da die Schaltung beide Stromversorgungsleitungen des Raspberry Pi nutzt, +3,3 V für den Portexpander und +5 V für das LC-Display.



Binäruhr mit zusätzlicher LCD-Anzeige

Benötigte Bauteile

3x Steckplatine, 1x Portexpander MCP23017, 1x LC-Display, 1x 560-Ohm-Widerstand (Grün-Blau-Braun), 1x Potenziometer, 2x LED rot, 2x LED orange, 2x LED gelb, 2x LED grün, 2x LED blau, 10x 220-Ohm-Widerstand (Rot-Rot-Braun), 11x Verbindungskabel, 24x Drahtbrücke (unterschiedliche Längen)



Binäruhr mit Port-expander MCP23017 und LCD-Uhr in einer Schaltung

17.2.1 | So funktioniert das Programm

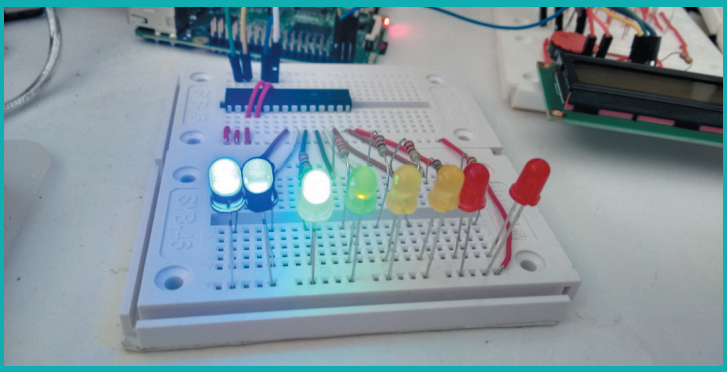
Programmdatei:
binaer-lcd-uhr.py

Das Programm `binaer-lcd-uhr.py` kombiniert die beiden Programme der Binäruhr und der LCD-Uhr.

Das Programm enthält keine neuen Funktionen. Nach den Initialisierungen für das Display wird der i2c-Bus initialisiert. In der Hauptschleife folgen nacheinander die bekannten Routinen für die Zeitanzeige auf der Binäruhr und dem Display.

18 CPU-LASTANZEIGE MIT LEDS AM PORTEXPANDER

Das nächste Experiment liefert auf acht LEDs eine Pegelanzeige der aktuellen CPU-Auslastung.



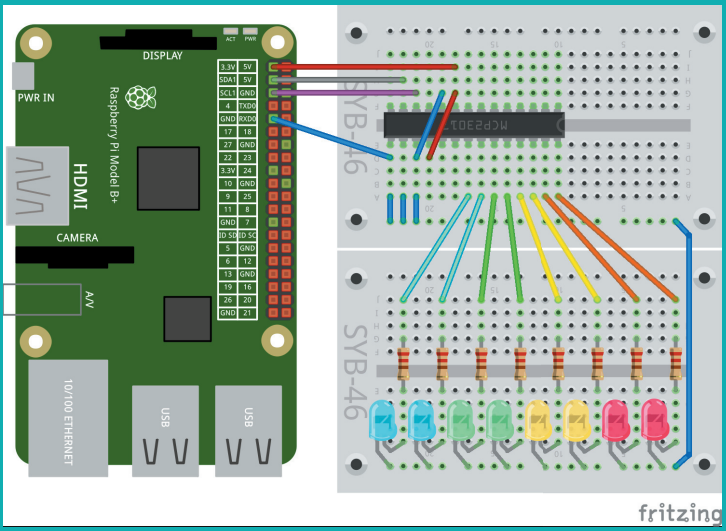
Pegelanzeige der CPU-Last

Bauen Sie die Schaltung wie in der Abbildung auf. Der Schaltungsaufbau entspricht dem ersten Experiment mit dem Portexpander, wo ebenfalls einfach acht LEDs an den acht Pins der GPIOA-Schnittstelle angeschlossen sind.

Benötigte Bauteile

2x Steckplatine, 1x Portexpander MCP23017, 2x LED rot, 2x LED gelb, 2x LED grün, 2x LED blau, 8x 220-Ohm-Widerstand (Rot-Rot-Braun), 4x Verbindungskabel, 14x Drahtbrücke (unterschiedliche Längen)

Je nach CPU-Auslastung leuchten ein bis acht LEDs. Das Programm `cpu.py` basiert auf dem Programm aus dem Experiment mit dem Lauflicht und verwendet zusätzlich einige interessante Funktionen zur Berechnung der CPU-Last.



CPU-Lastanzeige mit Portexpander MCP23017

Raspberry Pi 3

Beim gegenüber den Vorgängermodellen deutlich leistungsstärkeren Raspberry Pi 3 lässt sich die CPU-Auslastung durch einfache Mausbewegungen oder das Starten von Programmen kaum höher als auf einen einstelligen Prozentwert hochtreiben, sodass auf der CPU-Lastanzeige meistens nichts zu sehen ist. Öffnen Sie im Browser eine aufwendige Webseite, wie z. B. *www.youtube.com* oder starten Sie in Mathematica eines der komplexeren Demos, um ein bisschen CPU-Last zu sehen. Beim Raspberry Pi B+ und Pi 2 lässt sich bereits durch das Öffnen von Programmen oder durch hektische Mausbewegungen erkennbare CPU-Last erzeugen.

```
001 #!/usr/bin/python
002 import time, os, smbus
003
004 DEVICE = 0x20
005 IODIRA = 0x00
006 GPIOA  = 0x12
007
008 bus = smbus.SMBus(1)
009 bus.write_byte_data(DEVICE,IODIRA,0x00)
010 bus.write_byte_data(DEVICE,GPIOA,0x00)
011
```

Programmdatei:

cpu.py

18

```
012 while True:
013     cpu1 = os.popen("vmstat 1 3").readlines()
014     cpu2 = 2 ** int((100 - int(cpu1[4].split()[14])) * 0.08) - 1
015     bus.write_byte_data(DEVICE,GPIOA,cpu2)
```

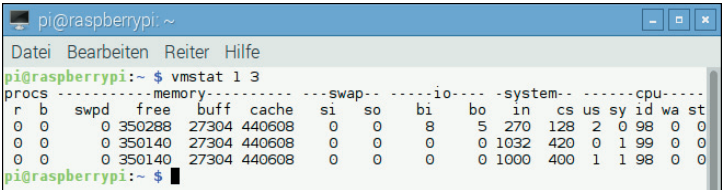
18.1 | So funktioniert das Programm

Die ersten Programmzeilen zur Initialisierung des i2c-Busses sind bereits bekannt. In diesem Fall wird nur zusätzlich das Modul `os` zum Zugriff auf Systemfunktionen importiert.

```
012 while True:
013     cpu1 = os.popen("vmstat 1 3").readlines()
```

Das Programm läuft in einer Endlosschleife. Diese ruft in jedem Durchlauf zuerst das Linux-Kommando `vmstat 1 3` auf, womit sich diverse Statusangaben des Systems auslesen lassen. Der Parameter `3` bedeutet hier, dass dreimal hintereinander die Statusdaten gelesen werden. Beim ersten Lesen ergeben sich verfälschte Informationen, da der Aufruf des Kommandos selbst einiges an Systemleistung frisst.

Die Funktion `os.popen()` schreibt das Ergebnis eines Linux-Kommandos zeilenweise in ein Array mit Zeichenketten, hier die Variable `cpu1`.



Ausgabe des Befehls »vmstat 1 3« in einem Konsolenfenster

Die nächste Zeile ermittelt mit einer interessanten Berechnungsformel den Wert, der mit den LEDs angezeigt werden soll, und speichert diesen in der Variablen `cpu2`.

```
014     cpu2 = 2 ** int((100 - int(cpu1[4].split()[14])) * 0.08) - 1
```

Die dritte Statusabfrage steht nach den beiden Tabellenüberschriften in der fünften Zeile und damit im fünften Element des Arrays, das bei Zählung ab `0` die Nummer `4` trägt.

```
cpu1[4].split()
```

Diese Zeichenkette wird mit der Methode `split()` an den Leerzeichen in einzelne Zeichenketten aufgespalten. Interessant ist das Element der

Nummer `[14]`, das – wieder bei `0` mit der Zählung beginnend – den Inhalt der Spalte `id` enthält. Diese Spalte gibt die ungenutzte CPU-Zeit in Prozent an.

```
100 - int(cpu1[4].split()[14])
```

Subtrahiert man diesen Wert von 100, erhält man die verbrauchte CPU-Zeit in Prozent, also die Auslastung der CPU. Zuvor wird die Zeichenkette mit der `int()`-Funktion in eine Ganzzahl umgewandelt.

```
(100 - int(cpu1[4].split()[14])) * 0.08)
```

Eine Multiplikation mit dem Faktor `0,08` ergibt zur Darstellung auf den acht LEDs eine Zahl zwischen 0 und 8 statt zwischen 0 und 100. Zur Darstellung über den Portexpander braucht man 8-Bit-Zahlen. In diesem Fall einer Pegelanzeige sollen LEDs in durchgehender Folge leuchten. Die Tabelle zeigt alle Zahlen zwischen 0 und 8, die in der Binärdarstellung eine durchgehende Folge von Einsen enthalten.

Binär	Dezimal	Berechnungsformel
0000 0000	0	2 ** 0 - 1
0000 0001	1	2 ** 1 - 1
0000 0011	3	2 ** 2 - 1
0000 0111	7	2 ** 3 - 1
0000 1111	15	2 ** 4 - 1
0001 1111	31	2 ** 5 - 1
0011 1111	63	2 ** 6 - 1
0111 1111	127	2 ** 7 - 1
1111 1111	255	2 ** 8 - 1

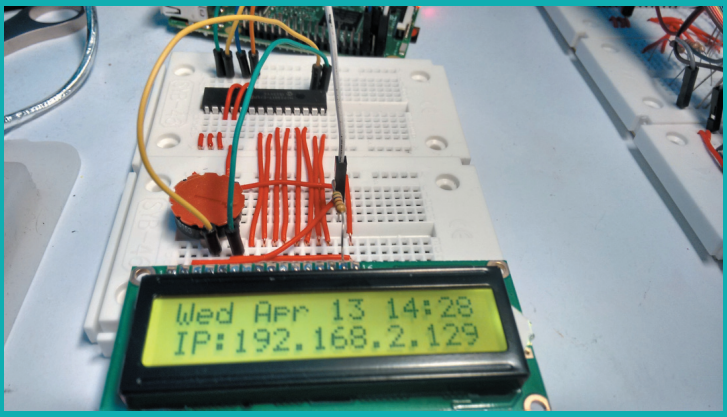
Über die Berechnungsfunktion in der rechten Spalte der Tabelle werden die Werte ermittelt, die binärcodiert die entsprechenden LEDs ansteuern. Das Symbol `**` steht in Python für die Exponentialfunktion.

```
015     bus.write_byte_data(DEVICE,GPIOA,cpu2)
```

Das in der Variablen `cpu2` gespeicherte Ergebnis wird auf den GPIOA-Port des Portexpanders geschrieben und mit den LEDs angezeigt. Danach startet die Endlosschleife neu und ermittelt wiederum die aktuelle CPU-Last.

19 LC-DISPLAY AM PORTEXPANDER

Das LC-Display nutzt je nach eingestelltem Modus vier oder acht Datenleitungen, auf denen es bitcodierte Daten empfängt. Der Portexpander verfügt ebenfalls über zweimal acht Datenleitungen, auf denen bitcodierte Daten gesendet werden können. Da bietet es sich geradezu an, das LC-Display mit dem Portexpander zu verbinden. Man braucht zum Anschluss des Displays am Raspberry Pi neben der Stromversorgung nur noch zwei Leitungen für den i2c-Bus und nicht mehr sechs bzw. zehn GPIO-Ports.

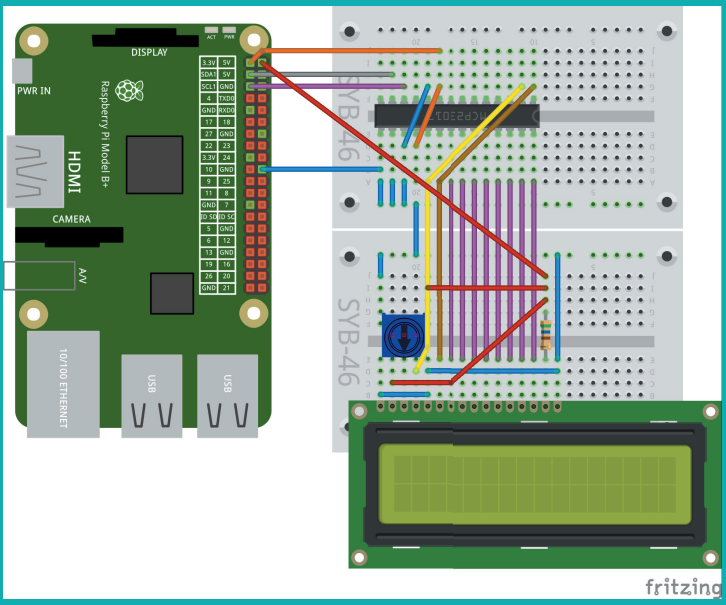


LC-Display mit Portexpander am Raspberry Pi

Für den Anschluss am Portexpander empfiehlt sich der 8-Bit-Modus des LC-Displays, da hier die Zeichen direkt byteweise anstatt in zwei Blöcken auf das Display geschrieben werden können, was das Programm deutlich vereinfacht. Die acht Datenleitungen des GPIOA-Ports des Portexpanders werden eins zu eins mit den Datenleitungen des LC-Displays verbunden. Zwei der Leitungen des GPIOB-Ports werden für die Steuersignale *RS* und *E* verwendet.

Benötigte Bauteile

2x Steckplatine, 1x Portexpander MCP23017, 1x LC-Display, 1x 560-Ohm-Widerstand (Grün-Blau-Braun), 1x Potenziometer, 5x Verbindungskabel, 22x Drahtbrücke (unterschiedliche Längen)



LC-Display am Portexpander

Zwei verschiedene Stromversorgungen

Achten Sie beim Aufbau besonders genau auf den korrekten Anschluss der Stromversorgung, da die Schaltung beide Stromversorgungsleitungen des Raspberry Pi nutzt: +3,3 V für den Portexpander und +5 V für das LC-Display.

Das Programm `i2c_display.py` übernimmt die grundlegende Struktur der Statusanzeige `status.py`, wobei einige Funktionen für die Steuerung über den Portexpander angepasst wurden. Zusätzlich sind die bereits bekannten Initialisierungsroutinen für den i2c-Bus enthalten, dafür fällt die Initialisierung der GPIO-Schnittstelle weg.

```
001 #!/usr/bin/python
002 import time, smbus, subprocess
003
```

Programmdatei:
i2c_display.py

19

```
004 DEVICE = 0x20
005 IODIRA = 0x00
006 IODIRB = 0x01
007 GPIOA  = 0x12
008 GPIOB  = 0x13
009
010 LCD_WIDTH = 16
011 LCD_LINE_1 = 0x80
012 LCD_LINE_2 = 0xC0
013 LCD_CHR = 1
014 LCD_CMD = 0
015 LCD_RS = 0x02
016 LCD_E  = 0x01
017 E_PULSE = 0.00005
018 E_DELAY = 0.00005
019 INIT  = 0.01
020 pause = 2
021
022 def lcd_byte(bits, mode):
023   bus.write_byte_data(DEVICE,GPIOB,LCD_RS*mode)
024   bus.write_byte_data(DEVICE,GPIOA,bits)
025   time.sleep(E_DELAY)
026   bus.write_byte_data(DEVICE,GPIOB,LCD_E+LCD_RS*mode)
027   time.sleep(E_PULSE)
028   bus.write_byte_data(DEVICE,GPIOB,LCD_RS*mode)
029   time.sleep(E_DELAY)
030
031 def lcd_string(message):
032   message = message.ljust(LCD_WIDTH," ")
033   for i in range(LCD_WIDTH):
034     lcd_byte(ord(message[i]),LCD_CHR)
035
036 bus = smbus.SMBus(1)
037 bus.write_byte_data(DEVICE,IODIRA,0x00)
038 bus.write_byte_data(DEVICE,IODIRB,0x00)
039 bus.write_byte_data(DEVICE,GPIOA,0x00)
040 bus.write_byte_data(DEVICE,GPIOB,0x00)
041
042 LCD_INIT = [0x33, 0x32, 0x38, 0x0C, 0x06, 0x01]
043 for i in LCD_INIT:
044   lcd_byte(i,LCD_CMD)
045   time.sleep(INIT)
046
```

```
047 while True:
048   lcd_byte(LCD_LINE_1, LCD_CMD)
049   lcd_string(time.asctime())
050   lcd_byte(LCD_LINE_2, LCD_CMD)
051   lcd_string("IP:" + subprocess.check_
    output(["hostname","-I"])[:-2])
052   time.sleep(pause)
```

19.1 | So funktioniert das Programm

Am Anfang werden die Module `smbus` für den i2c-Bus sowie `time` und `subprocess` importiert, die für die Statusanzeigen benötigt werden. Die GPIO-Bibliothek wird nicht gebraucht.

Die Konstanten für den Portexpander sind bereits bekannt, die Konstanten für das LC-Display wurden weitestgehend übernommen, um die Änderungen am Programmcode so gering wie möglich zu halten.

```
015 LCD_RS = 0x02
016 LCD_E  = 0x01
```

Die beiden Signale `LCD_RS` (*Register Select*) und `LCE_E` (*Enable*) sind jetzt keine GPIO-Pins mehr, sondern Bitcodes, die über den GPIOB-Port an die entsprechenden Steuerleitungen des LC-Displays gesendet werden. Die ehemaligen Konstanten für die vier oder acht Datenleitungen sowie die Initialisierung der GPIO-Ports entfallen ersatzlos.

Die größten Änderungen finden sich in der Funktion `lcd_byte()`, die der Einfachheit halber mit der ehemaligen Funktion `lcd_enable()` zusammengefasst wurde.

```
022 def lcd_byte(bits, mode):
```

Der Funktionsaufruf bleibt kompatibel zum ursprünglichen Programm, damit braucht das Hauptprogramm nicht verändert zu werden. Die Funktion `lcd_byte()` bekommt weiterhin zwei Parameter. Im Parameter `bits` bekommt die Funktion das zu sendende Byte, der Parameter `mode` gibt an, ob es sich um ein Zeichen oder um einen Steuerungsbefehl handelt. Die beiden Werte, die `mode` annehmen kann, sind in den Konstanten `LCD_CHR` (Zeichen) und `LCD_CMD` (Steuerungsbefehl) festgelegt.

```
023   bus.write_byte_data(DEVICE,GPIOB,LCD_RS*mode)
```

19

Als Erstes wird das Display auf Kommandomodus oder Zeichenmodus gesetzt. Dazu wird das Bitmuster für das zweite Bit des GPIOB-Ports (`0x02`), das in der Konstanten `LCD_RS` gespeichert ist, mit dem Parameter `mode`, der 0 oder 1 sein kann, multipliziert. Auf diese Weise wird dieses Bit für den Kommandomodus ausgeschaltet und für den Zeichenmodus eingeschaltet.

```
024    bus.write_byte_data(DEVICE,GPIOA,bits)
```

Diese Zeile schreibt das an das Display zu sendende Byte auf den GPIOA-Port. Bei Verwendung des Portexpanders entfällt die Umrechnung des Zeichens in Bitcode.

```
025    time.sleep(E_DELAY)
026    bus.write_byte_data(DEVICE,GPIOB,LCD_E+LCD_RS*mode)
```

Jetzt braucht das Display das Enable-Signal, einen Wechsel von `1` auf `0` auf der *E*-Leitung, um die Daten auf den Datenleitungen einzulesen und darzustellen. Nach einer kurzen Verzögerung, um die Displayträgheit zu überbrücken, wird das in der Konstanten `LCD_E` festgelegte erste Bit des `GPIOB`-Ports auf `1` gesetzt. Da auf dem Portexpander immer ein komplettes Byte auf einmal ausgegeben werden muss und der zuvor gesetzte Modus nicht verändert werden darf, wird dieser Modus zum Wert `LCD_E` addiert und so wieder mitgesendet.

```
027    time.sleep(E_PULSE)
028    bus.write_byte_data(DEVICE,GPIOB,LCD_RS*mode)
```

Nach einer weiteren kurzen Verzögerung wird das Bit `LCD_E` wieder auf `0` gesetzt. Dazu sendet die Funktion einfach nur den aktuellen Modus `0x02` oder `0x00` an das Display. Dabei ist das Bit `LCD_E` automatisch `0`.

```
029    time.sleep(E_DELAY)
```

Wie im ursprünglichen Programm folgt eine kurze Verzögerung, damit keine neuen Daten am Display ankommen können, bevor die aktuellen Daten dargestellt sind.

Die Funktion `lcd_string()` greift nicht direkt auf die Hardware des LC-Displays zu und kann daher unverändert übernommen werden.

```
036 bus = smbus.SMBus(1)
037 bus.write_byte_data(DEVICE,IODIRA,0x00)
038 bus.write_byte_data(DEVICE,IODIRB,0x00)
039 bus.write_byte_data(DEVICE,GPIOA,0x00)
040 bus.write_byte_data(DEVICE,GPIOB,0x00)
```

Nach der Definition der Funktionen beginnt das eigentliche Programm mit der bereits bekannten Initialisierung des i2c-Bus sowie der beiden Ports des Portexpanders `GPIOA` und `GPIOB`.

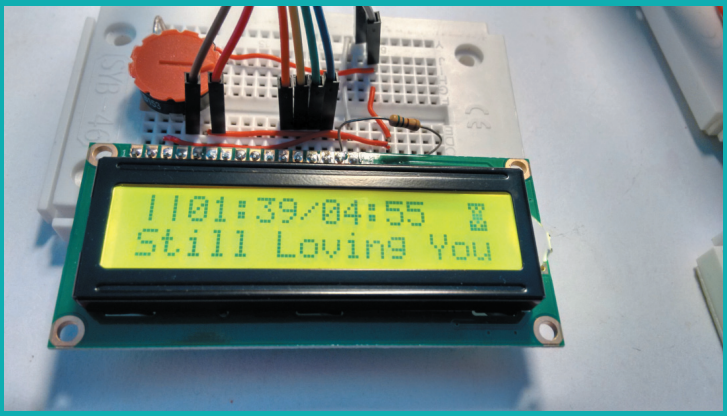
```
042 LCD_INIT = [0x33, 0x32, 0x38, 0x0C, 0x06, 0x01]
```

Der Initialisierungsstring für das Display enthält diesmal den Code `0x38` für den 8-Bit-Modus. Die Schleife zur Initialisierung des Displays sowie die Hauptschleife des Programms können unverändert übernommen werden, da auch diese nur über Funktionen und nicht direkt auf die Hardware des LC-Displays zugreifen.

20 LC-DISPLAY FÜR OSMC MEDIA CENTER

Ein Raspberry Pi ist leistungsfähig genug, Videos in HD-Qualität abzuspielen. Die Software OSMC (`www.osmc.tv`) macht aus dem Raspberry Pi ein komfortables Media Center für das Wohnzimmer. Das Programm basiert auf *kodi*, dem Nachfolger von xbmc, einem bekannten Media Center für PCs.

OSMC verwendet eine ganz eigene grafische Oberfläche, die speziell zur Benutzung auf einem Fernseher optimiert ist und sowohl per Maus und Tastatur, wie auch mit einer drahtlosen Fernbedienung gesteuert werden kann. Es läuft nicht auf dem normalen LXDE-Desktop, obwohl es im Hintergrund auf Raspbian basiert, und benötigt daher eine eigene bootfähige Speicherkarte. Die Mediendaten können auf dieser Speicherkarte, auf USB-Sticks, angeschlossenen externen Festplatten, auf Netzwerklaufwerken oder auch bei Clouddiensten liegen. Zusätzlich zu seiner grafischen Oberfläche bietet OSMC mit einem Zusatzmodul auch die Möglichkeit, laufende Tracks und andere Informationen auf einem LC-Display anzuzeigen.

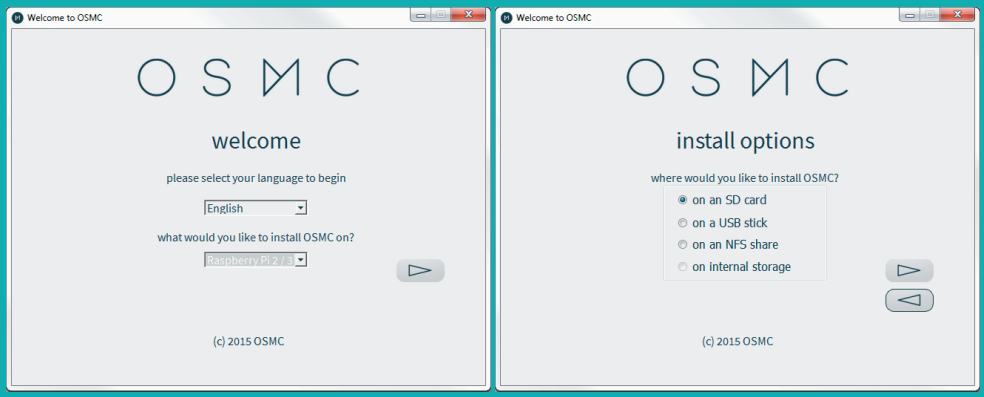


Titelanzeige für OSMC auf einem LC-Display

20.1 | OSMC auf dem Raspberry Pi installieren

OSMC steht als Auswahl im Betriebssysteminstaller NOOBS zur Verfügung, muss dort aber erst nachgeladen werden, was auf dem Raspberry Pi lange dauert. Einfacher ist es, bei *osmc.tv/download* einen eigenen Installer für Windows herunterzuladen, mit dem man auf einem Windows-PC Speicherkarten für den Raspberry Pi vorbereiten kann.

1. Wählen Sie im ersten Schritt des Installers das verwendete Raspberry-Pi-Modell aus. Beachten Sie, dass OSMC in zwei Varianten geliefert wird, eine für Raspberry Pi B, B+ und Zero, sowie eine zweite für Raspberry Pi 2 und 3.
2. Im zweiten Schritt wählen Sie die gewünschte OSMC-Version. In den meisten Fällen ist die aktuellste Version die beste Wahl.
3. Für die Installation auf dem Raspberry Pi wählen Sie die Option *on a SD card*. Die anderen Optionen funktionieren nur auf anderen Geräteplattformen.

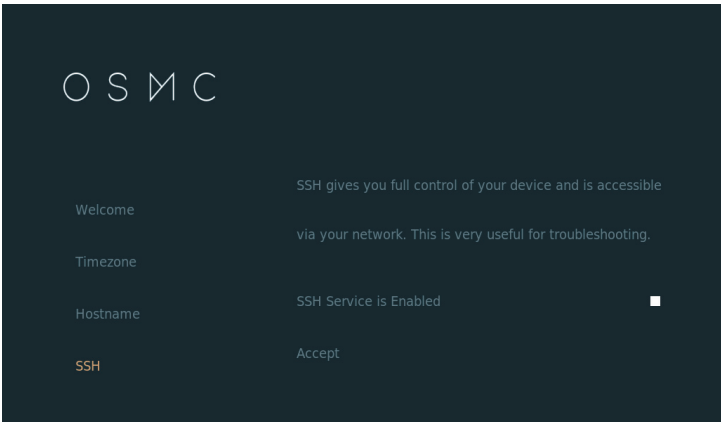


Raspberry-Pi-Modell und Speicherkarte auswählen

4. Wählen Sie im nächsten Schritt, ob der Raspberry Pi über ein Netzwerkkabel oder über WLAN mit dem Netzwerk verbunden ist. Der OSMC-Installer legt dann gleich die passenden Konfigurationsdateien an.
5. Jetzt brauchen Sie nur noch die Speicherkarte auszuwählen, die im Kartenleser automatisch erkannt wird. Danach starten Download der Software und Installation auf der Speicherkarte automatisch. Das dabei heruntergeladene OSMC-Betriebssystemimage ist mit etwa 250 MB deutlich kleiner als ein Raspbian-Image.

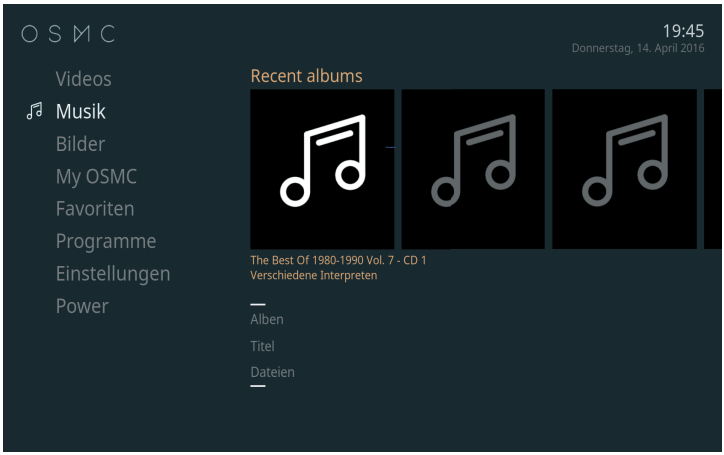
20

Stecken Sie die Speicherkarte, nachdem sie fertiggestellt ist, in den Raspberry Pi und booten Sie damit. Jetzt werden noch weitere Dateien heruntergeladen und eine kurze Installationsroutine wird gestartet. Nach Abschluss der Installation erscheint auf dem Raspberry Pi die OSMC-Oberfläche, wo Sie als Erstes die Sprache auswählen müssen. Wundern Sie sich nicht, wenn die Bedienung sehr wenig flüssig läuft. OSMC lädt beim ersten Start automatisch noch einige Updates nach und installiert diese, was die CPU erheblich fordert. Warten Sie einfach noch kurz mit der Auswahl der Sprache. Anschließend müssen Sie noch die Zeitzone einstellen, damit die Uhr richtig läuft. Geben Sie dem Raspberry Pi noch einen Namen oder bestätigen Sie die Vorgabe und achten Sie darauf, dass im letzten Schritt der Installation der Schalter *SSH Service is Enabled* eingeschaltet ist.



Die Ersteinrichtung von OSMC

Danach erscheint das Hauptmenü von OSMC, wo Sie Videos, Musik und Bilder betrachten können. Diese Oberfläche lässt sich mit der Maus oder den Pfeiltasten der Tastatur bedienen. Haben Sie einen HDMI-Monitor mit Fernbedienung, können Sie diese auch für OSMC nutzen. Der Trick heißt CEC (Consumer Electronics Control) und ist ein einfacher Datenbus auf einer zusätzlichen Leitung im HDMI-Kabel. Darüber werden die Signale der Fernbedienung an den Raspberry Pi übertragen. Der verwendete Monitor und auch das Kabel müssen CEC unterstützen. Hersteller von Fernsehern und Monitoren vermarkten CEC oft unter eigenen Markennamen wie T-Link, EasyLink, EZ-Sync, Simplink, Digital Link HD, NetCommand for HDMI, RIHD, Viera Link, Kuro Link, Anynet+, Aquos Link, BRAVIA Sync, Regza-Link, TechniLink, CSTLink, FUN-Link oder Digi-Link. Alle diese Markennamen bezeichnen prinzipiell die gleiche Technik.



Das Hauptmenü des OSMC-Mediacenters

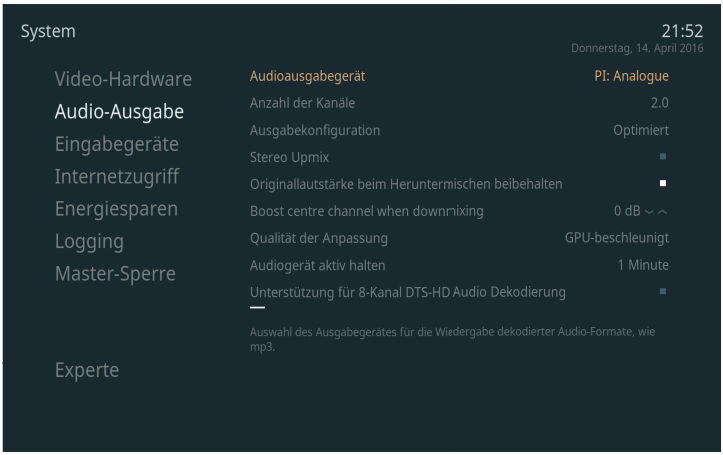
Zurück zum Hauptbildschirm

Da sich die Bedienung des OSMC weitgehend von selbst erklärt, gehen wir hier nicht näher darauf ein. In den verschachtelten Dialogen kann man sich nicht verlieren. Mit der Esc-Taste kommen Sie immer eine Ebene zurück. Beim Zurückspringen werden alle vorgenommenen Einstellungen automatisch gespeichert.

Die meisten Einstellungen werden von OSMC anhand der angeschlossenen Hardware automatisch vorgenommen. Die Audio-Einstellungen müssen Sie unter Umständen manuell anpassen, wenn Sie einen Computermonitor mit DVI-Anschluss nutzen und beim Abspielen von Video und Musik nichts hören. Der Raspberry Pi versucht, wenn ein Monitor am HDMI-Ausgang angeschlossen ist, immer das Audiosignal auch über diesen Ausgang abzuspielen, selbst wenn dies bei einem DVI-Monitor nicht möglich ist. Bei diesen Monitoren brauchen Sie aber externe Lautsprecher, die an der 3,5-mm-Klinkenbuchse angeschlossen sind.

Wählen Sie zur Einstellung der Audio-Ausgabe im Hauptmenü *Einstellungen*, wählen Sie im Einstellungsdialog *System* und dann auf dem nächsten Bildschirm *Audio-Ausgabe*. Stellen Sie die Audio-Ausgabe auf *PI:Analogue*, wenn Sie externe Lautsprecher oder Kopfhörer am Raspberry Pi angeschlossen haben.

20



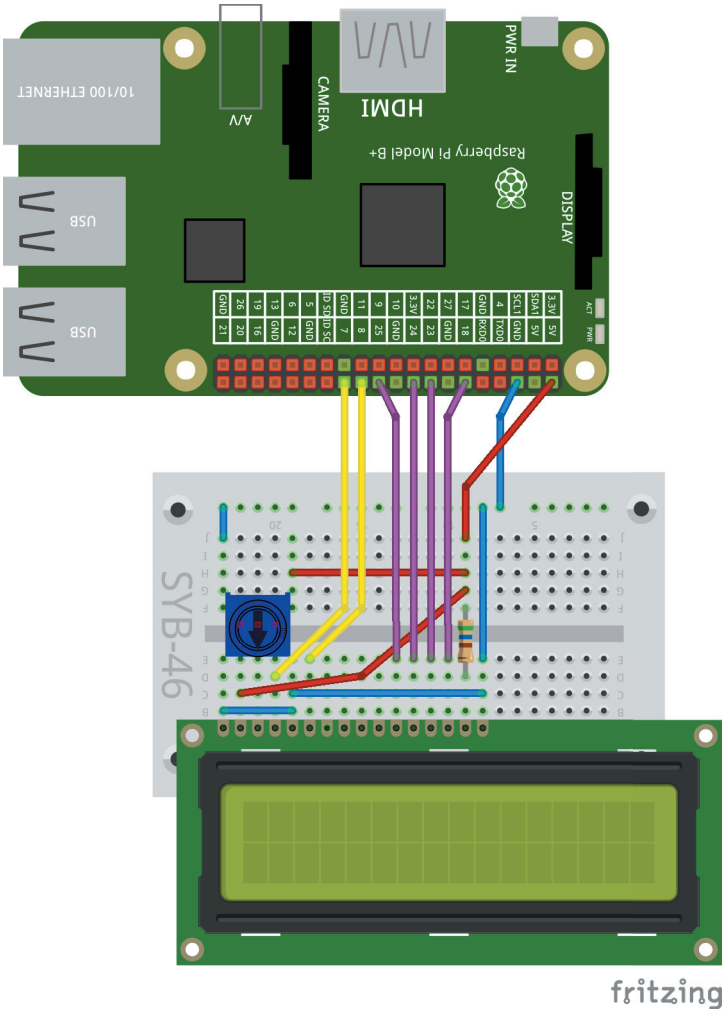
Audioeinstellungen in OSMC

20.2 | Display anschließen

Das LC-Display wird im 4-Bit-Modus am Raspberry Pi ohne Portexpander direkt angeschlossen. Wir verwenden die gleiche Belegung der GPIO-Ports wie in den Experimenten mit Python und Scratch.

Benötigte Bauteile

1x Steckplatine, 1x LC-Display, 1x 560-Ohm-Widerstand (Grün-Blau-Braun), 1x Potenziometer, 8x Verbindungskabel, 6x Drahtbrücke (unterschiedliche Längen)



Gelb: Steuerleitungen, violett: Datenleitungen, rot: +5 V, blau: Masse

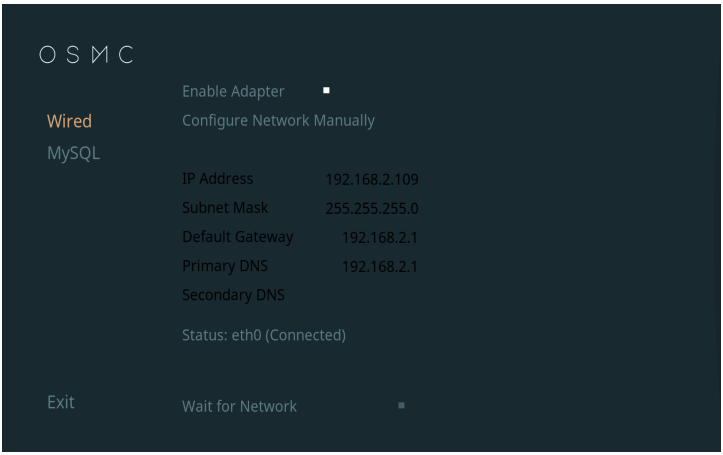
20.3 | LCDproc und Treiber konfigurieren

OSMC unterstützt über ein Addon die Software LCDproc, ein vielfältig verwendbares Programmpaket zur Steuerung diverser Displaymodule, das nicht nur auf dem Raspberry Pi, sondern auch für diverse andere Systeme angeboten wird.

20

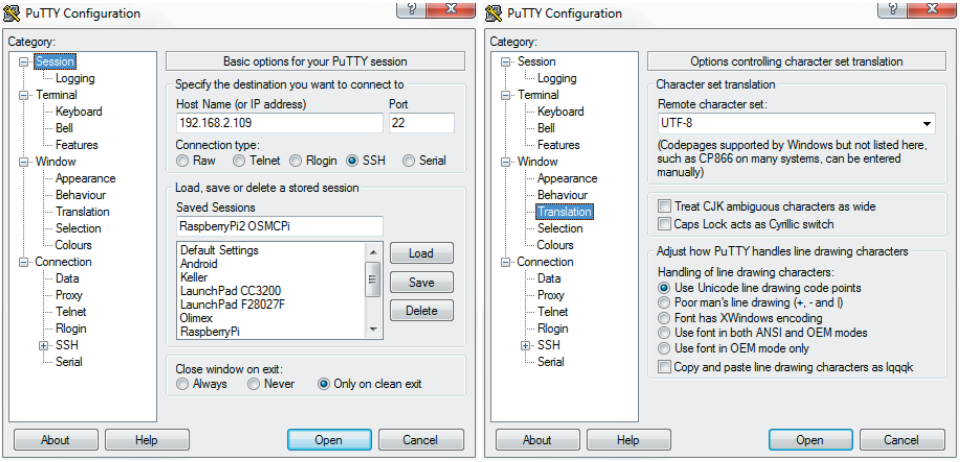
OSMC hat ähnlich wie Raspbian einen SSH-Server vorinstalliert. Auf diesem Weg können Sie von einem anderen PC im Netzwerk per Kommandozeile auf das unter OSMC liegende Linux-Betriebssystem zugreifen, Programme installieren und auch Dateien vom PC übertragen.

1. Installieren Sie auf dem PC das SSH-Terminal PuTTY von *www.putty.org*.
2. Lassen Sie sich in OSMC unter *My OSMC/Network* die aktuelle IP-Adresse des Raspberry Pi im lokalen Netzwerk anzeigen.

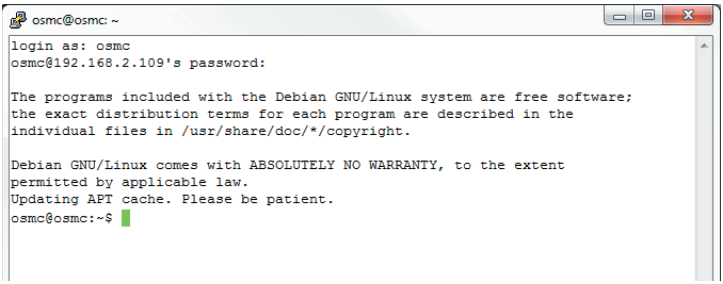


Anzeige der IP-Adresse in OSMC

3. Legen Sie in PuTTY eine neue Verbindung mit der IP-Adresse des Raspberry Pi über Port *22* mit dem Verbindungstyp *SSH* an.
4. Wählen Sie links im Bereich *Windows/Translation* in der Liste *Remote Character Set* die Option *UTF-8* aus.
5. Speichern Sie die Verbindung im Bereich *Session* mit einem Klick auf *Save* und vergeben Sie einen eindeutigen Namen.
6. Bei der ersten Verbindung erscheint eine Sicherheitswarnung, die aber nicht relevant ist, da es sich um keine Internetverbindung, sondern um eine Verbindung innerhalb des lokalen Netzwerks handelt.
7. Melden Sie sich jetzt im PuTTY-Terminal auf dem Raspberry Pi an. Der Benutzername lautet `osmc`, das Passwort ist ebenfalls `osmc`. Im PuTTY-Fenster können Sie wie in einem lokalen Terminalfenster auf dem Raspberry Pi Linux-Befehle ausführen.



SSH-Verbindung zum Raspberry Pi auf einem PC anlegen



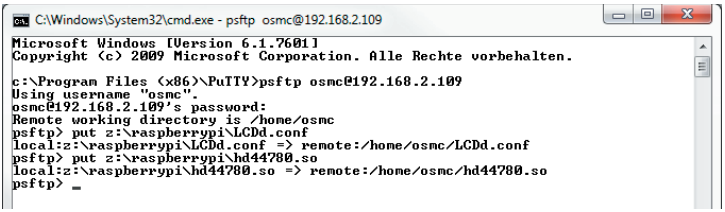
Anmeldung auf dem Raspberry Pi in PuTTY auf dem PC

8 Zur Konfiguration des LCD-Moduls benötigen wir noch zwei Dateien aus den Downloads zum Buch. Um diese auf den Raspberry Pi mit OSMC zu übertragen, verwenden Sie am einfachsten das Kommandozeilenprogramm `psftp` aus dem PuTTY-Paket. Wechseln Sie in einem Windows-Eingabeaufforderungsfenster in das Verzeichnis `c:\Program Files (x86)\PuTTY`, in dem PuTTY installiert ist, und starten Sie dort `psftp` mit der IP-Adresse des Raspberry Pi (ersetzen Sie die hier gezeigte IP-Adresse durch die Ihres Raspberry Pi):

```
psftp osmc@192.168.2.109
```

9 Als Erstes müssen Sie das Passwort `osmc` eingeben.

20



Dateien vom PC auf den Raspberry Pi übertragen

⑩ Kopieren Sie jetzt die beiden Dateien LCDd.conf und hd44780.so auf den Raspberry Pi (ersetzen Sie das hier gezeigte Verzeichnis `z:\raspberrypi` durch das Verzeichnis auf dem PC, wo Sie die Downloads entpackt haben):

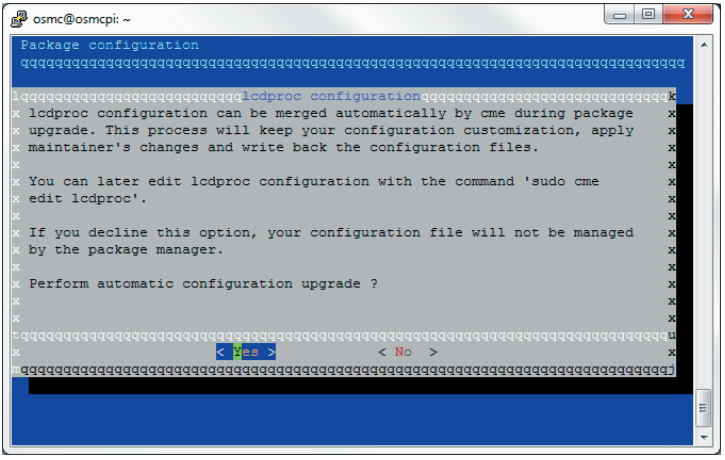
```
put z:\raspberrypi\LCDd.conf
put z:\raspberrypi\hd44780.so
```

⑪ Beenden Sie `psftp` mit dem Kommando `exit`.

⑫ Installieren Sie jetzt das Modul `lcdproc` auf dem Raspberry Pi. Geben Sie dazu im PuTTY-Terminal folgende Befehlsfolge ein:

```
sudo apt-get update
sudo apt-get install -y dpkg-dev gcc build-essential
sudo apt-get install -y lcdproc lcdproc-extra-drivers
```

⑬ Während der Installation von `lcdproc` erscheint einmal ein textbasierter Konfigurationsdialog. Bestätigen Sie hier die vorgegebene Option *Yes* mit der Enter-Taste.



Abfrage während der Installation von lcdproc

⓮ Kopieren Sie jetzt die beiden aus den Downloads vom PC übertragenen Dateien an die richtigen Positionen im Linux-Dateisystem:

```
sudo cp ~/hd44780.so /usr/lib/arm-linux-gnueabihf/lcdproc/hd44780.so
sudo cp ~/LCDd.conf /etc/LCDd.conf
```

⓯ Starten Sie anschließend das LCDproc-Modul:

```
sudo /etc/init.d/LCDd restart
```

20.3.1 | Die Konfigurationsdatei LCDd.conf

Programmdatei:
LCDd.conf

Damit Sie die Konfigurationsdatei `LCDd.conf` auf das verwendete Display und die GPIO-Anschlüsse nicht manuell anpassen müssen, liefern wir eine passende Datei mit. Die Originaldatei ist sehr lang und durch zahlreiche Kommentarzeilen auch alles andere als übersichtlich. Die in den Downloads enthaltene Konfigurationsdatei enthält dagegen nur die wirklich notwendigen Parameter und diese bereits mit den Werten, die OSMC zur Kommunikation mit dem LCDproc-Modul braucht.

```
001 [hd44780]
002 ConnectionType=raspberrypi
003 pin_D7=18
004 pin_D6=23
005 pin_D5=24
006 pin_D4=25
007 pin_RS=7
008 pin_EN=8
009 Size=16x2
010 CharMap = hd44780_euro
011 #RefreshDisplay = 1
012
013 [server]
014 Driver=hd44780
015 DriverPath=/usr/lib/arm-linux-gnueabihf/lcdproc/
016
017 Bind=127.0.0.1
018 Port=13666
019
020 ReportToSyslog=on
021 ReportLevel = 5
022 WaitTime=5
023 User=nobody
```

20

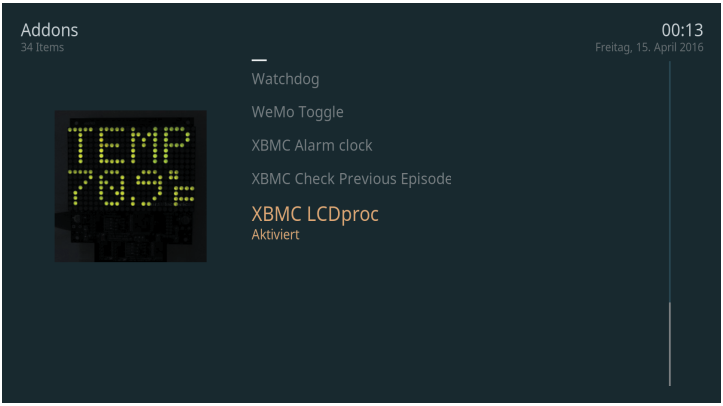
```
024 ServerScreen=on
025 Foreground=off
026 Heartbeat=open
027 InitialHeartbeat=on
```

Sollten Sie das LC-Display an anderen GPO-Pins angeschlossen haben, ändern Sie die Zeilen am Anfang der Datei so, dass die wirklich verwendeten GPIO-Pins eingetragen sind. Sollten auf dem LC-Display zwischendurch immer mal wieder wirre Zeichen anstatt eines erkennbaren Textes erscheinen, nehmen Sie das Kommentarzeichen vor dem Parameter `RefreshDisplay` heraus. Dann wird der Inhalt des Displays einmal pro Sekunde komplett aktualisiert, um solche Fehler zu vermeiden.

20.3.2 | Einrichtung in OSMC

OSMC benötigt ein Addon, um mit dem installierten Softwaremodul LCDproc zu kommunizieren.

1. Um das LCDproc-Addon in OSMC zu konfigurieren, wählen Sie dort *Einstellungen/Addons/Aus Repository installieren/Kodi Add-on repository*.
2. Jetzt erscheint ein umfangreicher Katalog mit Addons zu verschiedenen Themen. Wählen Sie hier das Addon *Dienste/XBMC LCDproc* aus und installieren Sie es.



Das Addon XBMC LCDproc im Addon-Repository

3. Die Installation des LCDproc-Addons erfolgt vollautomatisch. Danach erscheinen auf dem Display auch sofort aktuelle Textmeldungen. Spielen Sie einen Musiktitel ab, sehen Sie den Titelnamen und den Interpreten auf dem LC-Display.

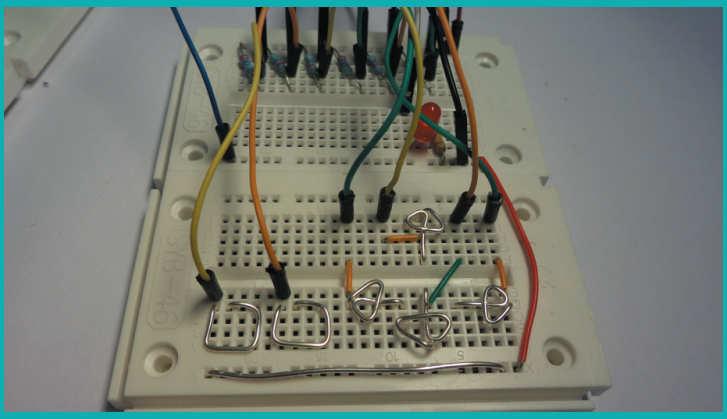
Der Info- und Konfigurationsbildschirm des Addons XBMC LCDproc

4 Über den Menüpunkt *Konfigurieren* in der unteren Menüleiste kommen Sie zum Konfigurationsdialog des XBMC LCDproc Addons. Hier lassen sich verschiedene Anzeigeparameter, wie die Geschwindigkeit der Laufschrift oder die Anzeigedauer der Navigationsansicht, einstellen. Schalten Sie auf der Seite *Verbindung* den Schalter *Verbindungsfehler-Benachrichtigung verstecken* aus, damit Sie sehen, wenn die Verbindung zwischen OSMC und LCDproc nicht funktioniert. In diesem Fall hilft es meistens, das Addon einmal zu deaktivieren und wieder zu aktivieren. Sollte dies keinen Erfolg bringen, starten Sie den Raspberry Pi neu, wobei das LCDproc-Modul und auch das Display selbst automatisch neu initialisiert werden.

»Verbindungsfehler-Benachrichtigung verstecken« ausschalten, um Fehler zu sehen

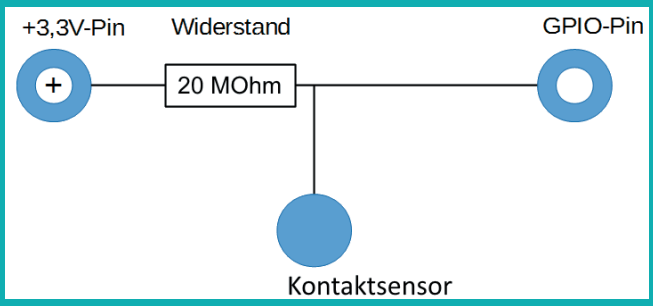
21 PIKEY PIKEY – SPIELE MIT DEN FINGERN STEUERN

Ein Spiel einfach nur durch Berührung selbst gebauter Sensortasten steuern … – mit unserem letzten Projekt „PiKey PiKey“ ist das möglich.



Selbst gebaute Sensortasten zur Steuerung eines Spiels

Das Prinzip der Sensortasten ist einfach: Die verwendeten GPIO-Ports sind über extrem hochohmige Widerstände (20 MOhm) mit +3,3 V verbunden, sodass ein schwaches, aber eindeutig als `GPIO.High` definiertes Signal anliegt. Ein Mensch, der nicht gerade frei in der Luft schwebt, ist immer geerdet und liefert über die elektrisch leitfähige Haut einen `GPIO.Low`-Pegel. Berührt dieser Mensch jetzt einen Sensorkontakt, wird das schwache `GPIO.High`-Signal von dem deutlich stärkeren `GPIO.Low`-Pegel der Fingerkuppe überlagert und zieht den entsprechenden GPIO-Pin auf `GPIO.Low`-Pegel. Damit dies funktioniert, müssen die internen Pulldown-Widerstände des Raspberry Pi abgeschaltet sein.



Schaltschema der PiKey-PiKey-Kontaktsensoren

Bauen Sie die abgebildete Schaltung auf zwei Steckplatinen auf. Dazu brauchen Sie außer dem isolierten Schaltdraht noch blanken Draht, im Bild grau dargestellt. Entfernen Sie dazu mit einem scharfen Messer die Isolierung von einem Stück Schaltdraht komplett oder verwenden Sie einfachen Kupfer- oder Silberdraht, der sich in jedem Bastlerhaushalt finden lässt. Biegen Sie aus dem blanken Draht die Kontaktsensoren für vier Pfeiltasten und zwei Buttons. Der lange Draht ganz unten quer ist mit 0 V Masse verbunden. Durch Berühren dieses Drahtes sollten Sie Ihre Hand vor Benutzung der Kontaktsensoren erden, um zu verhindern, dass durch statische Elektrizität Überspannungen auf die GPIO-Pins gelangen. Aus dem gleichen Grund sollten Sie die GPIO-Pins bei eingeschaltetem Raspberry Pi nicht unnötig berühren.

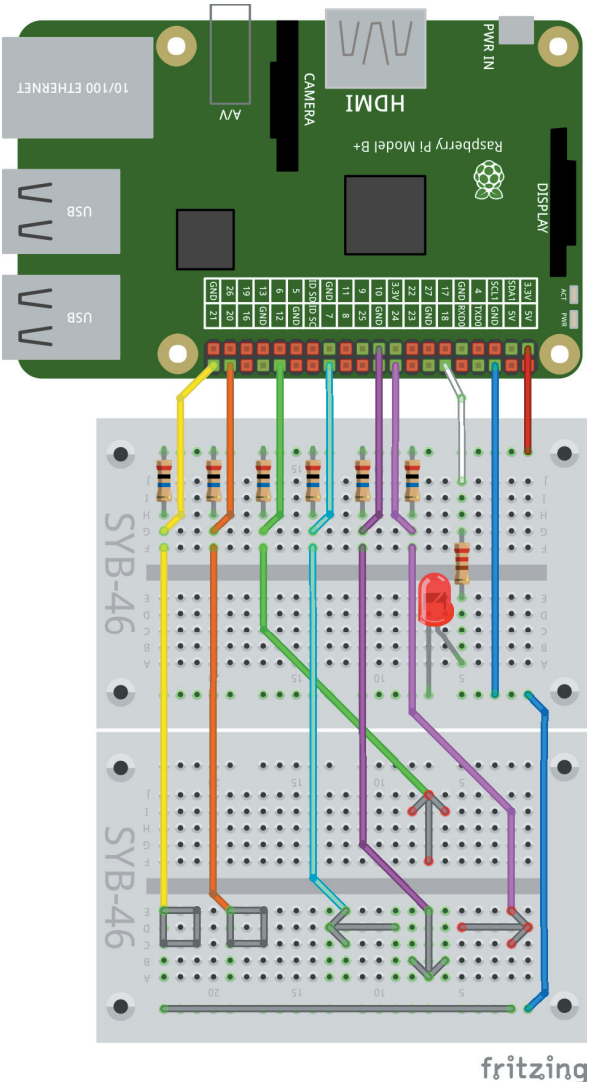
Benötigte Bauteile

2x Steckplatine, 1x LED rot, 1x 220-Ohm-Widerstand (Rot-Rot-Braun), 6x 20-MOhm-Widerstand (Rot-Schwarz-Blau), 9x Verbindungskabel, 7x Drahtbrücke (unterschiedliche Längen), 7x Draht blank (unterschiedliche Längen)

Zunächst müssen einige Module nachinstalliert werden, die nicht im Standard-Lieferumfang von Raspbian enthalten sind. Rufen Sie dazu folgende Linux-Kommandos in einem LXTerminal auf:

```
sudo apt-get update
sudo apt-get install python-setuptools python-xlib
sudo git clone https://github.com/SavinaRoja/PyUserInput.git
cd PyUserInput
sudo python setup.py install
```

21



PiKey PiKey mit Kontaktsensoren (grau)

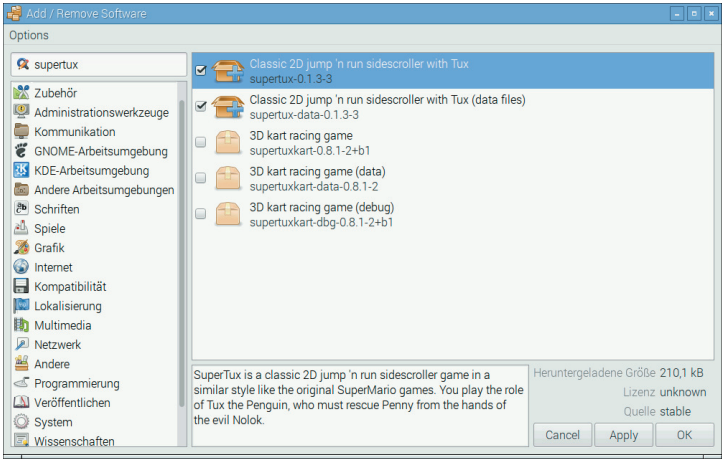
Programm-datei:
pikeypikey.py

Jetzt können Sie das Programm `pikeypikey.py` aufrufen. Es läuft nur im Hintergrund und tut auf den ersten Blick gar nichts. Mit Hilfe des Python-Moduls *PyKeyboard* werden am GPIO-Port Tastatureingaben simuliert, mit denen sich Spiele und beliebige andere Programme steuern lassen.

Supertux (*supertux.lethargik.org*) ist ein beliebtes Jump-and-Run-Spiel für Linux, das sich gut mit den Cursortasten sowie der Leertaste steuern lässt. Das Programm `pikeypikey.py` simuliert diese Tasten über die Kontaktsensoren.

Das Jump-and-Run-Spiel Supertux

Installieren Sie das Spiel über den Menüpunkt *Einstellungen/Add/Remove Software*. Suchen Sie hier nach `supertux`. Installieren Sie die beiden Pakete *supertux* und *supertux-data*. Vor der Installation müssen Sie einmal das Passwort `raspberry` eingeben.



Supertux über die Paketverwaltung von Raspbian installieren

Starten Sie jetzt das Python-Programm `pikeypikey.py` über die Python-IDE und dann das Spiel aus dem Menü *Spiele*. Anstatt mit Tastatur und Maus können Sie das Spiel jetzt über die PiKey-PiKey-Kontaktsensoren steuern.

21

21.1 | So funktioniert das Programm

Das Programm ähnelt den bisherigen Experimenten mit der GPIO-Schnittstelle, enthält aber wesentliche Neuerungen, besonders aufgrund des `PyKeyboard`-Moduls.

```
001 #!/usr/bin/python
002 import RPi.GPIO as GPIO
003 from pykeyboard import PyKeyboard
004 import time
```

Dieses Modul wird am Anfang mit den Modulen `RPi.GPIO` für die GPIO-Schnittstelle sowie `time` zur Steuerung von Wartezeiten importiert.

```
006 GPIO.setmode(GPIO.BCM)
007
008 K1 = 21
009 K2 = 20
010 K3 = 10
011 K4 = 24
012 K5 = 7
013 K6 = 12
014 L1 = 18
```

Die GPIO-Adressierung wird wie üblich auf den BCM-Modus gesetzt. Danach werden die sechs für die Tasten verwendeten GPIO-Portnummern in den Konstanten `K1` bis `K6` definiert und der GPIO-Port für die LED in der Konstanten `L1`.

```
016 gpio_in = [K1, K2, K3, K4, K5, K6]
```

Das Array `gpio_in[]` enthält die zuvor definierten GPIO-Portnummern der sechs Eingänge zur Verwendung in Schleifen, wo sich Arrays deutlich einfacher verarbeiten lassen, als einzelne Variablen.

```
017 gpio_old = {K1:0, K2:0, K3:0, K4:0, K5:0, K6:0}
```

Zum Speichern der vorherigen Werte der GPIO-Eingänge wird eine Variable `gpio_old{}` vom Typ *dictionary*, eine besondere Form der Liste, verwendet. In einem Dictionary werden die einzelnen Elemente nicht über ihre Nummer ausgewählt, sondern über ein beliebiges Wort, den sogenannten Schlüssel, der in diesem Fall die Konstante mit der GPIO-Portnummer enthält. Im Gegensatz zu einer einfachen Liste steht ein Dictionary in geschweiften Klammern. Es kann beliebig viele Paare aus Schlüssel und Wert enthalten. Am Anfang werden alle diese Werte auf `0` gesetzt.

```
019 k = PyKeyboard()
```

Der Variablen `k` wird eine Datenstruktur zugewiesen, die alle Eingaben der simulierten Tastatur `Pykeyboard()` enthält. Die verschiedenen Elemente dieser Struktur repräsentieren später gedrückte Tasten.

```
020 pause = 0.01
```

Um Tastenprellen zu verhindern, wird eine Pause von 0,01 Sekunden definiert.

```
022 for gpio in gpio_in:
      GPIO.setup(gpio, GPIO.IN)
```

Alle Ports aus dem Array `gpio_in[]` werden als Eingänge ohne Pulldown-Widerstände definiert.

```
025 GPIO.setup(L1, GPIO.OUT, initial=0)
```

Der GPIO-Port für die LED wird als Ausgang definiert und ausgeschaltet.

```
027 while True:
      time.sleep(pause)
```

Jetzt startet eine Endlosschleife, die in jedem Durchlauf mit einer kurzen Pause beginnt, um Tastenprellen zu verhindern.

```
030   if GPIO.input(K1) == 0 and gpio_old[K1] == 1:
031     k.press_key(k.enter_key)
032     gpio_old[K1] = 0
```

In jedem Durchlauf werden nacheinander die sechs Kontaktsensoren abgefragt. Liegt am GPIO-Eingang `K1` das Signal `0` an, berührt der Benutzer gerade den Kontakt. Ist gleichzeitig der im zugehörigen Feld aus `gpio_old{}` gespeicherte Wert 1, hat der Benutzer den Kontakt gerade erst berührt. Dieser Zustand wird über die Methode `k.press_key()` der Datenstruktur `k` als Tastendruck ausgewertet und der Wert `k.enter_key`, in diesem Fall die Enter-Taste, an das Betriebssystem bzw. ein laufendes Programm weitergegeben. Danach wird der Wert `gpio_old[]` für diesen Kontaktsensor auf `0` gesetzt, was bedeutet, dass der Sensor berührt wurde.

Solange der Sensor weiterhin berührt wird, reagiert diese Abfrage nicht mehr. Es werden keine weiteren Tastendrücke übermittelt.

```
033   elif GPIO.input(K1) == 1 and gpio_old[K1] == 0:
034     k.release_key(k.enter_key)
035     gpio_old[K1] = 1
```

21

Umgekehrt wird der Zustand, in dem der GPIO-Eingang den Wert `1` annimmt, also nicht mehr berührt wird, dann als Loslassen der Taste interpretiert, wenn er zuvor noch berührt wurde, das zugehörige Feld aus `gpio_old{}` den Wert `0` hat. In diesem Fall wird dem Betriebssystem über die Methode `k.release_key()` übermittelt, dass die Enter-Taste losgelassen wurde.

Die gleiche Abfrage wird für die anderen fünf Kontaktsensoren ausgeführt. PyKeyboard kann dabei alphanumerische Tasten direkt als Zeichenkette verwerten, wie z. B. die Leertaste `' '` für den GPIO-Port `K2`. Steuerzeichen werden durch Schlüsselwörter repräsentiert. Die Tabelle zeigt die im Programm verwendeten Tasten.

Variable im Programm	Taste	Code
K1	Enter	enter_key
K2	Leertaste	' '
K3	↓	down_key
K4	→	right_key
K5	←	left_key
K6	↑	up_key

Nachdem alle sechs Kontaktsensoren abgefragt wurden, soll die LED aufleuchten, wenn ein Kontakt berührt wurde.

```
072   LED = 0
073   for gpio in gpio_in:
074     LED = LED or (not GPIO.input(gpio))
075   GPIO.output(L1, LED)
```

Zunächst wird die Variable `LED` auf `0` gesetzt. Nur wenn einer der Kontakte berührt wurde, soll die LED eingeschaltet werden. Dazu fragt eine Schleife alle GPIO-Ports ab. Liefert ein GPIO-Port nicht den Wert `1`, wird die Variable `LED` auf `1` gesetzt. Die `or`-Verknüpfung verhindert, dass die LED sofort wieder auf `0` gesetzt wird, wenn der nächste abgefragte GPIO-Port den Wert `1` liefert, also der Kontaktsensor nicht berührt wird und somit über den Widerstand mit +3,3 V verbunden ist und den Pegel `GPIO.High` liefert.

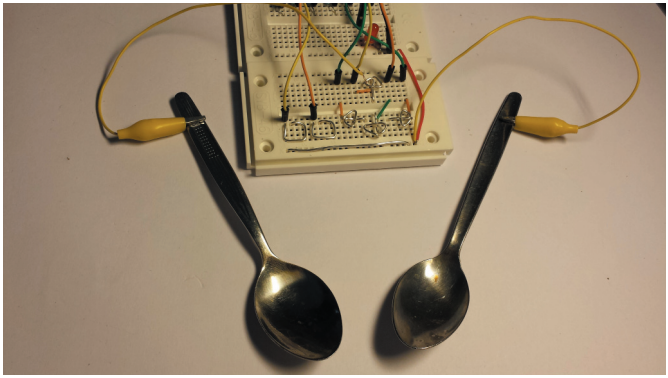
Am Ende der Schleife wird der als `L1` definierte GPIO-Ausgang auf den Wert der Variablen `LED` gesetzt. Bei `1` leuchtet die LED, bei `0` nicht. Danach

beginnt die Hauptschleife von neuem, die so schnell durchläuft und am Ende die LED wieder auf `0` setzt, dass das Einschalten nur als kurzes Aufblinken wahrgenommen wird.

21.2 | Kaffeelöffel als Boss-Taste und Erdung

Nicht immer reicht die Erdung über den Fußboden aus, um PiKey PiKey zu nutzen. Teppich oder Kunststofffußboden und auch manche Schuhsohlen isolieren so gut, dass man ohne Berühren des Massedrahtes keinen stabilen `GPIO.Low`-Pegel an den Fingern hat. Barfuss auf Steinfußboden gibt es keine Probleme.

Mit den Krokodilklemmenkabeln und Kaffeelöffeln oder anderen leitfähigen Gegenständen, die auf dem Tisch herumliegen, erweitern Sie das Spiel um eine stabile Erdung sowie eine Boss-Taste, mit der es sich schnell beenden lässt, wenn der Chef hereinkommt. Erden Sie sich einfach, indem Sie einen Kaffeelöffel in die eine Hand nehmen, mit der anderen Hand spielen Sie wie gewohnt. Kommt der Chef, berühren Sie ein paar Mal kurz hintereinander den anderen Kaffeelöffel, bis das Spiel beendet ist.



Zwei Kaffeelöffel als Boss-Taste und Erdung

Schließen Sie ein Krokodilklemmenkabel an der Masseleitung unten an, das andere an einem Pin, der mit dem Kontaktsensor *Pfeil nach oben* verbunden ist, und klemmen Sie zwei Löffel an die Krokodilklemmen.

Im Spiel Supertux können Sie über den Menüpunkt *Options/Keyboard Setup* festlegen, welche Tasten zur Steuerung verwendet werden. Diese Belegung brauchen Sie nicht zu verändern.

21

Die Tastenbelegung in Supertux

Hier sieht man deutlich: Zum tatsächlichen Spielen werden nur fünf Tasten benötigt. Der Pfeil nach oben wird nur im Spielmenü verwendet, ist also entbehrlich. Drückt man nur oft genug auf den Pfeil nach unten, fängt das Menü von oben wieder an, sodass jeder Menüpunkt auf diese Weise erreichbar ist. Die Taste Esc zum Beenden des Spiels lässt sich nicht umdefinieren. Damit der Pfeil nach oben geht und der angeklemmte Kaffeelöffel diese Funktion übernimmt, ändern Sie das Python-Programm.

Die letzte Abfrage in der Endlosschleife ist für die Pfeiltaste nach oben verantwortlich. Ersetzen Sie hier zweimal den Parameter `k.up_key` durch `k.escape_key`.

Programmdatei:
pikeypikey_esc.py

```
065   if GPIO.input(K6) == 0 and gpio_old[K6] == 1:
066     k.press_key(k.escape_key)
067     gpio_old[K6] = 0
068   elif GPIO.input(K6) == 1 and gpio_old[K6] == 0:
069     k.release_key(k.escape_key)
070     gpio_old[K6] = 1
```

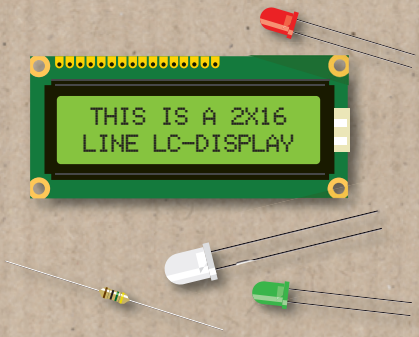
Diese kleine Änderung macht aus der Pfeiltaste eine Boss-Taste.

Platz für Ideen

Dieses Beispiel stellt nur eine von vielen Möglichkeiten dar, PiKey PiKey als Eingabemethode zu verwenden. Über die im MakerKit enthaltenen Krokodilklemmenkabel können Sie die unterschiedlichsten leitfähigen Gegenstände als Kontaktsensoren anschließen und sind damit nicht mehr auf den Aufbau auf einer Steckplatine beschränkt.

TURN ON YOUR CREATIVITY

FRANZIS
RASPBERRY PI
MAKER KIT



Sie haben bereits einen Raspberry Pi im Einsatz und vielleicht auch schon ein erstes Elektronik-Projekt umgesetzt? Dann sind Sie hier richtig: Dieses Lernpaket bietet alles, um mehr als nur einfache Projekte umzusetzen: 62 Bauteile sowie ein 160-seitiges Handbuch. Nachdem Sie die 20 detailliert beschriebenen Projekte umgesetzt haben, sind Sie bereit für die Welt der Maker.

DIESE PROJEKTE FÜHREN SIE DURCH:

- LED mit Python ein-/ausschalten
- Erste Projekte mit Scratch
- GPIO mit Scratch ansteuern
- Fußgängerampel mit Python und mit Scratch
- Spielwürfel mit LEDs
- Scratch-Katze mit GPIO-Tasten steuern
- Digitaluhr mit Scratch auf dem LC-Display
- LC-Display mit Python ansteuern
- LC-Display im 8-Bit-Modus
- IP-Adresse des Raspberry Pi anzeigen
- Laufschrift auf dem LC-Display
- Erweiterte Statusanzeige
- Interaktive Statusanzeige mit Tasten
- Lauflicht mit dem Portexpander
- Binäruhr
- CPU-Lastanzeige mit LEDs am Port-Expander
- LC-Display am Port-Expander
- LC-Display für RaspBMC Media Center
- PiKey PiKey – Spiele mit den Fingern steuern

Nicht enthalten: Raspberry Pi

Für Kinder unter 14 Jahren nicht geeignet!

ISBN 978-3-645-65269-8

FRANZIS