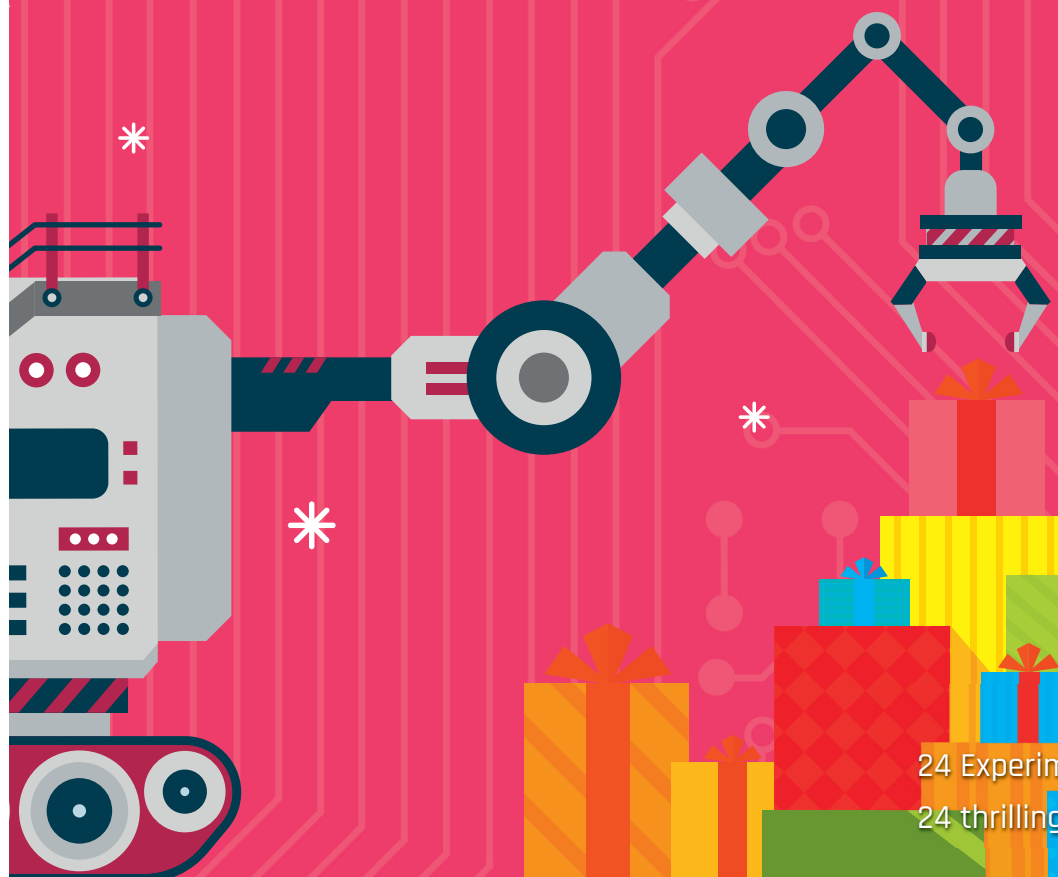
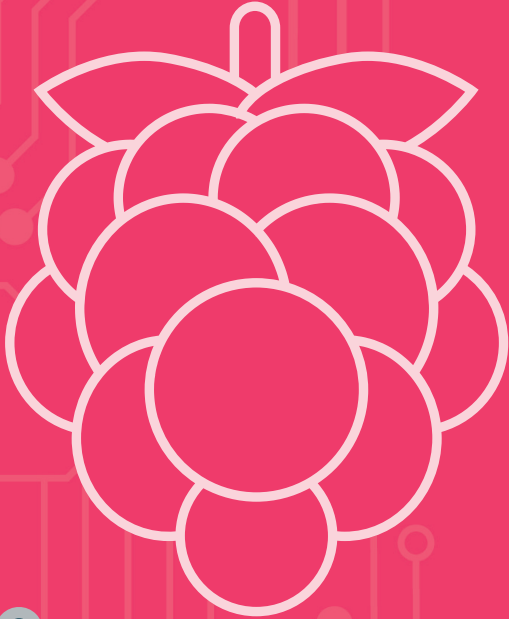




ADVENTSKALENDER FÜR RASPBERRY PI

COMPATIBLE WITH
RASPBERRY
PI

ADVENT CALENDAR FOR RASPBERRY PI



24 Experimente, die Spaß machen
24 thrilling experiments

Alle Versuche im Überblick

Raspberry-Pi-Adventskalender 2019	3	13. Tag	34
1. Tag	4	Heute im Adventskalender	34
Heute im Adventskalender	4	Taster an einem GPIO-Pin anschließen	34
Den Raspberry Pi vorbereiten	4	Weihnachts-Countdown mit Taster	34
Betriebssysteminstallation in Kürze	4	Das Programm	34
Die wichtigsten Bauteile kurz erklärt	5	So funktioniert es	36
LED leuchtet	8	14. Tag	37
2. Tag	9	Heute im Adventskalender	37
Heute im Adventskalender	9	Weihnachts-Countdown mit zwei Tastern	37
Zwei LEDs blinken abwechselnd	9	Das Programm	37
Grundeinstellungen für Scratch	9	So funktioniert es	39
Das Programm	10	15. Tag	40
3. Tag	12	Heute im Adventskalender	40
Heute im Adventskalender	12	Zahlen mit Scratch auf der Sieben-Segment-Anzeige	40
Ampelschaltung mit Python	12	Das Programm	40
Das Programm	13	16. Tag	42
So funktioniert es	14	Heute im Adventskalender	42
4. Tag	16	Digitaluhr	42
Heute im Adventskalender	16	Das Programm	42
Drei Segmente der Sieben-Segment-Anzeige blinken abwechselnd	16	So funktioniert es	43
Anschlusschema der Sieben-Segment-Anzeige	16	Digitaluhr automatisch starten	44
Das Programm	17	17. Tag	45
So funktioniert es	17	Heute im Adventskalender	45
5. Tag	18	IP-Adresse des Raspberry Pi anzeigen	45
Heute im Adventskalender	18	Das Programm	45
Lauflicht auf der Sieben-Segment-Anzeige	18	So funktioniert es	46
Das Programm	18	18. Tag	48
So funktioniert es	18	Heute im Adventskalender	48
6. Tag	19	Stoppuhr	48
Heute im Adventskalender	19	Das Programm	48
Noch ein Lauflicht auf der Sieben-Segment-Anzeige	19	So funktioniert es	49
Das Programm	19	19. Tag	51
So funktioniert es	20	Heute im Adventskalender	51
7. Tag	21	Sensorkontakt aus Knete	51
Heute im Adventskalender	21	Stoppuhr mit Sensorkontakt	51
Schaltdraht	21	Das Programm	52
Vier Ziffern zeigen das gleiche Blinkmuster	21	So funktioniert es	53
Das Programm	22	20. Tag	54
So funktioniert es	22	Heute im Adventskalender	54
8. Tag	23	Zähler mit Sensorkontakten	54
Heute im Adventskalender	23	Das Programm	54
Sieben-Segment-Anzeige mit Scratch steuern	23	So funktioniert es	55
Das Programm	23	Der Zähler zählt zu schnell	56
9. Tag	26	21. Tag	57
Heute im Adventskalender	26	Heute im Adventskalender	57
Zahlen auf der Sieben-Segment-Anzeige	26	Der Raspberry Pi erzeugt Töne	57
Das Programm	26	Das Programm	57
So funktioniert es	27	22. Tag	60
10. Tag	28	Heute im Adventskalender	60
Heute im Adventskalender	28	Ampel mit Countdown	60
Mehrstellige Zahlen auf der Sieben-Segment-Anzeige	28	Das Programm	60
Der Trick mit dem Nachleuchten	28	23. Tag	64
Das Programm	28	Heute im Adventskalender	64
So funktioniert es	29	Zahlenraten mit drei Tasten	64
11. Tag	30	Das Programm	64
Heute im Adventskalender	30	So funktioniert es	65
Beliebige Zahlen anzeigen	30	24. Tag	68
Das Programm	30	Heute im Adventskalender	68
So funktioniert es	31	Weihnachtslieder auf dem Raspberry Pi	68
12. Tag	33	Das Programm	68
Heute im Adventskalender	33	So funktioniert es	70
Countdown bis Weihnachten	33		
Das Programm	33		
So funktioniert es	33		

Raspberry-Pi-Adventskalender 2019

Dieser Adventskalender enthält für jeden Tag ein Hardwareexperiment mit dem Raspberry Pi. Die Experimente werden mit Scratch und Python programmiert. Beide Programmiersprachen sind auf dem Raspberry Pi vorinstalliert. Alle Experimente funktionieren mit dem Raspberry Pi 3, Raspberry Pi 3 B+ und Raspberry Pi 4 mit der aktuellen Version des Raspbian Betriebssystems.

Mit einem normalen PC oder gar einem Notebook einfache Elektronik zu steuern ist - auch wenn es nur ein paar LEDs sind - für Hobbyprogrammierer mit kaum vertretbarem Aufwand verbunden. Dem PC fehlen einfach die dafür notwendigen Schnittstellen. Außerdem ist das Windows-Betriebssystem denkbar ungeeignet dafür, mit Elektronik zu kommunizieren.

Der Raspberry Pi ist - obwohl es auf den ersten Blick gar nicht so aussieht - ein vollwertiger Computer. Vieles geht etwas langsamer, als man es von modernen PCs gewohnt ist, dafür ist der Raspberry Pi aber auch viel kleiner und vor allem billiger als ein PC.

Programme zum Download

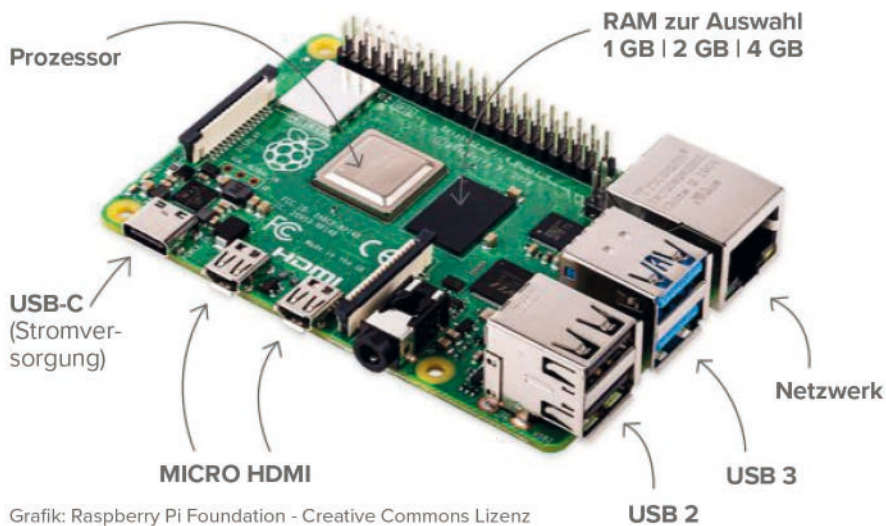
Die im Adventskalender verwendeten Programme gibt es hier zum Download: www.buch.cd. Tragen Sie für dieses Produkt den Code 15045-5 in das Eingabefeld ein.

1. Tag

1. Tag

Heute im Adventskalender

- 1 Steckbrett
- 1 LED rot
- 1 Widerstand 220 Ohm
- 2 GPIO-Verbindungskabel



Grafik: Raspberry Pi Foundation - Creative Commons Lizenz

Die Anschlüsse am Raspberry Pi (Grafik: Raspberry Pi Foundation - Creative Commons Lizenz)

Den Raspberry Pi vorbereiten

Achtung!

Dieser Adventskalender basiert auf dem Raspberry Pi 3 bzw. Raspberry Pi 3+ und der bei Druckdatum aktuellen Software. Neuerungen bei Software oder Hardware ergeben sich aber bisweilen kurzfristig. Damit Sie auf jeden Fall Spaß mit Ihrem Adventskalender haben, finden Sie im Download-Paket unter www.buch.cd hierzu aktuelle Hinweise.

Um den Raspberry Pi in Betrieb zu nehmen, braucht man:

- USB-Tastatur und Maus
- HDMI-Kabel für Monitor
- Netzwerkkabel oder WLAN
- MicroSD-Karte mit Betriebssystem Raspbian
- Micro-USB-Handyladegerät als Netzteil (mindestens 1.500 mA)
- Audiokabel für Lautsprecher (optional)

Das Netzteil muss als Letztes angeschlossen werden, damit schaltet sich der Raspberry Pi automatisch ein. Es gibt keinen eigenen Ein-/Ausschalter.

Der Raspberry Pi 4

Im Juni 2019 erschien der Raspberry Pi 4 mit deutlich leistungsfähigerer Hardware. Erstmals gibt es den Raspberry Pi mit drei verschiedenen Arbeitsspeichergrößen: 1 GB, 2 GB und 4 GB. Weitere Merkmale zusammengefasst sind:

- Prozessor: Broadcom BCM2711, Quad core Cortex-A72 64-bit SoC 1.5 GHz
- Netzwerk: Gigabit Ethernet und Dual-Band IEEE 802.11ac WLAN
- USB-Anschlüsse: 2x USB 3.0, 2x USB 2.0 (an der Farbe unterscheidbar)
- HDMI-Anschlüsse: 2x Micro-HDMI. Erstmals werden damit zwei Monitore unterstützt. Zum Anschluss sind Micro-HDMI-Kabel oder Adapter erforderlich.
- Stromversorgung : USB Typ C-Anschluss, mindestens 3 A erforderlich.

Der Steckplatz für MicroSD-Karten, die 40-polige GPIO-Pinleiste sowie die Klinkenbuchse für Stereo-Audio und Composite Video sind gegenüber dem Vorgängermodell unverändert.

Verwenden Sie für Tastatur und Maus die USB 2.0 Anschlüsse. Die USB 3.0 Anschlüsse verwenden Sie besser für USB-Sticks und externe Festplatten.

Betriebssysteminstallation in Kürze

Für alle, die ihren Raspberry Pi noch nicht mit der aktuellen Raspbian-Version betriebsbereit haben, folgt hier die Systeminstallation in zehn Schritten:

1. NOOBS (mindestens Version 3.1.1) von www.raspberrypi.org/downloads auf den PC herunterladen und Zip-Archiv auf die Festplatte entpacken.

2. Wurde die SD-Karte bereits benutzt, mit SD-Formatter im PC neu formatieren: www.sdcard.org/downloads/formatter_4. Dabei **Format Size Adjustment** einschalten (die SD-Karte muss mindestens 8 GB groß sein).
3. Alle Dateien und Unterverzeichnisse von NOOBS auf die SD-Karte kopieren.
4. SD-Karte aus dem PC nehmen, in den Raspberry Pi stecken und booten. Ganz unten **Deutsch** als Installationsssprache wählen. Damit wird automatisch auch die deutsche Tastatur ausgewählt.
5. Das Häkchen beim vorausgewählten Raspbian-Betriebssystem setzen und oben links auf **Install** klicken. Nach Bestätigung einer Sicherheitsabfrage, dass die Speicherkarte komplett überschrieben wird, startet die Installation, die einige Minuten dauert.
6. Nach abgeschlossener Installation bootet der Raspberry Pi neu.
7. Auf dem Raspbian-Desktop startet der Konfigurationsassistent und zeigt die IP-Adresse des Raspberry Pi. Hier auf **Next** klicken und Sprache und Zeitzone auswählen, falls sie nicht automatisch auf Deutsch gesetzt sind.
8. Im nächsten Schritt wird empfohlen, das Standardpasswort zu ändern, was für die Experimente in diesem Lernpaket nicht unbedingt nötig ist. Der Standardbenutzer **pi** wird beim Booten automatisch angemeldet, sodass Sie das Passwort nur selten brauchen werden.
9. Bei der WLAN-Verbindung das gewünschte Netzwerk auswählen und das Passwort eingeben, bei Ethernetverbindung einfach auf **Skip** klicken.
10. Zum Schluss automatisch Updates herunterladen und den Raspberry Pi neu starten.

Num-Lock-Taste

Wie bei fast allen Linux-Systemen ist auch beim Start von Raspbian der Ziffernblock standardmäßig ausgeschaltet. Drücken Sie die Num-Lock-Taste auf der Tastatur, um ihn einzuschalten.

Programme zum Download

Die im Adventskalender verwendeten Programme gibt es hier zum Download: www.buch.cd. Tragen Sie für dieses Produkt den Code 15045-5 in das Eingabefeld ein.



Öffnen Sie die Webseite direkt mit dem vorinstallierten Browser auf dem Raspberry Pi und laden Sie die Zip-Datei in das Home-Verzeichnis `/home/pi` herunter.



Starten Sie den Dateimanager auf dem Raspberry Pi. Dieser zeigt beim Start automatisch das Home-Verzeichnis an. Klicken Sie mit der rechten Maustaste auf die heruntergeladene Zip-Datei und wählen Sie im Kontextmenü *Hier entpacken*.

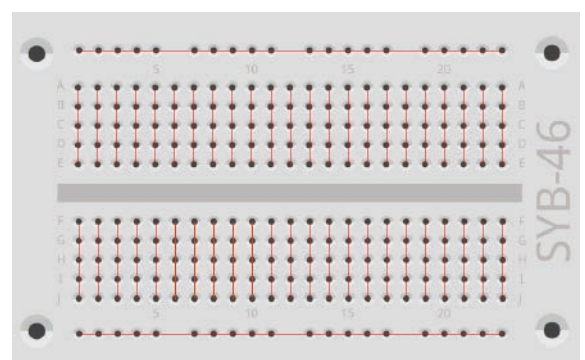
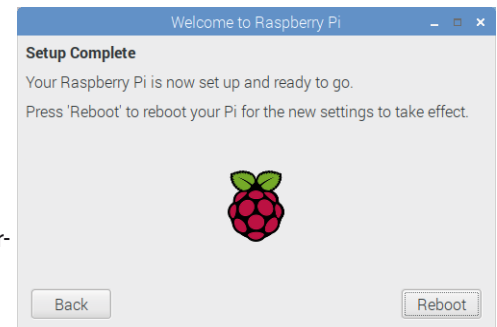
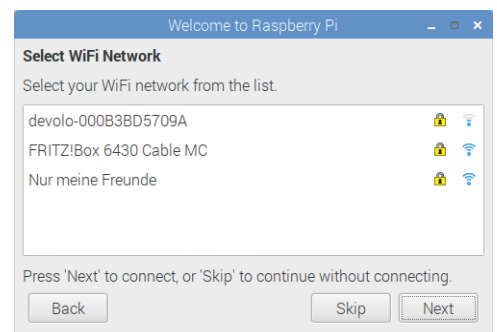
Das Downloadarchiv zum Adventskalender enthält dieses Handbuch als PDF in Farbe, damit Sie auf den Schaltplänen die einzelnen Leitungen besser erkennen können.

Die wichtigsten Bauteile kurz erklärt

Steckbrett

Für den schnellen Aufbau elektronischer Schaltungen, ohne löten zu müssen, enthält der Adventskalender ein Steckbrett. Hier können elektronische Bauteile direkt in ein Lochraster gesteckt werden.

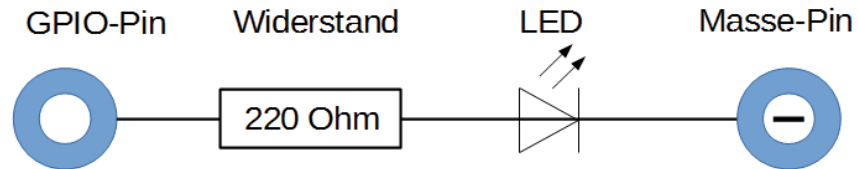
Bei diesem Steckbrett sind alle äußeren Längsreihen über Kontakte (X und Y) miteinander verbunden. Diese Kontaktreihen werden oft als Plus- und Minuspol zur Stromversorgung der Schaltungen genutzt. In den anderen Kontaktreihen sind jeweils fünf Kontakte (A bis E und F bis J) quer miteinander verbunden, wobei in der Mitte der Platine eine Lücke ist. Dort können größere Bauelemente eingesteckt und nach außen hin verdrahtet werden.



Die Verbindungen auf dem Steckbrett.

LEDs

LEDs (Deutsch: Leuchtdioden) leuchten, wenn Strom in Durchflussrichtung durch sie fließt. LEDs werden in Schaltungen mit einem pfeilförmigen Dreieckssymbol dargestellt, das die Flussrichtung vom Pluspol zum Minuspol oder zur Masseleitung angibt. Eine LED lässt in Durchflussrichtung nahezu beliebig viel Strom durch, sie hat nur einen sehr geringen Widerstand. Um den Durchflussstrom zu begrenzen und damit ein Durchbrennen der LED zu verhindern, wird üblicherweise zwischen dem verwendeten GPIO-Pin und der Anode der LED oder zwischen Kathode und Massepin ein 220-Ohm-Vorwiderstand eingebaut. Dieser Vorwiderstand schützt auch den GPIO-Ausgang des Raspberry Pi vor zu hohen Stromstärken.



Schaltplan einer LED mit Vorwiderstand.

LED in welcher Richtung anschließen?

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

Widerstand

Widerstände werden zur Strombegrenzung an empfindlichen elektronischen Bauteilen 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. Oft wird für die Einheit Ohm auch das Omega-Zeichen Ω verwendet.

Die farbigen Ringe auf den Widerständen geben den Widerstandswert an. Mit etwas Übung sind sie deutlich leichter zu erkennen als winzig kleine Zahlen, die man nur noch auf ganz alten Widerständen findet.

Die meisten Widerstände haben vier solcher Farbringe. Die ersten beiden Farbringe stehen für die Ziffern, der dritte bezeichnet einen Multiplikator und der vierte die Toleranz. Dieser Toleranzring ist meistens gold- oder silberfarben - Farben, die auf den ersten Ringen nicht vorkommen. Dadurch ist die Leserichtung immer eindeutig. Der Toleranzwert selbst spielt in der Digitalelektronik kaum eine Rolle. Da der

Toleranzring fast immer golden ist, ist er eine Hilfe beim Festlegen der Leserichtung für die Ringe. Die Tabelle zeigt die Bedeutung der farbigen Ringe auf Widerständen. Die 220-Ohm-Vorwiderstände für die LEDs haben den Farbcode Rot-Rot-Braun.

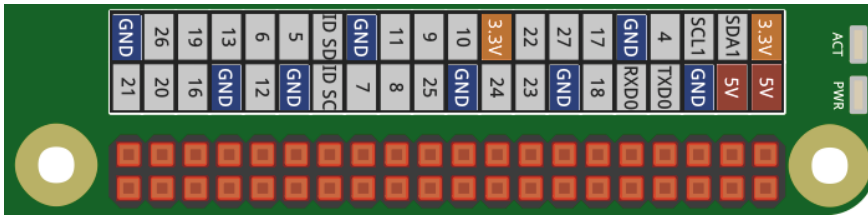
In welcher Richtung ein Widerstand eingebaut wird, ist egal. Bei LEDs dagegen spielt die Einbau- richtung eine wichtige Rolle.

Noch einfacher als mit diesen Tabellen ermitteln Sie den Widerstandswert aus einem Farbcode mit der Freeware **Widerstandsrechner** von www.ab-tools.com/de/software/widerstandsrechner. Umgekehrt können Sie sich damit auch den Farbcode zu einem beliebigen Widerstandswert anzeigen lassen.

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

GPIO-Verbindungskabel

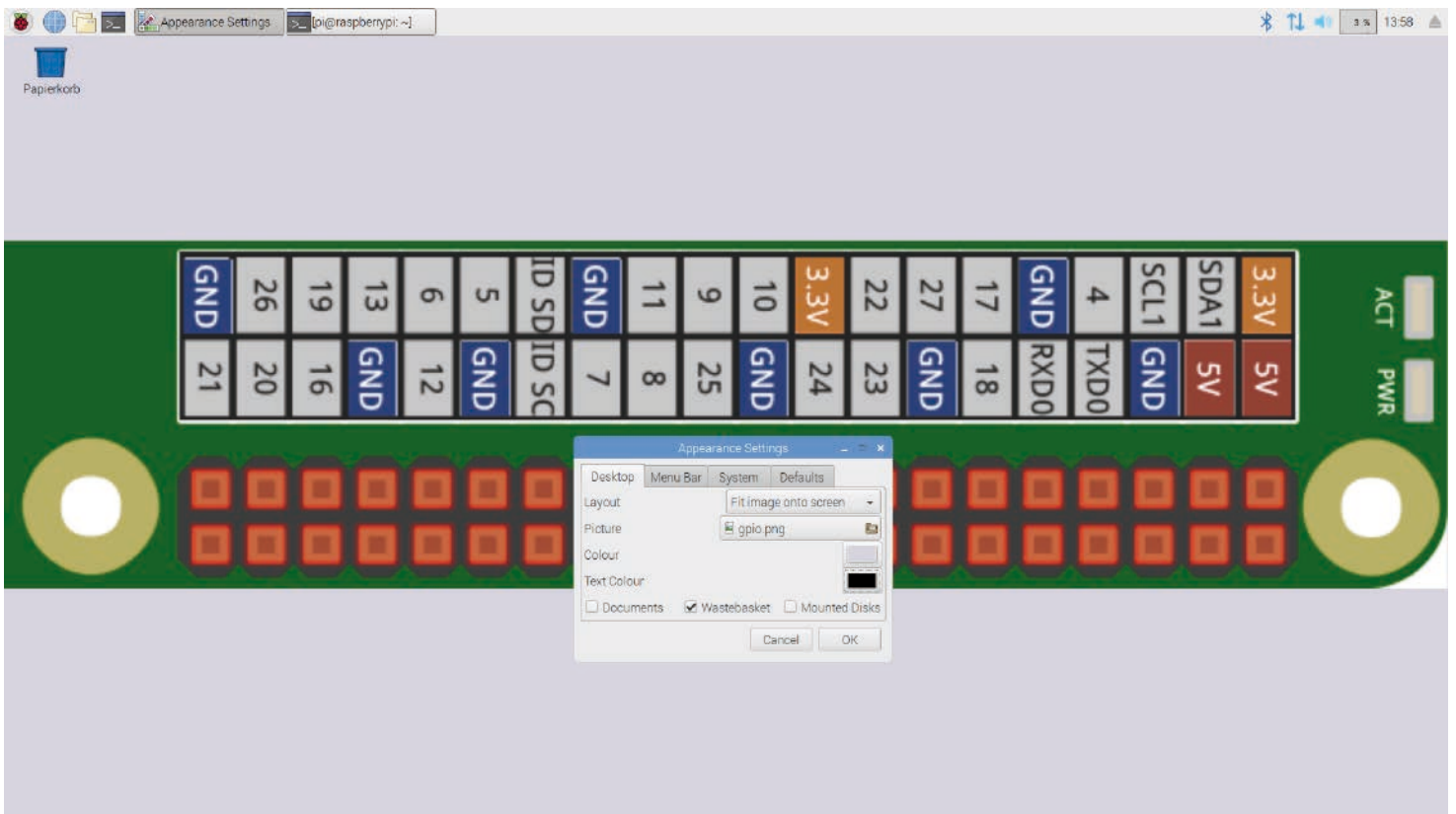
Die farbigen Verbindungskabel haben alle auf einer Seite einen Stecker, auf der anderen Seite eine Steckbuchse, die auf einen GPIO-Pin des Raspberry Pi passt. Die LEDs werden ebenfalls direkt in diese Steckbuchsen gesteckt. Die Stecker werden in das Steckbrett gesteckt. Die programmierbaren GPIO-Pins auf dem Raspberry Pi haben Nummern, die Masse-Pins sind in der Abbildung mit GND gekennzeichnet.



Belegung der GPIO-Pins.

Dieses Bild ist bei den Downloads zum Adventskalender dabei. Speichern Sie es auf dem Raspberry Pi, um jederzeit darauf Zugriff zu haben, da sich nur die wenigsten die Pin-Nummern merken können.

Sie können das Bild auch einfach als Desktophintergrund festlegen, um es immer im Blick zu haben. Klicken Sie dazu mit der rechten Maustaste auf den Desktop und im nächsten Dialogfeld auf das Standardbild `road.jpg`. Wählen Sie die Datei `gpio.png` aus dem Ordner `pi`. Wählen Sie bei **Layout** die Option **Fit image onto screen**.



Der Raspbian Desktop mit dem neuen Hintergrund

Vorsichtsmaßnahmen

Auf keinen Fall sollte man irgendwelche GPIO-Pins miteinander verbinden und abwarten ab, was passiert.

Nicht alle GPIO-Pins lassen sich frei programmieren. Ein paar sind für die Stromversorgung und andere Zwecke fest eingerichtet.

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 Schutzwiderstand dazwischengeschaltet werden. Eine Ausnahme bilden die LEDs mit eingebautem Vorwiderstand.

Für Logiksignale muss immer Pin 1 verwendet werden, der +3,3 V liefert und bis 50 mA belastet werden kann. Pin 6 ist die Masseleitung für Logiksignale.

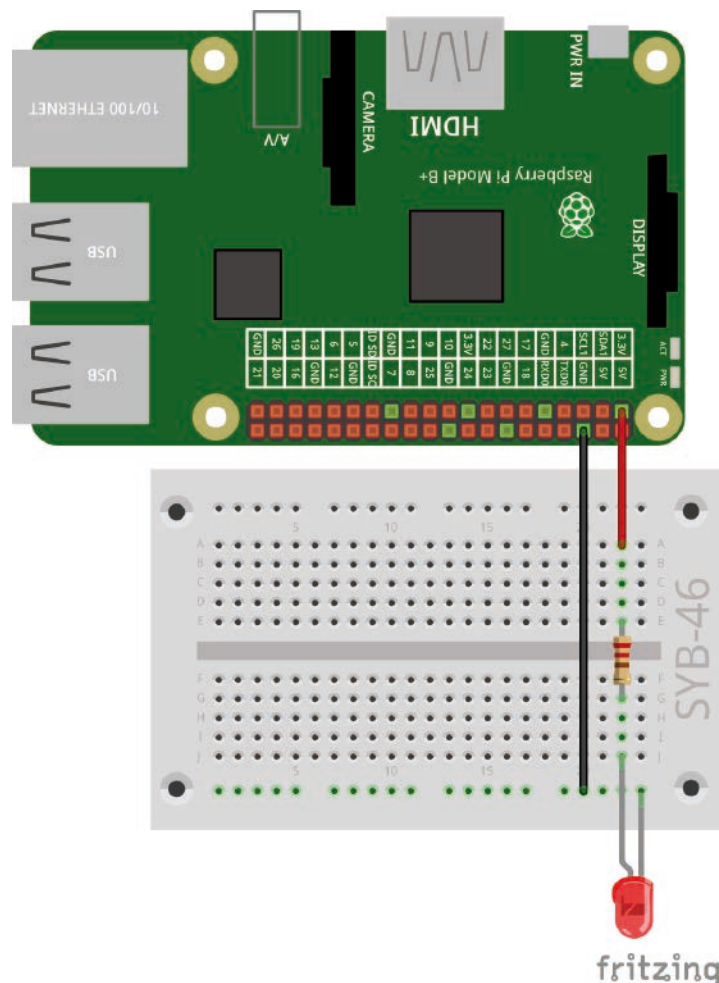
Pin 2 und 4 liefern +5 V zur Stromversorgung externer Hardware. Hier kann so viel Strom entnommen werden, wie das USB-Netzteil des Raspberry Pi liefert. Diese Pins dürfen aber nicht mit einem GPIO-Eingang verbunden werden.

LED leuchtet

Für das erste Experiment wird kein Programm benötigt. Der Raspberry Pi dient hier nur als Stromversorgung für die LED. Das Experiment zeigt, wie LEDs angeschlossen werden. Achten Sie darauf, dass die LED richtig herum eingebaut ist. Die flache Seite ist in der Abbildung rechts.

Die meisten Schaltungsaufbauten verwenden die Kontaktleiste an der einen Längsseite des Steckbretts als Massekontakt. Dort hinein werden die Kathoden aller LEDs gesteckt und mit einem Kabel mit einem GND-Pin auf dem Raspberry Pi verbunden.

Bauteile: 1 Steckbrett SYB-46, 1 LED rot, 1 220-Ohm-Widerstand (rot-rot-braun), 2 GPIO-Verbindungskabel



Die erste LED leuchtet am Raspberry Pi.

2. Tag

Heute im Adventskalender

- 1 LED grün
- 1 Widerstand 220 Ohm
- 1 GPIO-Verbindungskabel

Zwei LEDs blinken abwechselnd

Das Experiment des 2. Tags lässt zwei LEDs abwechselnd rot und grün leuchten. Gesteuert wird das Ganze über eine Endlosschleife in Scratch.

Bauteile: 1 Steckbrett SYB-46, 1 LED rot, 1 LED grün, 2 220-Ohm-Widerstände (rot-rot-braun), 3 GPIO-Verbindungskabel

Diesmal leuchtet die LED nicht permanent, sondern wird von einem Programm in der Programmiersprache Scratch für eine halbe Sekunde eingeschaltet.

Scratch ist auf dem Raspberry Pi im Menü - bei Klick auf das Raspberry-Logo links oben - unter **Entwicklung** vorinstalliert und gilt als eine der am leichtesten zu erlernenden Programmiersprachen.

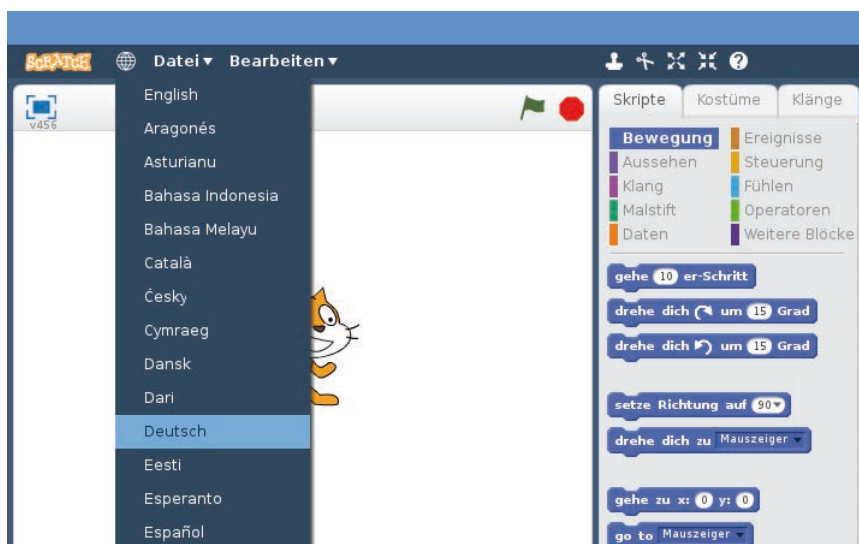
Das neue Scratch 2

Seit der ersten Raspbian-Version war die Programmiersprache Scratch in der Version 1.x vorinstalliert. Die neue Version Scratch 2 bietet deutlich mehr Möglichkeiten. Unter anderem lassen sich eigene Funktionsblöcke erstellen.

Scratch 2 läuft auf dem PC online im Browser. Dafür ist allerdings mehr Rechenleistung erforderlich, als ein Raspberry Pi zurzeit bieten kann. Seit der Version NOOBS 2.4.0 ist im Raspbian-Betriebssystem eine Version von Scratch 2 vorinstalliert, die offline ohne Browser läuft und so mit der Leistung eines Raspberry Pi 3 problemlos auskommt. Mit Scratch 2 ist die Hardwaresteuerung über die GPIO-Schnittstelle wesentlich einfacher geworden. Wir verwenden für die Scratch-Projekte in diesem Adventskalender die neue Version Scratch 2 aus dem Menü **Entwicklung**.

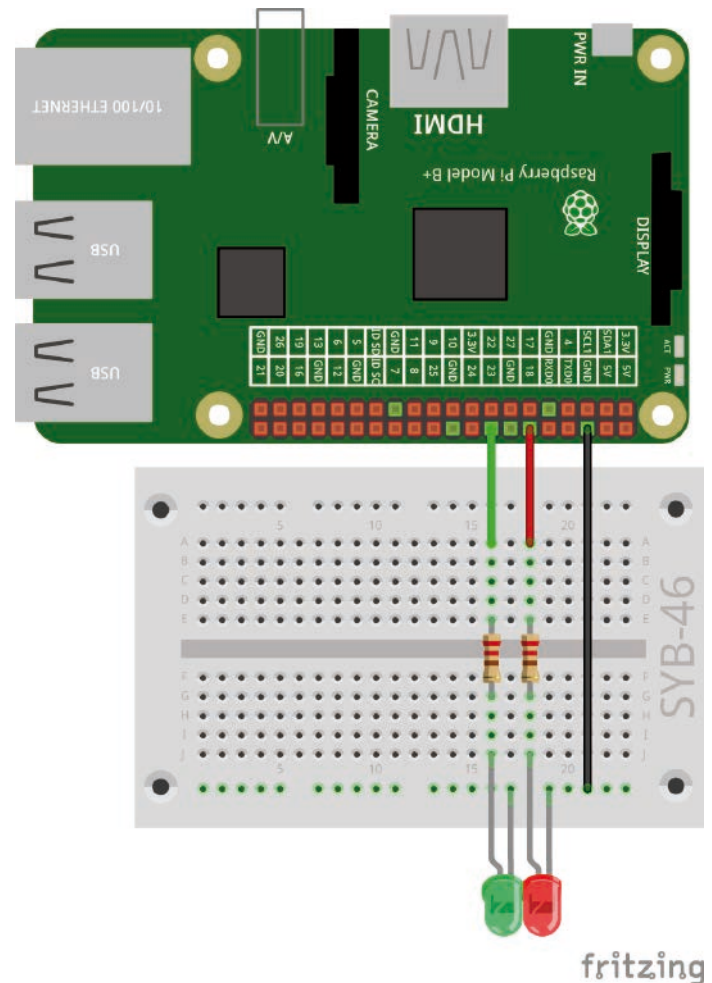
Grundeinstellungen für Scratch

Klicken Sie oben neben dem Scratch-Logo auf die Weltkugel und wählen Sie **Deutsch**. Die ausgewählte Sprache bleibt gespeichert, muss also nicht jedes Mal neu ausgewählt werden.



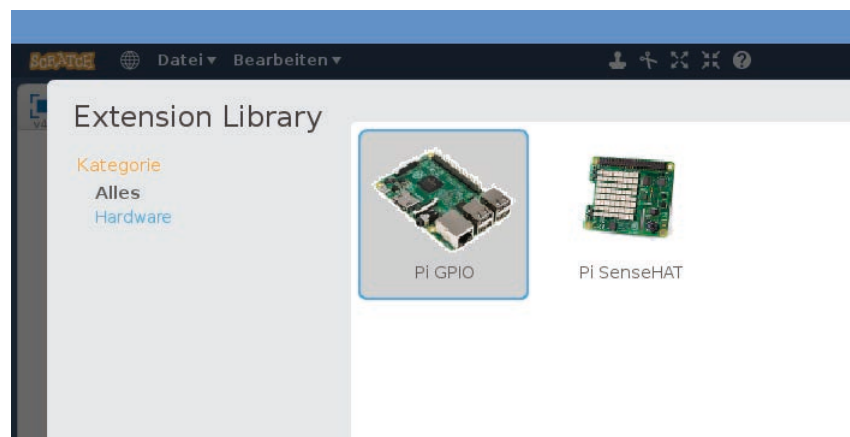
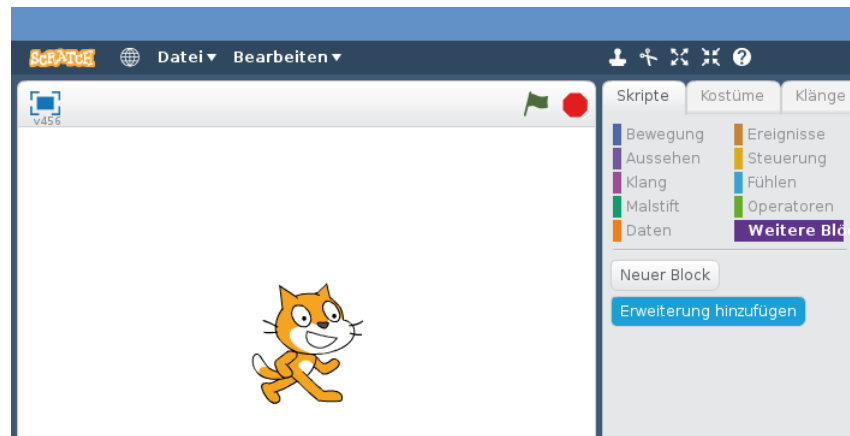
Scratch 2 auf Deutsch umschalten

2. Tag



Zwei LEDs blinken am Raspberry Pi.

Scratch 2 bietet eine Unterstützung für verschiedene Hardwarekomponenten am GPIO-Port, die über ein Zusatzmodul einmal aktiviert werden muss. Klicken Sie dazu auf der Blockpalette auf **Weitere Blöcke** und dann auf **Erweiterung hinzufügen**. Wählen Sie hier die Erweiterung **Pi GPIO** und installieren Sie sie per Doppelklick.



Scratch 2 Pi-GPIO-Erweiterung installieren

Das Programm

In Scratch braucht man beim Programmieren keinen Programmcode zu tippen. Die Blöcke werden einfach nur per Drag and Drop aneinander gehängt. Die Blockpalette in der Mitte des Scratch-Fensters enthält, nach Themen geordnet, die verfügbaren Blöcke.



Dieses Scratch-Programm 021led02 lässt zwei LEDs abwechselnd blinken.

Sie können das Programm auf dem Bildschirm selbst zusammenbauen oder das Programm 021led02 aus dem Download zum Adventskalender verwenden. Wählen Sie dazu im Menü **Datei/Load Project** und klicken im nächsten Dialogfeld auf die Schaltfläche **pi**, um das persönliche Home-Verzeichnis auszuwählen, in dem die heruntergeladenen Programme liegen.

Um das Programm selbst zusammenzubauen, klicken Sie auf der Blockpalette im mittleren Bereich des Scratch-Fensters oben auf das gelbe Symbol **Steuerung**. Dann werden in der Blockpalette die Blöcke zur Steuerung angezeigt.

Ziehen Sie die Blöcke, die Sie brauchen, einfach aus der Blockpalette in das Skriptfenster in der rechten Bildschirmhälfte von Scratch.

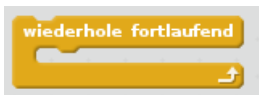


Der Block **Wenn (grüne Fahne) angeklickt** dient dazu, ein Programm zu starten. Der Block ist oben rund, passt also unter keinen anderen Block. Er muss immer als Erstes gesetzt werden.

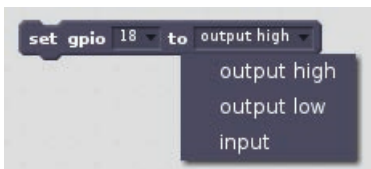
Die folgenden Skriptelemente werden ausgeführt, wenn man auf das grüne Fähnchen rechts oben auf der Scratch-Bühne klickt. Die Scratch-Bühne ist das Teilfenster links oben mit der Katze. Hier spielen auch interaktive Programme.



Eine **wiederhole fortlaufend**-Schleife sorgt dafür, dass die beiden LEDs endlos blinken, und zwar so lange, bis der Benutzer auf das rote Stoppsymbol auf der Scratch-Bühne klickt.



Als Erstes soll die rote LED an Pin 18 eingeschaltet und die grüne an Pin 23 ausgeschaltet werden. Ziehen Sie dazu einen Block **set gpio ... to ...** aus der PI-GPIO-Erweiterung in das Programm und wählen Sie im linken Listenfeld den Pin **18**. Wählen Sie im rechten Listenfeld **output high**, um diesen Pin als Ausgang zu definieren und einzuschalten.



Im nächsten Schritt wird über einen weiteren Scratch-Block **set gpio ... to ...** die am GPIO-Pin 23 angeschlossene grüne LED ausgeschaltet. Wählen Sie dazu im rechten Listenfeld des Blocks **output low**.

Nachdem die rote LED an Pin 18 eingeschaltet und die grüne an Pin 23 ausgeschaltet ist, wartet das Programm eine halbe Sekunde. Dazu bietet Scratch 2 einen eigenen Block **warte...Sek.** an. Scratch 2 verwendet wie viele amerikanische Programme den Punkt als Dezimaltrennzeichen, nicht das in Deutschland übliche Komma. Eine halbe Sekunde wird also als 0.5 eingetragen und nicht als 0,5.



Danach werden auf die gleiche Weise die rote LED an Pin 18 ausgeschaltet die und grüne an Pin 23 eingeschaltet. Nach einer weiteren halben Sekunde wiederholt sich der Zyklus.



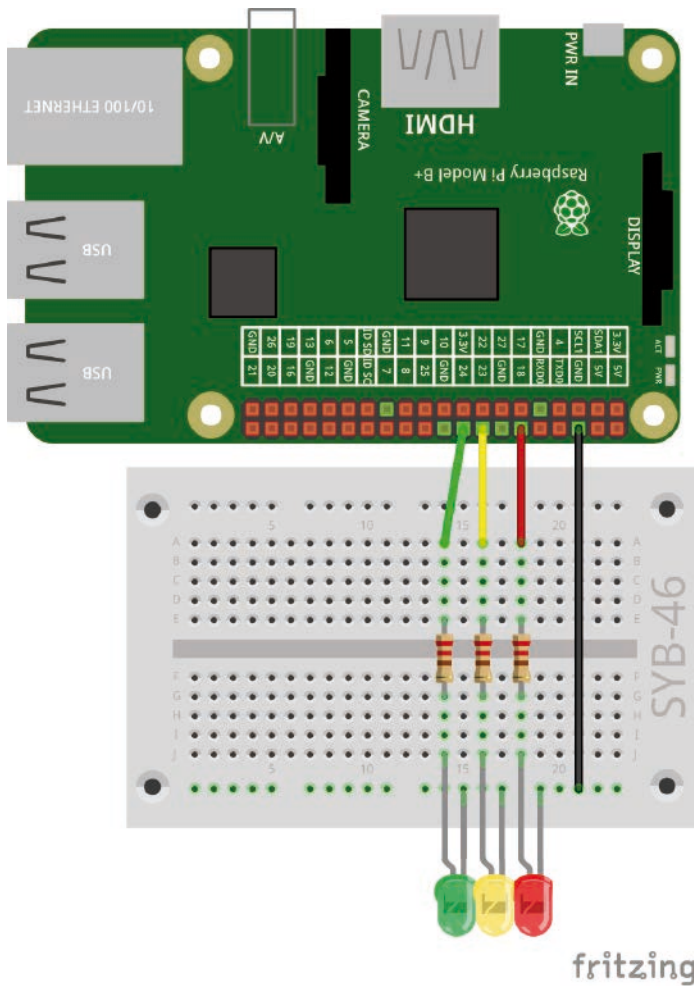
Das Programm startet, wenn man auf das grüne Fähnchen klickt, und endet beim Klick auf das rote Stopp-Symbol.

3. Tag

3. Tag

Heute im Adventskalender

- 1 LED gelb
- 1 Widerstand 220 Ohm
- 1 GPIO-Verbindungskabel



Eine einfache Verkehrsampel

Ampelschaltung mit Python

Eine Verkehrsampel mit ihrem typischen Leuchtzyklus von Grün über Gelb nach Rot und dann über eine Lichtkombination Rot-Gelb wieder zu Grün ist mit drei LEDs leicht aufzubauen und zeigt, wie man in der beliebten Programmiersprache Python die GPIO-Pins am Raspberry Pi steuert.

Bauteile: 1 Steckbrett SYB-46, 1 LED rot, 1 LED gelb, 1 LED grün, 3 220-Ohm-Widerstände (rot-rot-braun), 4 GPIO-Verbindungskabel

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 komplizierten Regeln zu beachten sind, macht das Programmieren wirklich Spaß.

Python 3 mit dem Code-Editor Mu

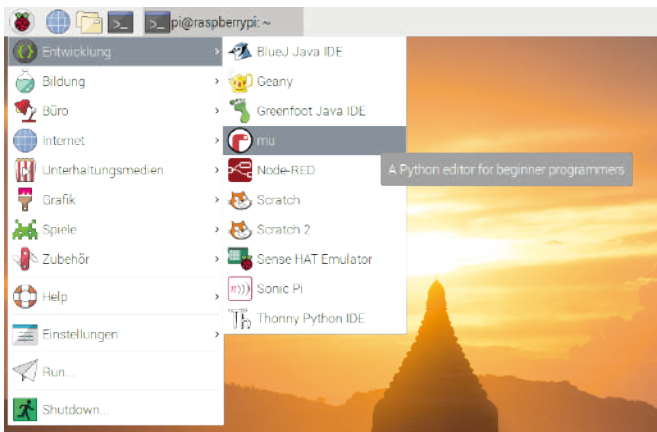
Seit der Betriebssystemversion NOOBS 3.1.1, die speziell für den Raspberry Pi 4 entwickelt wurde, ist statt der klassischen Python-Entwicklungsumgebung IDLE der neue Code-Editor **Mu** im Menü **Entwicklung** vorinstalliert. Mu ist eine komplette Entwicklungsumgebung. Für den Start in die Programmierung sind keine zusätzlichen Komponenten nötig.

Starten Sie im Menü unter **Entwicklung** das Programm **Python 3. IDLE** ist eine komplette Python-Shell und Entwicklungsumgebung. Für den Start in die Programmierung sind keine zusätzlichen Komponenten nötig.

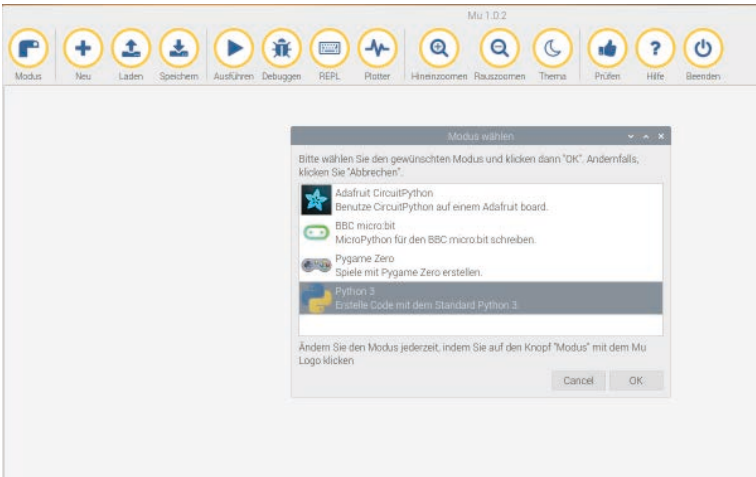
Wählen Sie beim ersten Start von Mu den Modus **Python 3**. Über das Symbol **Modus** können Sie jederzeit auch wieder zu einer anderen Programmiersprache wechseln.

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/pythonflashcards3. Diese Flashcards erklären nicht die technischen Hintergründe, sondern beschreiben nur anhand ganz kurzer Beispiele die Syntax, also wie etwas gemacht wird.



Der Code-Editor Mu im Menü.



Der erste Start von Mu.

BEDINGUNGEN	8	IF ELSE	9
<pre>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")</pre>		<pre>alter=10 if alter>17: print("Du darfst Auto fahren") else: print("Du bist nicht alt genug")</pre>	
python 3 V1 (deutsch) – softwarehandbuch.de		python 3 V1 (deutsch) – softwarehandbuch.de	
IF ELIF ELSE	10	AND/OR BEDINGUNGEN	11
<pre>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")</pre>		<pre>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")</pre>	
python 3 V1 (deutsch) – softwarehandbuch.de		python 3 V1 (deutsch) – softwarehandbuch.de	

Ausschnitt aus den Python-Flashcards

Das Programm

Anstatt uns mit Programmiertheorie, Algorithmen und Datentypen aufzuhalten, schreiben wir gleich das erste kleine Programm in Python.

Tippen Sie im Hauptfenster von Mu den abgebildeten Programmcode ein. Bereits beim Tippen sehen Sie interaktive Hilfetexte. Mu erkennt die Wörter und Standardfunktionen der Programmiersprache Python und zeigt Tipps zur Verwendung an.

Die Zeile:

```
# Schreibe hier Deinen Code :-)
```

ist wie alle Zeilen, die mit einem #-Zeichen beginnen, nur Kommentar und kann problemlos gelöscht werden.

Speichern Sie die Datei mit dem Symbol **Speichern** als 03ampel.py ab. Oder Sie öffnen die fertige Programmdatei aus dem Download in der mit dem Symbol **Laden**. Die Farbcodierung im Quelltext erscheint automatisch und hilft dabei, Tippfehler zu finden.

```
*03ampel.py - /home/pi/03ampel.py (3.5.3)*
File Edit Format Run Options Window Help
#!/usr/bin/python
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

rot    = 18
gelb   = 23
gruen  = 24

GPIO.setup(rot, GPIO.OUT, initial=0)
GPIO.setup(gelb, GPIO.OUT, initial=0)
GPIO.setup(gruen, GPIO.OUT, initial=1)

print("Strg+C beendet das Programm")

try:
    while True:
        time.sleep(2)
        GPIO.output(gruen,0)
        GPIO.output(gelb,1)
        time.sleep(0.6)
        GPIO.output(gelb,0)
        GPIO.output(rot,1)
        time.sleep(2)
        GPIO.output(gelb,1)
        time.sleep(0.6)
        GPIO.output(rot,0)
        GPIO.output(gelb,0)
        GPIO.output(gruen,1)
except KeyboardInterrupt:
    GPIO.cleanup()
|
```

Das erste Programm in Python 3

```

1 #!/usr/bin/python
2 import RPi.GPIO as GPIO
3 import time
4
5 GPIO.setmode(GPIO.BCM)
6
7 rot = 18
8 gelb = 23
9 gruen = 24
10
11 GPIO.setup(rot, GPIO.OUT, initial=0)
12 GPIO.setup(gelb, GPIO.OUT, initial=0)
13 GPIO.setup(gruen, GPIO.OUT, initial=1)
14
15 print("Strg+C beendet das Programm")
16
17 try:
18     while True:
19         time.sleep(2)
20         GPIO.output(gruen, 0)
21         GPIO.output(gelb, 1)
22         time.sleep(0.6)
23         GPIO.output(gelb, 0)
24         GPIO.output(rot, 1)
25         time.sleep(2)
26         GPIO.output(gelb, 1)
27         time.sleep(0.6)
28         GPIO.output(rot, 0)
29         GPIO.output(gelb, 0)
30         GPIO.output(gruen, 1)
31
32 except KeyboardInterrupt:
33     GPIO.cleanup()
34

```

Das erste Programm in Python 3

Starten Sie das Programm mit dem Symbol **Ausführen**.

So funktioniert es

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

Dieses erste Projekt zeigt die grundlegenden Elemente von Python und der `RPi.GPIO`-Bibliothek.

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

Python-Programme, die über die Kommandozeile gestartet werden, müssen am Anfang immer obige Zeile enthalten. Bei Programmen, die nur über die Python-Shell gestartet werden, ist das nicht nötig. Aus Gründen der Kompatibilität sollten Sie sich aber angewöhnen, diese Zeile am Anfang jedes Python-Programms einzutragen.

```
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.

```
import time
```

Die häufig verwendete 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 in einem Programm realisieren lassen.

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

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

Nummerierung der GPIO-Pins

Die Bibliothek `RPi.GPIO` unterstützt zwei verschiedene Methoden zur Bezeichnung der Pins. Im Modus `BCM` werden die bekannten GPIO-Portnummern genutzt, die auch auf Kommandozeilenebene oder in Scratch verwendet werden. Im alternativen, weitgehend ungebräuchlichen Modus `BOARD` entsprechen die Bezeichnungen den Pin-Nummern von 1 bis 40 auf der Raspberry-Pi-Platine, der Reihe nach durchnummeriert.

```
rot = 18
gelb = 23
gruen = 24
```

Diese drei Zeilen definieren Variablen mit den GPIO-Pins, an denen die drei LEDs angeschlossen sind. Das macht das Programm übersichtlicher, und es lässt sich leichter an einen anderen Schaltungsaufbau anpassen, da die Pinnummern nur an einer Stelle im Programm geändert werden müssen.

```
GPIO.setup(rot, GPIO.OUT, initial=0)
GPIO.setup(gelb, GPIO.OUT, initial=0)
GPIO.setup(gruen, GPIO.OUT, initial=1)
```

Nacheinander werden die drei verwendeten GPIO-Pins als Ausgänge initialisiert. Dabei verwenden wir keine GPIO-Portnummern, sondern die zuvor definierten Variablen. Die `GPIO.setup()`-Anweisung kann einen optionalen Parameter `initial` enthalten, der dem GPIO-Pin bereits beim Initialisieren einen logischen Status zuweist. Damit schalten wir in diesem Programm die grüne LED bereits von Anfang an ein. Die anderen beiden LEDs beginnen das Programm im ausgeschalteten Zustand.

```
print("Strg+C beendet das Programm")
```

Jetzt wird eine kurze Bedienungsanleitung auf dem Bildschirm ausgegeben. 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.

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

Durch diese Tastenkombination wird ein `KeyboardInterrupt` ausgelöst und die Schleife 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:

```
while True:
```

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

```
    time.sleep(2)
```

Am Anfang des Programms und auch bei jedem neuen Beginn der Schleife leuchtet die grüne LED 2 Sekunden lang.

```
    GPIO.output(gruen,0)
    GPIO.output(gelb,1)
    time.sleep(0.6)
```

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

```
    GPIO.output(gelb,0)
    GPIO.output(rot,1)
    time.sleep(2)
```

Jetzt wird die gelbe LED wieder ausgeschaltet und dafür die rote LED eingeschaltet. Diese leuchtet dann alleine für 2 Sekunden. Die Rotphase einer Ampel ist üblicherweise deutlich länger als die Gelbphase.

```
    GPIO.output(gelb,1)
    time.sleep(0.6)
```

Zum Start der Rot-Gelb-Phase wird die gelbe LED zusätzlich eingeschaltet, ohne dass eine andere LED ausgeschaltet wird. Diese Phase dauert 0,6 Sekunden lang.

```
    GPIO.output(rot,0)
    GPIO.output(gelb,0)
    GPIO.output(gruen,1)
```

Am Ende der Schleife springt die Ampel wieder auf Grün. Die rote und die gelbe LED werden ausgeschaltet, und die grüne LED wird eingeschaltet. Die Schleife beginnt in der Grünphase der Ampel von Neuem mit einer Wartezeit von 2 Sekunden. 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.

Diese Schleife wird wiederholt, bis der Benutzer die Tastenkombination [Strg]+[C] drückt.

4. Tag

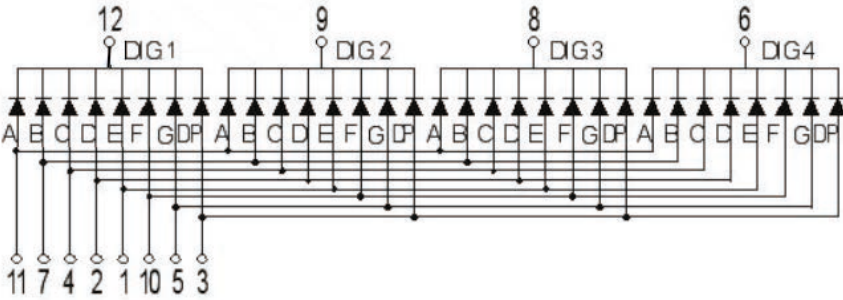
4. Tag

Heute im Adventskalender

• Sieben-Segment-Anzeige

Der Adventskalender enthält heute eine vierstellige Sieben-Segment-Anzeige, mit der sich Ziffern und einfache Symbole darstellen lassen. Solche Sieben-Segment-Anzeigen werden häufig in Digitaluhren genutzt, früher auch in Taschenrechnern. Tankstellen verwenden diese Anzeigetechnologie im großen Format. Allerdings sind hier oft statt LEDs mechanische Anzeigeelemente im Einsatz. Nicht alle Buchstaben des Alphabets lassen sich auf diesen Anzeigen eindeutig darstellen, die Buchstaben A bis F, die zur Darstellung von Hexadezimalzahlen notwendig sind, funktionieren aber. Die meisten Sieben-Segment-Anzeigen verfügen noch über eine achte LED für den Dezimalpunkt.

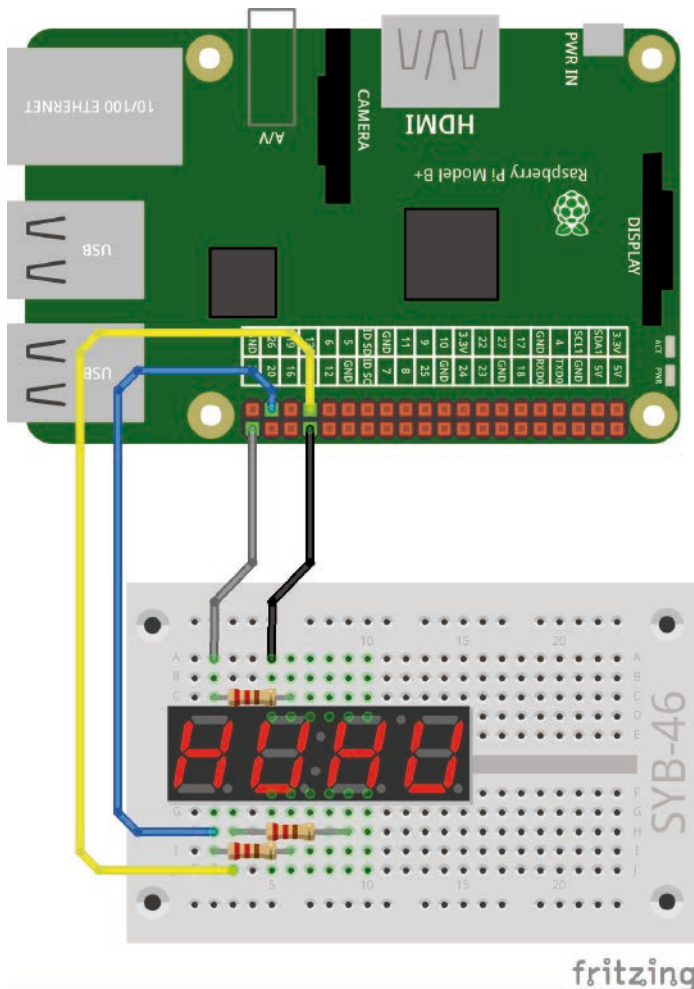
Bei einer einstelligen Sieben-Segment-Anzeige ist jede LED einzeln über ihre Anode ansteuerbar. Alle LEDs verwenden eine gemeinsame Kathode oder umgekehrt. Dementsprechend werden die Anzeigemodule als „Common Cathode“ oder „Common Anode“ bezeichnet.



Schaltschema der Sieben-Segment-Anzeige

Bei der im Adventskalender mitgelieferten vierstelligen Sieben-Segment-Anzeige sind nach dem Common-Cathode-Schaltenschema die Kathoden der acht LEDs einer Ziffer zusammengeschaltet. Zusätzlich sind die vier Anoden der jeweils gleichen LED-Position in allen vier Ziffern zusammengeschaltet. Dadurch braucht die gesamte Anzeige für insgesamt 32 LEDs statt 64 nur 12 Anschlüsse.

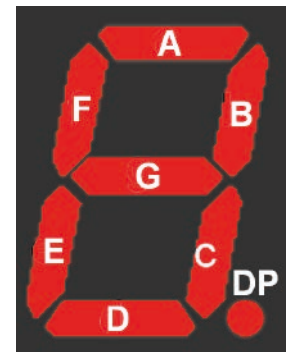
Die Anschlüsse der Anoden sind nach den vier Ziffern mit 1 bis 4 bezeichnet, die Anschlüsse der Kathoden nach den sieben Segmenten mit A bis G sowie DP für den Dezimalpunkt.



Drei Segmente der Sieben-Segment-Anzeige blinken abwechselnd.



Die Anschlüsse auf der Sieben-Segment-Segmente Anzeige



Die Bezeichnungen der Segmente

Um eine einzelne LED auf der Anzeige leuchten zu lassen, muss die Kathode der entsprechenden Ziffer mit der Masseleitung und die Anode des Segments mit einem +3,3-V-Signal verbunden sein. Vor jeder LED muss wie üblich ein 220-Ohm-Vorwiderstand angeschlossen werden.

Drei Segmente der Sieben-Segment-Anzeige blinken abwechselnd

Als erstes einfaches Beispiel werden die drei horizontalen Segmente der Sieben-Segment-Anzeige abwechselnd blinken.

Bauteile: 1 Steckbrett SYB-46, 1 Sieben-Segment-Anzeige, 3 220-Ohm-Widerstände (rot-rot-braun), 4 GPIO-Verbindungskabel

Anschlusschema der Sieben-Segment-Anzeige

Wenn die Sieben-Segment-Anzeige eingebaut ist, sind die Anschlüsse schwer zu erkennen. Sie ist in allen Experimenten in diesem Adventskalender auf dem Steckbrett in den Reihen 5 bis 10 eingebaut.

Die folgende Tabelle zeigt, welche Pins der Sieben-Segment-Anzeige mit welchen GPIO-Pins verbunden sind. Die grau hinterlegten Pins werden in diesem Programm nicht verwendet.

In diesem Programm werden nur die drei waagerechten Segmente A, G und D über Vorwiderstände an GPIO-Pins angeschlossen. Die Ziffer 1 ist mit GND verbunden, damit die Segmente der ersten Ziffer blinken.

Das Programm

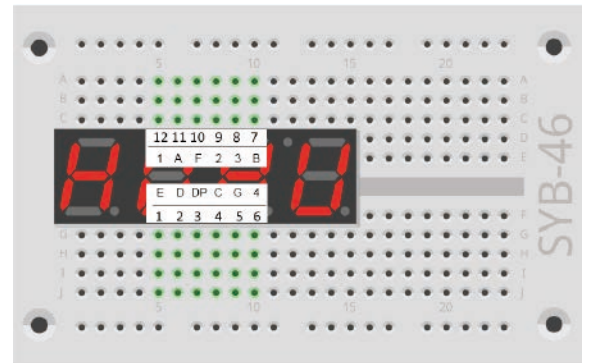
Das Programm `04seg7.py` lässt die drei LEDs der waagerechten Segmente A, G und D abwechselnd nacheinander blinken.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

seg=[21, 13, 26]
for s in range(3):
    GPIO.setup(seg[s], GPIO.OUT, initial=0)

print("Strg+C beendet das Programm")
try:
    while True:
        for i in range(3):
            GPIO.output(seg[i], 1)
            time.sleep(0.2)
            GPIO.output(seg[i], 0)
except KeyboardInterrupt:
    GPIO.cleanup()
```



Sieben-Segment-Anzeige mit Anschlussnummern auf dem Steckbrett

Pin Sieben-Segment-Anzeige	Steckbrett	Segment / Ziffer	GPIO-Pin
1	5 F-J	E	-
2	6 F-J	D	26
3	7 F-J	DP	-
4	8 F-J	C	-
5	9 F-J	G	13
6	10 F-J	4	-
7	10 A-E	B	-
8	9 A-E	3	-
9	8 A-E	2	-
10	7 A-E	F	-
11	6 A-E	A	21
12	5 A-E	1	GND

So funktioniert es

Die ersten Zeilen sind bereits bekannt, sie importieren die Bibliotheken `RPi.GPIO` für die Ansteuerung der GPIO-Ports und `time` für Zeitverzögerungen. Danach wird die Nummerierung der GPIO-Ports wie im vorherigen Beispiel auf BCM gesetzt.

```
seg=[21, 13, 26]
```

Zur Ansteuerung der drei LEDs wird eine Liste eingerichtet, die die GPIO-Nummern der verwendeten Pins enthält.

```
for s in range(3):
    GPIO.setup(seg[s], GPIO.OUT, initial=0)
```

Nacheinander werden die drei verwendeten GPIO-Ports als Ausgänge initialisiert und ausgeschaltet. Dabei verwenden wir diesmal keine GPIO-Portnummern, sondern die zuvor definierte Liste. Innerhalb einer Liste werden die einzelnen Elemente über Zahlen, mit 0 beginnend, indiziert. `seg[0]` ist also das erste Element, in diesem Fall 21. Der Parameter `range()` in der `for`-Schleife gibt an, wie oft die Schleife durchlaufen wird. Genauer gesagt, bezeichnet er den ersten Wert, der nicht mehr erreicht wird.

Jede `for`-Schleife benötigt einen Schleifenzähler, eine Variable, die bei jedem Schleifendurchlauf einen neuen Wert annimmt. Dieser Wert kann wie jede andere Variable innerhalb der Schleife abgefragt werden. In den drei Schleifendurchläufen nimmt die Variable `s` die Werte 0, 1 und 2 an und initialisiert so die drei Elemente der Liste. Für das endlose Blinken verwenden wir wie im letzten Programm ein `try: ... except`-Konstrukt mit einer Endlosschleife, die über die Tastenkombination `[Strg]+[C]` abgebrochen wird.

```
while True:
    for i in range(3):
        GPIO.output(seg[i], 1)
        time.sleep(0.2)
        GPIO.output(seg[i], 0)
```

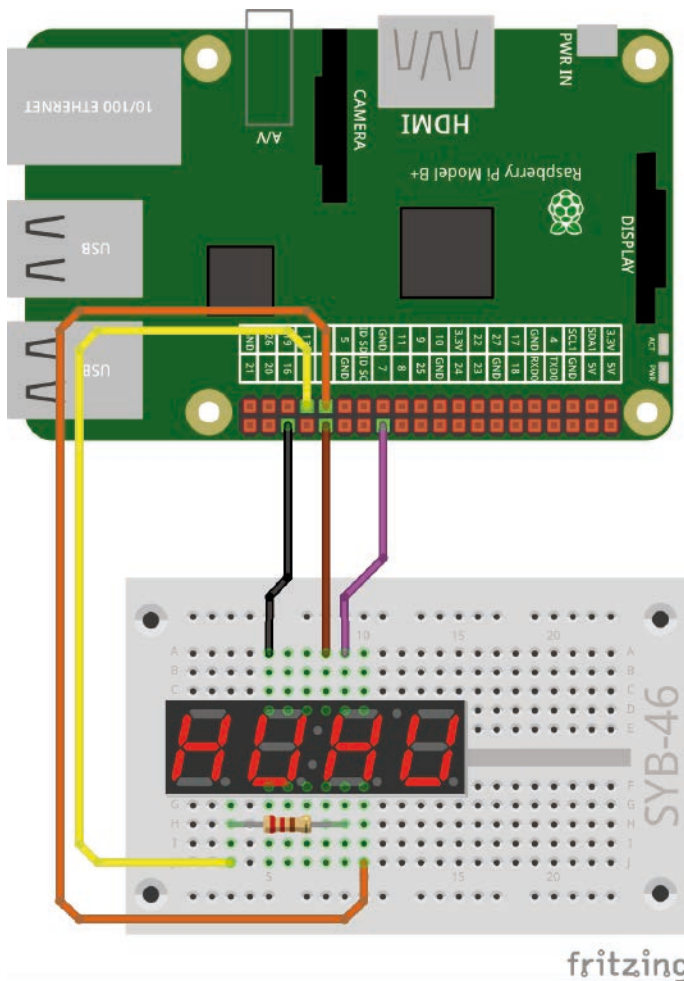
Innerhalb der Endlosschleife läuft eine weitere Schleife dreimal und schaltet in jedem Durchlauf eines der Segmente ein, wartet 0,2 Sekunden und schaltet es wieder aus. Danach kommt das nächste Segment aus der Liste an die Reihe.

5. Tag

5. Tag

Heute im Adventskalender

- 1 GPIO-Verbindungskabel



Ein Segment und vier Ziffern der Sieben-Segment-Anzeige sind angeschlossen.

Pin Sieben-Segment-Anzeige	Steckbrett	Segment / Ziffer	GPIO-Pin
1	5 F-J	E	-
2	6 F-J	D	-
3	7 F-J	DP	-
4	8 F-J	C	-
5	9 F-J	G	13
6	10 F-J	4	6
7	10 A-E	B	-
8	9 A-E	3	7
9	8 A-E	2	12
10	7 A-E	F	-
11	6 A-E	A	-
12	5 A-E	1	16

So funktioniert es

Das Programm funktioniert mit ein paar kleinen Unterschieden wie das Programm des 4. Tags.

Das Segment G am Pin 13 wird permanent eingeschaltet. Die Kathoden der vier Ziffern sind in der Liste `zif[]` definiert und werden in der Endlosschleife nacheinander kurz auf 0 gesetzt. Im Zustand 0 ist die Kathode mit Masse verbunden, das Segment leuchtet. Die Kathoden der anderen drei Segmente sind solange auf 1 gesetzt und damit mit +3,3 V verbunden. Die jeweiligen Segmente leuchten nicht.

Lauflicht auf der Sieben-Segment-Anzeige

Das Experiment des 5. Tags lässt das mittlere Segment aller vier Ziffern der Sieben-Segment-Anzeige als Lauflicht aufleuchten. Dazu werden die Kathoden der Ziffern vom Programm abwechselnd umgeschaltet, während die Anode für das Segment immer eingeschaltet bleibt.

Bauteile: 1 Steckbrett SYB-46, 1 Sieben-Segment-Anzeige, 1 220-Ohm-Widerstand (rot-rot-braun), 5 GPIO-Verbindungskabel

Diese Schaltung verwendet nur einen einzigen Vorwiderstand, der an der gemeinsamen Kathode angeschlossen ist. Ob der Vorwiderstand an der Anode vor der LED oder an der Kathode hinter der LED angeschlossen ist, spielt keine Rolle. Achten Sie jedoch darauf, dass jede LED ihren eigenen Vorwiderstand hat. In diesem Beispiel leuchtet immer nur eine LED, sodass sich hier kein Problem ergibt.

Anschlussschema der Sieben-Segment-Anzeige

Die folgende Tabelle zeigt, welche Pins der Sieben-Segment-Anzeige mit welchen GPIO-Pins verbunden sind. Die grau hinterlegten Pins werden in diesem Programm nicht verwendet.

Das Programm

Das Programm `05seg7.py` schaltet in einer Endlosschleife die G-Segmente aller vier Ziffern nacheinander ein und nach jeweils 0,2 Sekunden wieder aus, so dass sich ein Lauflichteffekt ergibt.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

seg=13

zif=[16, 12, 7, 6]

GPIO.setup(seg, GPIO.OUT, initial=1)
for z in range(4):
    GPIO.setup(zif[z], GPIO.OUT, initial=1)

print("Strg+C beendet das Programm")
try:
    while True:
        for i in range(4):
            GPIO.output(zif[i], 0)
            time.sleep(0.2)
            GPIO.output(zif[i], 1)
except KeyboardInterrupt:
    GPIO.cleanup()
```

6. Tag

Heute im Adventskalender

• 3 GPIO-Verbindungskabel

Noch ein Lauflicht auf der Sieben-Segment-Anzeige

Das Experiment des 6. Tags lässt alle Segmente der ersten Ziffer der Sieben-Segment-Anzeige als Lauflicht aufleuchten. Durch einen Schaltstrick ist das mit nur einem Vorwiderstand möglich.

Bauteile: 1 Steckbrett SYB-46, 1 Sieben-Segment-Anzeige, 1 220-Ohm-Widerstand (rot-rot-braun), 8 GPIO-Verbindungskabel

Auch diese Schaltung verwendet nur einen einzigen Vorwiderstand, der an der gemeinsamen Kathode angeschlossen ist.

Anschlussschema der Sieben-Segment-Anzeige

Die folgende Tabelle zeigt, welche Pins der Sieben-Segment-Anzeige mit welchen GPIO-Pins verbunden sind. Die grau hinterlegten Pins werden in diesem Programm nicht verwendet.

Pin Sieben-Segment-Anzeige	Steckbrett	Segment / Ziffer	GPIO-Pin
1	5 F-J	E	19
2	6 F-J	D	26
3	7 F-J	DP	-
4	8 F-J	C	11
5	9 F-J	G	13
6	10 F-J	4	-
7	10 A-E	B	8
8	9 A-E	3	-
9	8 A-E	2	-
10	7 A-E	F	20
11	6 A-E	A	21
12	5 A-E	1	16

Das Programm

Das Programm `06seg7.py` schaltet nacheinander die Segmente einer Ziffer ein und nach 0,1 Sekunden wieder aus. Der Lauflichteffekt beschreibt eine 8. Das Segment G in der Mitte wird in jedem Durchgang zweimal eingeschaltet.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time

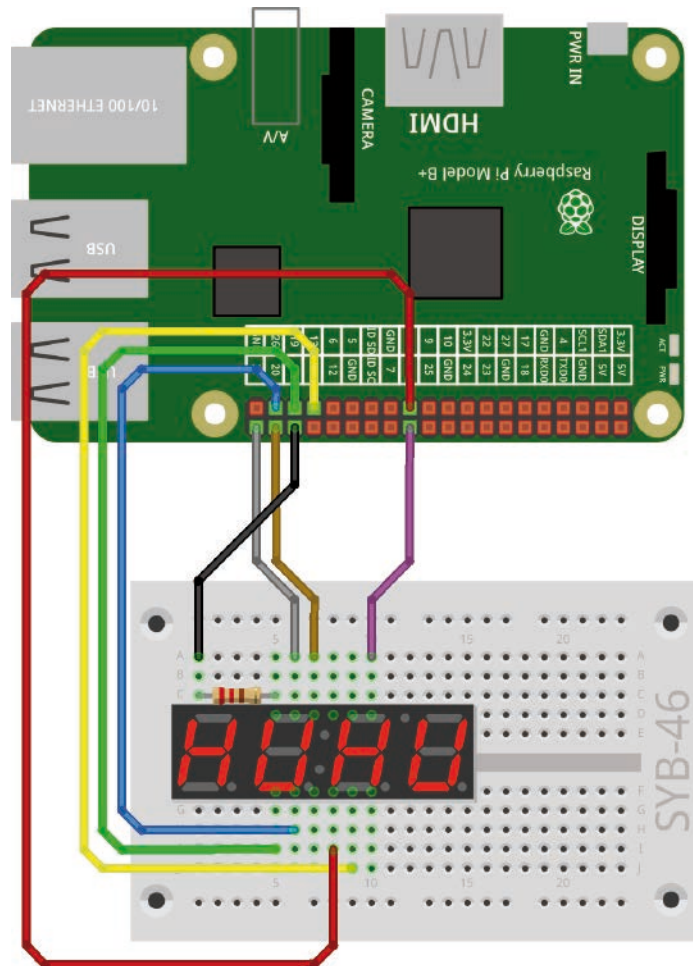
GPIO.setmode(GPIO.BCM)

seg={'a':21, 'b':8, 'c':11, 'd':26, 'e':19, 'f':20, 'g':13}
for s in "abcdefg":
    GPIO.setup(seg[s], GPIO.OUT, initial=0)

zif=16
GPIO.setup(zif, GPIO.OUT, initial=0)

print("Strg+C beendet das Programm")
try:
    while True:
        for s in "abgedcgf":
            GPIO.output(seg[s], 1)
            time.sleep(0.1)
            GPIO.output(seg[s], 0)
except KeyboardInterrupt:
    GPIO.cleanup()
```

6. Tag



fritzing

Sieben Segmente und eine Ziffer sind an der Sieben-Segment-Anzeige angeschlossen.

So funktioniert es

Für die sieben Segmente der Anzeige wird eine Variable 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 auch aus einem einzigen Buchstaben bestehen kann. Im Gegensatz zu einer einfachen Liste steht ein `dictionary` in geschweiften Klammern. Es kann beliebig viele Paare aus Schlüssel und Wert enthalten.

```
seg={'a':21, 'b':8, 'c':11, 'd':26, 'e':19, 'f':20, 'g':13}
```

In unserem Fall ist der Schlüssel der jeweilige Kennbuchstabe für das Segment, der Wert dahinter der verwendete GPIO-Pin.

```
for s in "abcdefg":
    GPIO.setup(seg[s], GPIO.OUT, initial=0)
```

Die GPIO-Pins aller sieben Segmente werden als Ausgänge definiert und auf 0, was bei einer Anode ausgeschaltet bedeutet, gesetzt. Die Schleife arbeitet nacheinander alle Buchstaben der angegebenen Zeichenkette ab, anstatt, wie sonst üblich, innerhalb eines bestimmten Zahlenbereichs hochzuzählen.

```
zif=16
GPIO.setup(zif, GPIO.OUT, initial=0)
```

Die Kathode der verwendeten Ziffer wird auf 0 gesetzt, um die Ziffer einzuschalten.

```
for s in "abgedcgf":
    GPIO.output(seg[s], 1)
    time.sleep(0.1)
    GPIO.output(seg[s], 0)
```

Bei jedem Durchlauf der Endlosschleife arbeitet eine innere Schleife die eingetragene Zeichenkette ab, setzt die entsprechenden Segmente der Anzeige nacheinander auf 1 und schaltet sie damit für 0,1 Sekunden ein.

Drückt der Benutzer die Tastenkombination [Strg]+[C], wird wie in vorherigen Experimenten die Endlosschleife beendet, und die GPIO-Ports werden geschlossen.

7. Tag

Heute im Adventskalender

• 1 Widerstand 220 Ohm

• Schaltdraht

Schaltdraht

Heute ist Schaltdraht im Adventskalender enthalten. Damit stellen Sie kurze Verbindungsbrücken her, mit denen Kontaktreihen auf der Steckplatine 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, sie leicht schräg abzuschneiden, sodass eine Art Keil entsteht.

Vier Ziffern zeigen das gleiche Blinkmuster

Das Programm des 7. Tags zeigt auf allen vier Ziffern ein Blinkmuster mit vier Segmenten. Das Programm zeigt, wie Sie auf der Basis eines früheren - allgemein gehaltenen - Programms mit nur minimalen Änderungen einen anderen Effekt erzielen.

Bauteile: 1 Steckbrett SYB-46, 1 Sieben-Segment-Anzeige, 4 220-Ohm-Widerstände (rot-rot-braun), 5 GPIO-Verbindungskabel, 4 Drahtbrücken (unterschiedliche Längen)

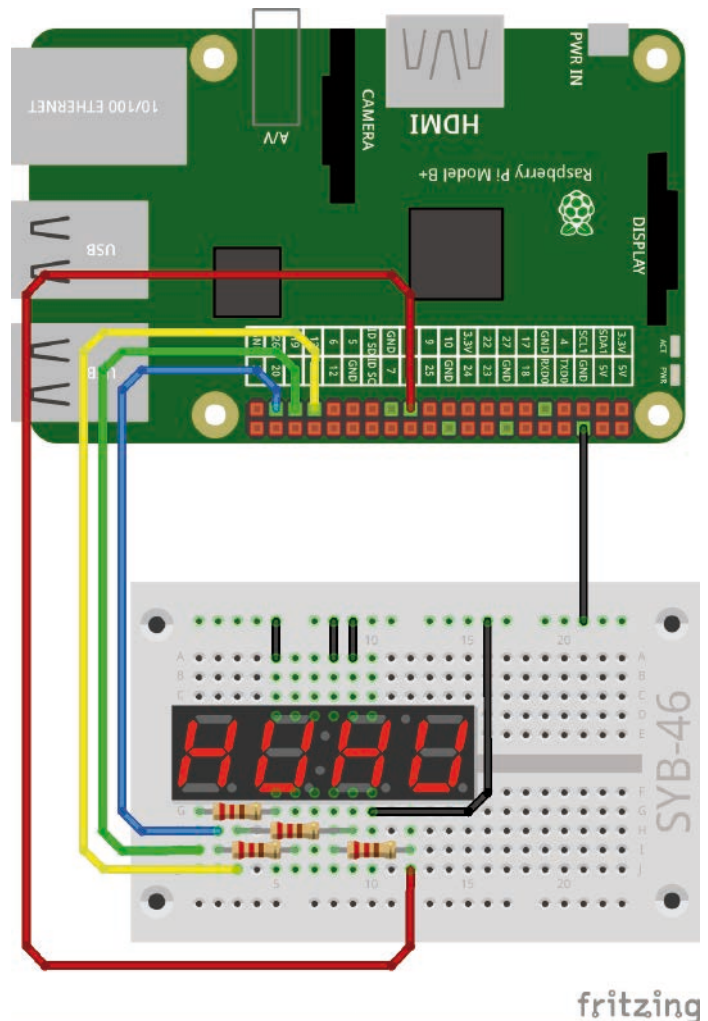
Die vier Kathoden der Ziffern sind über Drahtbrücken mit der in der Abbildung oberen Kontaktleiste des Steckbretts verbunden. Diese ist an einem GND-Pin des Raspberry Pi angeschlossen.

Anschlusschema der Sieben-Segment-Anzeige

Die folgende Tabelle zeigt, welche Pins der Sieben-Segment-Anzeige mit welchen GPIO-Pins verbunden sind. Die grau hinterlegten Pins werden in diesem Programm nicht verwendet.

Pin Sieben-Segment-Anzeige	Steckbrett	Segment / Ziffer	GPIO-Pin
1	5 F-J	E	19
2	6 F-J	D	26
3	7 F-J	DP	
4	8 F-J	C	11
5	9 F-J	G	13
6	10 F-J	4	GND
7	10 A-E	B	
8	9 A-E	3	GND
9	8 A-E	2	GND
10	7 A-E	F	
11	6 A-E	A	
12	5 A-E	1	GND

7. Tag



Vier Segmente der Sieben-Segment-Anzeige sind angeschlossen. Die Ziffern sind mit Masse verbunden.

Das Programm

Das Programm `07seg7.py` lässt auf allen vier Ziffern ein Lauflicht über die unteren vier Segmente (C, D, E, G) laufen.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

seg={'a':21, 'b':8, 'c':11, 'd':26, 'e':19, 'f':20, 'g':13}
for s in "abcdefg":
    GPIO.setup(seg[s], GPIO.OUT, initial=0)

zif=16
GPIO.setup(zif, GPIO.OUT, initial=0)

print("Strg+C beendet das Programm")
try:
    while True:
        for s in "cdeg":
            GPIO.output(seg[s], 1)
            time.sleep(0.1)
            GPIO.output(seg[s], 0)

except KeyboardInterrupt:
    GPIO.cleanup()
```

So funktioniert es

Das Programm unterscheidet sich vom vorherigen nur in einer einzigen Zeile:

```
for s in "cdeg":
```

Diese Zeile gibt an, welche Segmente in der Schleife aufleuchten sollen. Alle Definitionen und die Programmlogik bleiben gleich. Wegen der Parallelschaltung der Ziffern brauchen keine weiteren GPIO-Pins initialisiert zu werden.

Für diese spezielle Schaltung kann das allgemein gehaltene Programm noch vereinfacht werden, indem nicht benötigte Definitionen weggelassen werden, wie das Programm `07seg7_02.py` zeigt.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

seg={'c':11, 'd':26, 'e':19, 'g':13}
for s in "cdeg":
    GPIO.setup(seg[s], GPIO.OUT, initial=0)

print("Strg+C beendet das Programm")
try:
    while True:
        for s in "cdeg":
            GPIO.output(seg[s], 1)
            time.sleep(0.1)
            GPIO.output(seg[s], 0)

except KeyboardInterrupt:
    GPIO.cleanup()
```

8. Tag

Heute im Adventskalender

- 3 Widerstände 220 Ohm

8. Tag

Sieben-Segment-Anzeige mit Scratch steuern

Das Programm des 8. Tags steuert die Segmente der Sieben-Segment-Anzeige interaktiv und einzeln mit Scratch. Dazu werden neue, bisher noch nicht beschriebene Programmier Techniken von Scratch vorgestellt.

Bauteile: 1 Steckbrett SYB-46, 1 Sieben-Segment-Anzeige, 7 220-Ohm-Widerstände (rot-rot-braun), 8 GPIO-Verbindungskabel, 4 Drahtbrücken (unterschiedliche Längen)

Anschlusschema der Sieben-Segment-Anzeige

Die folgende Tabelle zeigt, welche Pins der Sieben-Segment-Anzeige mit welchen GPIO-Pins verbunden sind. Der grau hinterlegte Pin wird in diesem Programm nicht verwendet.

Pin Sieben-Segment-Anzeige	Steckbrett	Segment / Ziffer	GPIO-Pin
1	5 F-J	E	19
2	6 F-J	D	26
3	7 F-J	DP	
4	8 F-J	C	11
5	9 F-J	G	13
6	10 F-J	4	GND
7	10 A-E	B	8
8	9 A-E	3	GND
9	8 A-E	2	GND
10	7 A-E	F	20
11	6 A-E	A	21
12	5 A-E	1	GND

Das Programm

Das Scratch-2-Programm 08seg7.sb2 bietet eine grafische Oberfläche, um die sieben Segmente interaktiv einzeln ein- oder auszuschalten.

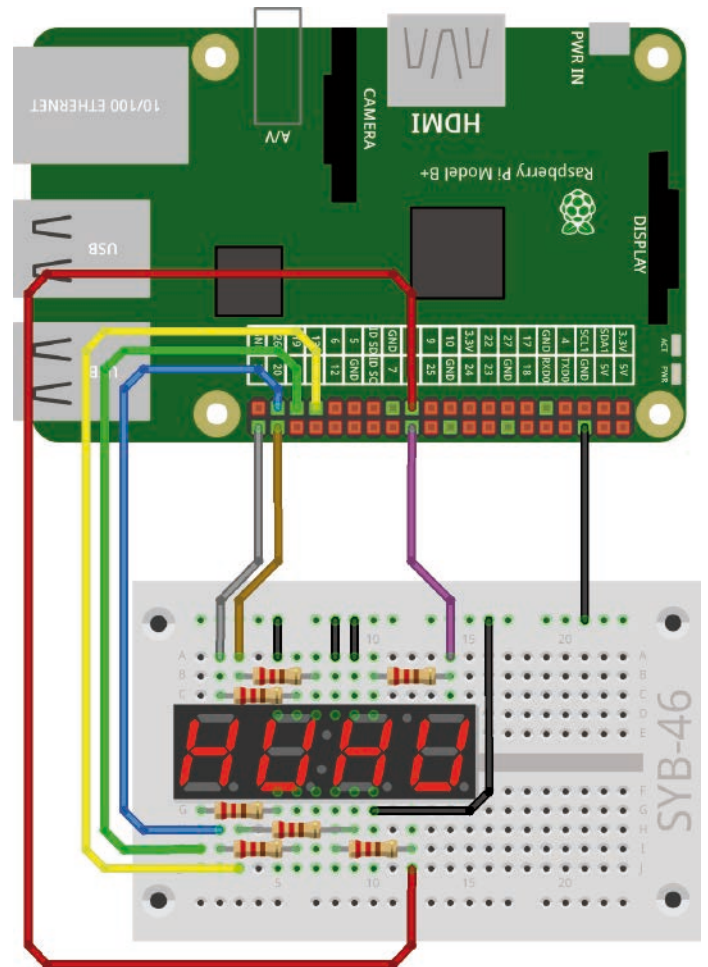
Löschen Sie als Erstes die Katze, die Scratch-Symbolfigur, die in diesem Programm nicht gebraucht wird. Klicken Sie dazu mit der rechten Maustaste auf die Katze im Figurenfenster links unten und wählen Sie im Kontextmenü **Löschen**.

Im Programm sollen rechteckige Figuren in der Anordnung der sieben Segmente die LEDs umschalten. Sie können diese Figuren direkt in Scratch malen. Klicken Sie dazu auf das Symbol **Neue Figur zeichnen** im Figurenfenster.

Scratch enthält ein einfaches Malprogramm, mit dem Sie Figuren malen können. Stellen Sie erst eine Figur komplett fertig, die weiteren können Sie später mitsamt aller Einstellungen und Skriptblöcke duplizieren und nur noch geringfügig anpassen.

Schalten Sie das Malprogramm unten rechts auf den Vektormodus. Zeichnen Sie dann mit dem Rechteckwerkzeug in der Symbolleiste rechts ein Rechteck etwa in der abgebildeten Größe. Dieses soll das obere waagerechte Segment der Sieben-Segment-Anzeige darstellen.

Jede Figur in Scratch kann über sogenannte Kostüme ihr Aussehen verändern. Kopieren Sie das vorhandene Kostüm. Klicken Sie dazu in der Liste der Kostüme mit der rechten Maustaste darauf und wählen Sie im Kontextmenü **Duplizieren**. Benennen Sie die beiden Kostüme um: **aus** für die ausgeschaltete LED und **ein** für die eingeschaltete.

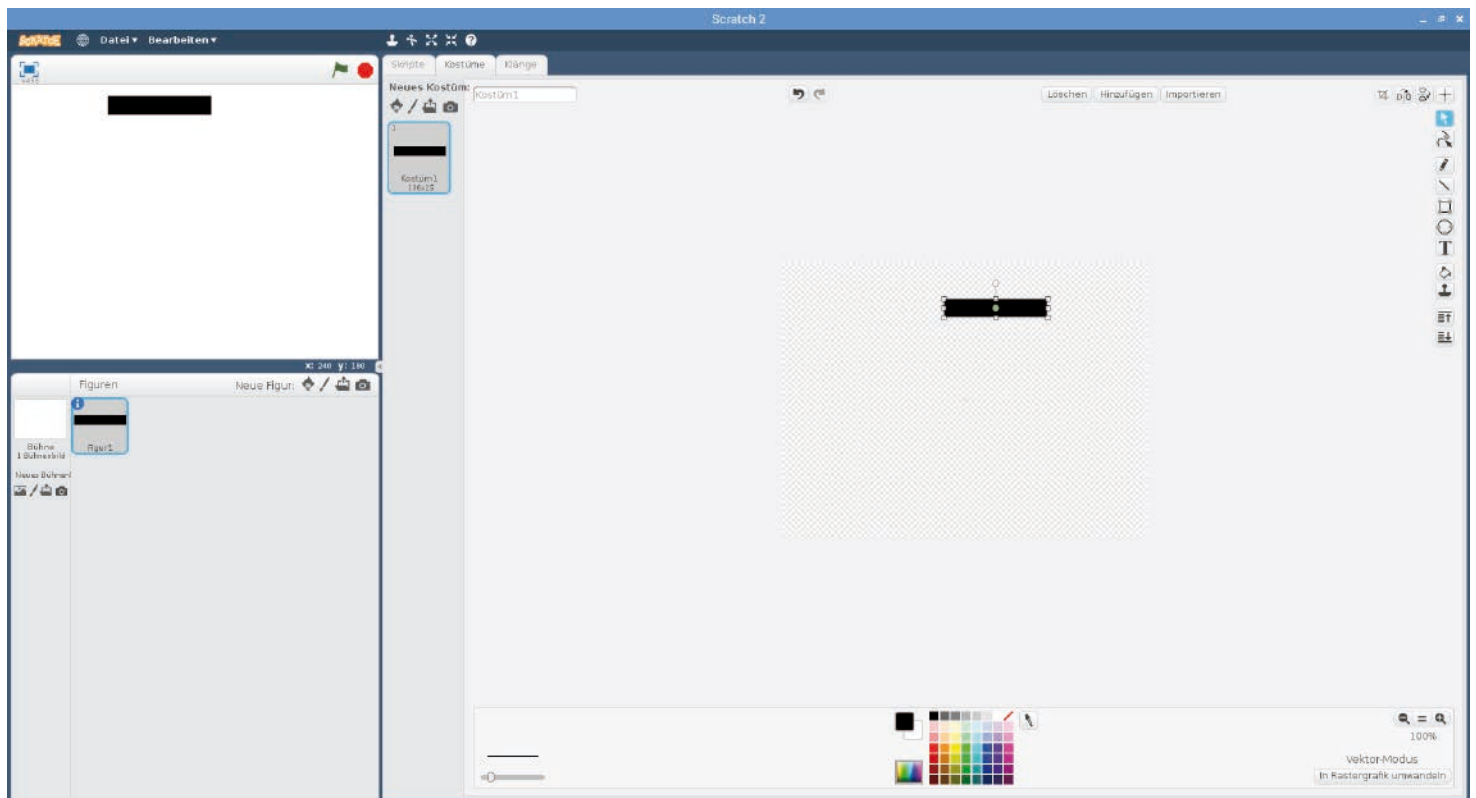


fritzing

Alle sieben Segmente der Sieben-Segment-Anzeige sind angeschlossen. Die Ziffern sind mit Masse verbunden.

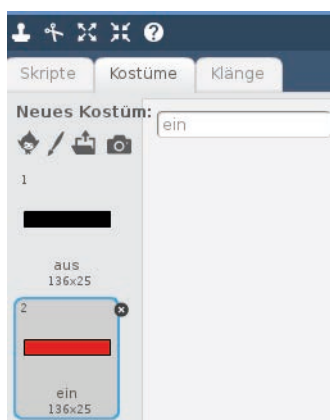


Das leere Figurenfenster in Scratch 2



Das erste Segment im Malprogramm von Scratch 2

Bearbeiten Sie das Kostüm **ein** mit dem Malprogramm. Füllen Sie die farbige Fläche mit dem Farbeimer-Symbol mit Rot, um die eingeschaltete LED darzustellen.



Die beiden Kostüme der ersten Figur.



Kostüm für eine ausgeschaltete LED im Malprogramm

Benennen Sie die Figur in der Figurenpalette links unten in **A** um, da sie das A-Segment der Sieben-Segment-Anzeige darstellt. Klicken Sie dazu auf das blaue Info-Symbol. Dann erscheint ein Fenster zum Umbenennen der aktuellen Figur.

Als Nächstes erstellen Sie das Programm für diese Figur. Da die Programme aller sieben Segmente sehr ähnlich sind, können Sie sie gleich mit den Figuren duplizieren und später nur noch leicht anpassen.

Jede Figur bekommt eigene Skriptblöcke. Wenn das grüne Fähnchen angeklickt wird, wird der GPIO-Pin 21 als Ausgang definiert und ausgeschaltet. Hängen Sie dazu an den Block **Wenn grünes Fähnchen angeklickt** einen Block **set gpio... to ...** an.

Wird die Figur selbst angeklickt, soll dieses Segment sowohl auf dem Bildschirm als auch auf der Sieben-Segment-Anzeige ein- und beim nächsten Anklicken wieder ausgeschaltet werden.

Verwenden Sie dazu den Block **Wenn ich angeklickt werde** von der Registerkarte **Steuerung**.

In diesem Fall wird zuerst das Kostüm auf das nächste gewechselt. So schaltet sich das Segment auf dem Bildschirm bei jedem Klick um. Hängen Sie dazu einen Block **nächstes Kostüm Aussehen** an.

Danach prüft eine **falls ... dann ... sonst** Abfrage, welches Kostüm gerade angezeigt ist. Die Kostüme haben die Nummern 1 und 2. Je nach aktivem Kostüm wird das Segment am GPIO-Pin 21 ein- oder ausgeschaltet.

Nachdem alle Skriptblöcke fertig gestellt sind, duplizieren Sie die Figur **A** mit einem Rechtsklick im Figurenfenster. Benennen Sie die neue Figur in **B** um.



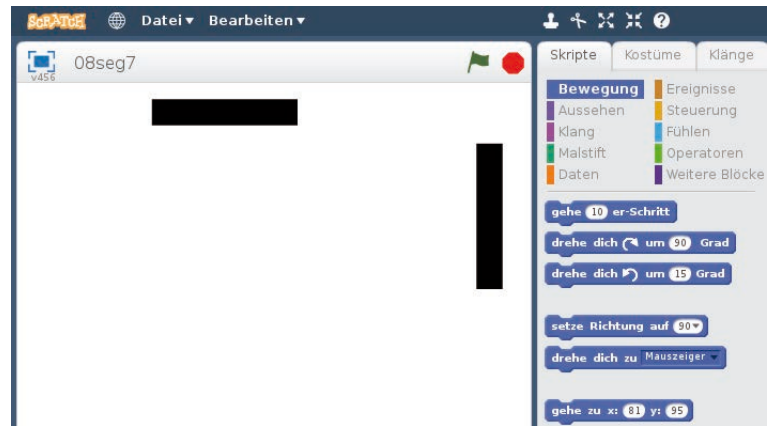
Scratch-Blöcke für das Segment A der Sieben-Segment-Anzeige

Drehen Sie die neue Figur um 90°, da sie das senkrechte Segment B darstellt. Drehen Sie dazu nicht die Kostüme, sondern die ganze Figur. Schalten Sie auf die Registerkarte **Skripte** um und tragen Sie dort auf der Blockpalette **Bewegung** im Block **Drehe dich im Uhrzeigersinn um ... Grad** die 90 ein. Sie brauchen den Block nicht in das Programm zu ziehen. Klicken sie einfach doppelt darauf, wird er ausgeführt und die Figur gedreht. Schieben Sie die Figur anschließend an die passende Stelle neben das Segment A.

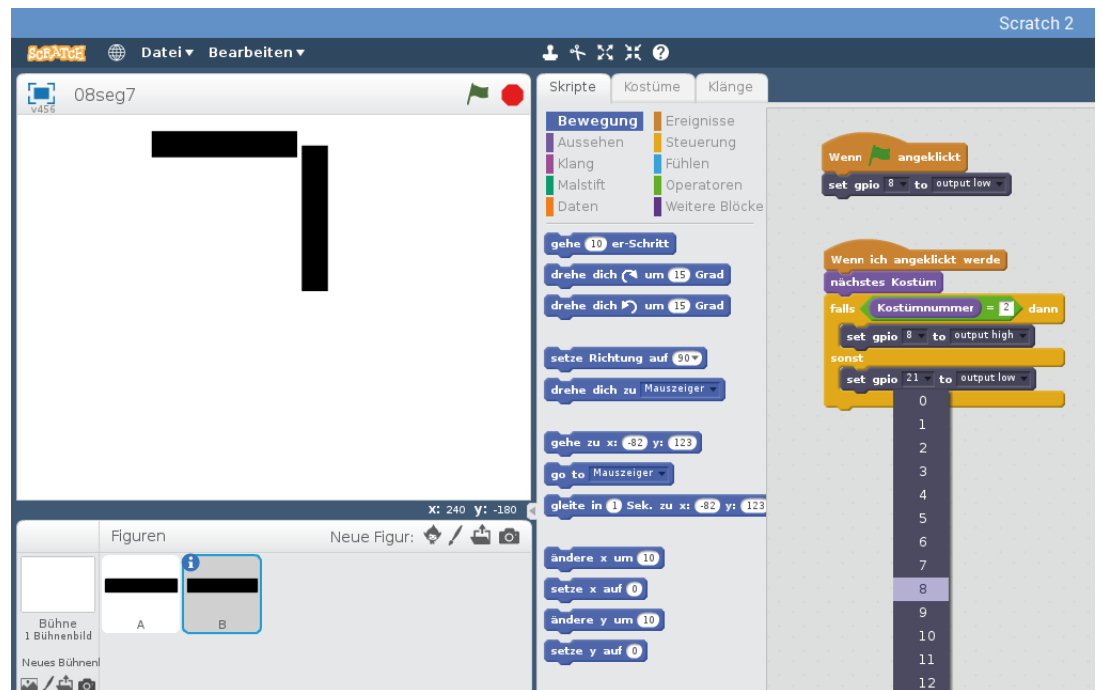
Wählen Sie in allen drei **set gpio... to ...**-Blöcken den GPIO-Pin 8, an dem das Segment B der Sieben-Segment-Anzeige angeschlossen ist.

Duplizieren Sie auf die gleiche Weise die anderen Figuren, benennen Sie sie um, schieben Sie sie an die passenden Positionen und ändern Sie die GPIO-Pins.

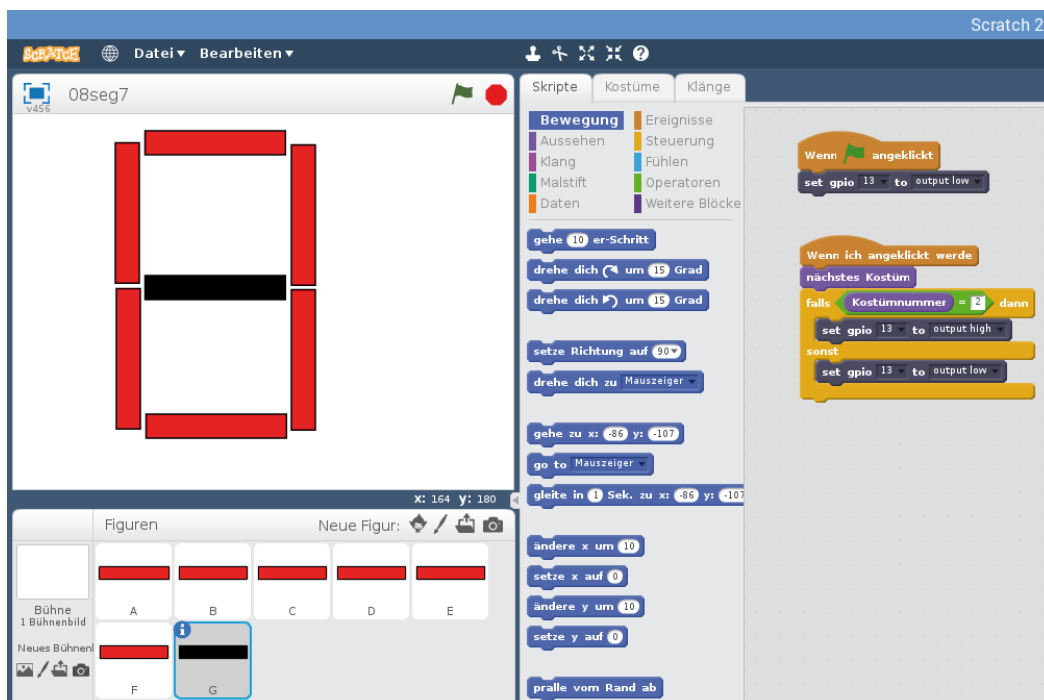
Starten Sie dann das Programm mit einem Klick auf das grüne Fähnchen. Klicken Sie auf eine der Figuren, ändert sie ihre Farbe und die zugehörige LED der Sieben-Segment-Anzeige wird ein- oder ausgeschaltet.



Segment duplizieren und drehen



GPIO-Pins für Segment B auswählen



Das fertige Programm in Aktion

9. Tag

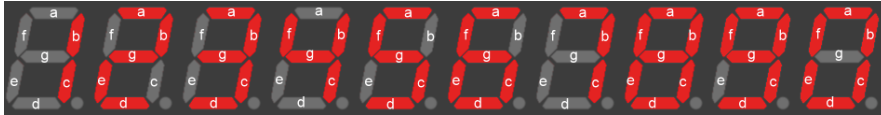
Heute im Adventskalender

• 1 GPIO-Verbindungskabel

Zahlen auf der Sieben-Segment-Anzeige

Der eigentliche Sinn und Zweck einer Sieben-Segment-Anzeige ist in den meisten Fällen keine grafische Spielerei, sondern die Darstellung von Zahlen. Das Programm des 9. Tags zeigt, wie das funktioniert.

Der Schaltungsaufbau ist der gleiche wie am 8. Tag, nur verwenden wir diesmal Python statt Scratch.



Die zehn Ziffernbilder einer Sieben-Segment-Anzeige.

Das Programm

Das Programm `09zahl.py` zeigt einen Zähler, der alle Ziffern der Reihe nach auf der Anzeige durchzählt. Alle vier Ziffern der Anzeige zeigen immer die gleiche Zahl.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

seg={'a':21, 'b':8, 'c':11, 'd':26, 'e':19, 'f':20, 'g':13}
for s in "abcdefg":
    GPIO.setup(seg[s], GPIO.OUT, initial=0)

zahl=[
    "abcdef", #0
    "bc", #1
    "abdeg", #2
    "abcdg", #3
    "bcfg", #4
    "acdfg", #5
    "acdefg", #6
    "abc", #7
    "abcdefg", #8
    "abcdfg", #9
]

print("Strg+C beendet das Programm")
try:
    while True:
        for i in range(10):
            for s in zahl[i]:
                GPIO.output(seg[s], 1)
            time.sleep(0.5)
            for s in "abcdefg":
                GPIO.output(seg[s], 0)

except KeyboardInterrupt:
    GPIO.cleanup()
```

So funktioniert es

Auch in diesem Programm werden die GPIO-Pins für die sieben Segmente über ein dictionary initialisiert.

```
zahl=[
    "abcdef", #0
    "bc", #1
    "abdeg", #2
    "abcdg", #3

    "bcfg", #4
    "acdfg", #5
    "acdefg", #6
    "abc", #7
    "abcdefg", #8
    "abcdfg", #9
]
```

Die Liste `zahl` legt die einzuschaltenden Segmente für die einzelnen Ziffern von 0 bis 9 fest. Es handelt sich um eine normale Liste, deren Elemente nur der Übersichtlichkeit halber untereinander geschrieben sind. Die Ziffern selbst sind kein Bestandteil der Liste. Sie werden - wie alles, was hinter einem `#` in einer Zeile steht - von Python als Kommentar betrachtet und bei der Programmausführung ignoriert.

Das Programm läuft wieder in einer Endlosschleife, die vom Benutzer mit der Tastenkombination `[Strg]+[C]` jederzeit beendet werden kann.

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

Eine Schleife zählt fortlaufend von 0 bis 9 durch, um nacheinander die Ziffern anzuzeigen.

```
    for s in zahl[i]:
        GPIO.output(seg[s], 1)
```

Bei jedem Durchlauf der äußeren Schleife arbeitet eine innere Schleife die in der Liste `zahl` für diese Ziffer eingetragene Zeichenkette ab, setzt die entsprechenden Segmente der Anzeige auf `1` und schaltet sie damit ein.

```
        time.sleep(0.5)
    for s in "abcdefg":
        GPIO.output(seg[s], 0)
```

Nachdem die Schleife für eine Ziffer durchgelaufen ist, bleibt diese 0,5 Sekunden lang angezeigt, danach werden alle sieben Segmente unabhängig von ihrem aktuellen Status auf `0` gesetzt und damit ausgeschaltet. Danach startet die Darstellung der nächsten Zahl.

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

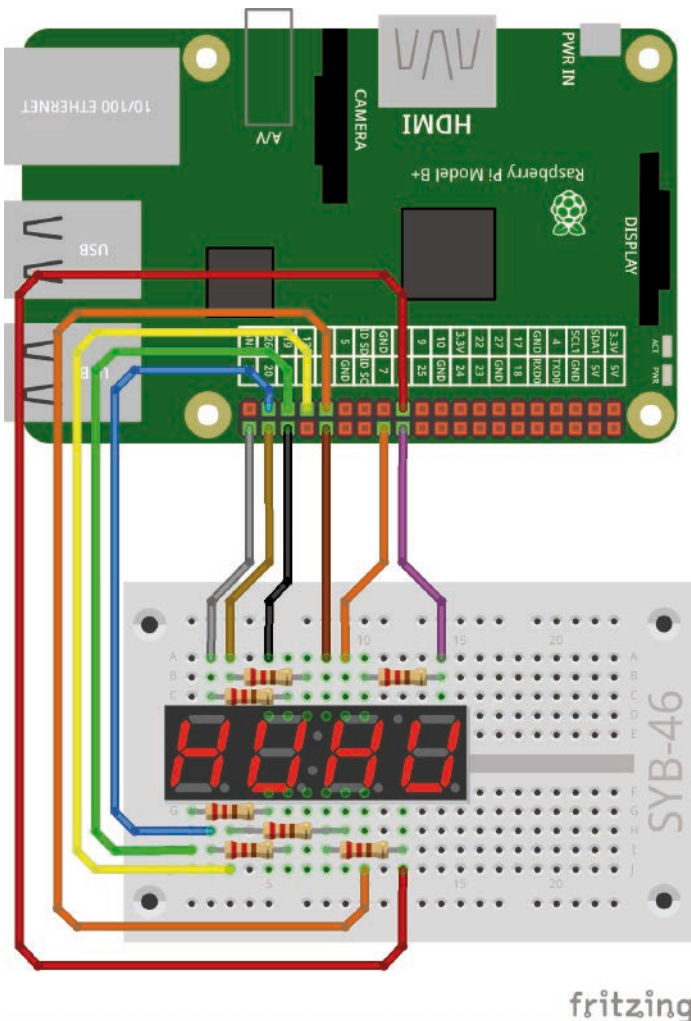
Drückt der Benutzer die Tastenkombination `[Strg]+[C]`, wird wie in vorherigen Experimenten die Endlosschleife beendet, und die GPIO-Ports werden geschlossen.

10. Tag

10. Tag

Heute im Adventskalender

- 1 GPIO-Verbindungskabel



Alle Sieben-Segmente der Sieben-Segment-Anzeige sind angeschlossen.
Die Ziffern sind einzeln mit GPIO-Pins verbunden.

Mehrstellige Zahlen auf der Sieben-Segment-Anzeige

Das vorige Experiment nutzte alle vier Ziffern der Sieben-Segment-Anzeige gleichzeitig. Wesentlich interessanter ist es, alle vier Ziffern eigenständig zu verwenden, um verschiedene Ziffern darzustellen.

Bauteile: 1 Steckbrett SYB-46, 1 Sieben-Segment-Anzeige, 7 220-Ohm-Widerstände (rot-rot-braun), 11 GPIO-Verbindungskabel

Anschlussschema der Sieben-Segment-Anzeige

Die folgende Tabelle zeigt, welche Pins der Sieben-Segment-Anzeige mit welchen GPIO-Pins verbunden sind. Der grau hinterlegte Pin wird in diesem Programm nicht verwendet.

Pin Sieben-Segment-Anzeige	Steckbrett	Segment / Ziffer	GPIO-Pin
1	5 F-J	E	19
2	6 F-J	D	26
3	7 F-J	DP	5
4	8 F-J	C	11
5	9 F-J	G	13
6	10 F-J	4	6
7	10 A-E	B	8
8	9 A-E	3	7
9	8 A-E	2	12
10	7 A-E	F	20
11	6 A-E	A	21
12	5 A-E	1	16

Der Trick mit dem Nachleuchten

Da auf jedem Segment immer für alle Ziffern das gleiche Signal anliegt, können einzelne Ziffern zwar ein- und ausgeschaltet werden, nicht aber verschiedene Zahlen anzeigen. Um auf jeder Ziffer der Anzeige trotzdem eine andere Zahl darzustellen, muss man sich eines Tricks bedienen. Eine Lichtquelle wird vom menschlichen Auge noch eine kurze Zeit als leuchtend wahrgenommen, obwohl sie eigentlich schon ausgeschaltet ist.

Das sogenannte Zeitmultiplexverfahren schaltet ganz schnell von einer zur nächsten Ziffer um und zeigt gleichzeitig bei jedem Wechsel eine andere Zahl an. Der Trick dabei ist, das richtige Zeitintervall herauszufinden. Wechseln die Zahlen zu schnell, scheinen sie für die menschliche Wahrnehmung zu verschwimmen, wechseln sie zu langsam, ist ein deutliches Flackern zu sehen.

Viele derartige Experimente, die man im Internet findet, verwenden zusätzlich einen ATmega-Mikrocontroller, wie ihn zum Beispiel auch der Arduino nutzt, oder einen Schieberegisterbaustein für die Ansteuerung der Sieben-Segment-Anzeige. Es gibt mittlerweile auch Sieben-Segment-Anzeigen, bei denen eine solche Elektronik bereits eingebaut ist. Das Programm des 10. Tags zeigt, dass eine Steuerung per Zeitmultiplex auch ohne zusätzliche Elektronik allein mit dem Raspberry Pi möglich ist.

Das Programm

Das Programm `10zahl.py` stellt die abgebildete Zahlenfolge 1234 auf der Sieben-Segment-Anzeige dar.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

seg={'a':21, 'b':8, 'c':11, 'd':26, 'e':19, 'f':20, 'g':13}
for s in "abcdefg":
    GPIO.setup(seg[s], GPIO.OUT, initial=0)
```



```

zif=[16, 12, 7, 6]
for z in zif:
    GPIO.setup(z, GPIO.OUT, initial=1)

zahl=[
    "abcdef", #0
    "bc",     #1
    "abdeg",  #2
    "abcdg",  #3
    "bcfg",   #4
    "acdfg",  #5
    "acdefg", #6
    "abc",    #7
    "abcdefg", #8
    "abcdfg", #9
    ]

print("Strg+C beendet das Programm")
try:
    while True:
        for i in range(4):
            for s in "abcdefg":
                GPIO.output(seg[s], 0)
            GPIO.output(zif[i], 0)
            for s in zahl[i+1]:
                GPIO.output(seg[s], 1)
            time.sleep(0.001)
            GPIO.output(zif[i], 1)

except KeyboardInterrupt:
    GPIO.cleanup()

```

So funktioniert es

Die Initialisierung der GPIO-Ports, die Listenvariablen für die LEDs und die Darstellung der Zahlen wurden aus dem letzten Experiment übernommen. Die Hauptschleife, die die Zahlen anzeigt, ist neu.

```
for i in range(4):
```

Innerhalb der Endlosschleife läuft eine weitere Schleife viermal durch, um schnell nacheinander die vier Ziffern darzustellen.

```
for s in "abcdefg":
    GPIO.output(seg[s], 0)
```

Eine Schleife setzt die Anoden aller sieben Segmente auf 0, schaltet sie also aus. In diesem Moment sind alle LEDs aus.

```
GPIO.output(zif[i], 0)
```

Die Kathode der aktuellen Ziffer wird auf 0 gesetzt, damit diese Ziffer danach die entsprechende Zahl anzeigen kann.

```
for s in zahl[i+1]:
    GPIO.output(seg[s], 1)
```

Die nächste Schleife schaltet alle Segmente der anzuzeigenden Zahl auf 1 und damit ein. Diese Zahl ist um eins höher als der Schleifenzähler, der von 0 bis 3 zählt. Es sollen die Ziffern 1 bis 4 angezeigt werden.

```
time.sleep(0.001)
```

Diese Zeile lässt das Programm 1 Millisekunde warten. Genauso lange leuchten die LEDs einer Ziffer wirklich, bevor zur nächsten Ziffer gewechselt wird. Experimentieren Sie mit unterschiedlichen Wartezeiten, um den Zeitmultiplex so einzustellen, dass die Anzeige möglichst wenig flackert, aber auch nicht verschwimmt.

```
GPIO.output(zif[i], 1)
```

Am Ende der Schleife wird die Kathode der aktuellen Ziffer auf 1 gesetzt und damit ausgeschaltet. Im nächsten Schleifendurchlauf wird die nächste Ziffer angezeigt.

11. Tag

Heute im Adventskalender

· 1 GPIO-Verbindungskabel

Der Schaltungsaufbau ist der gleiche wie am 10. Tag.

Beliebige Zahlen anzeigen

Im letzten Programm war die Zahl 1234 fest im Programm eingetragen. Das nächste Programm bietet die Möglichkeit, beliebige Zahlen einzugeben und darzustellen. Dabei werden auch Möglichkeiten gezeigt, Fehler bei der Benutzereingabe zu korrigieren.

Das Programm

Das Programm `11zahl.py` erfordert eine vierstellige Zahl, was aber noch lange nicht heißt, dass der Benutzer sie auch wirklich eingibt. Daher muss das Programm sämtliche Fehleingaben abfangen oder zu gültigen Zahlen umwandeln. Wenn der Benutzer zum Beispiel versehentlich oder auch absichtlich eine längere oder kürzere Zahl oder sogar Buchstaben und Sonderzeichen eingibt, darf es zu keinem fehlerbedingten Programmabbruch kommen.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

seg={'a':21, 'b':8, 'c':11, 'd':26, 'e':19, 'f':20, 'g':13}
for s in "abcdefg":
    GPIO.setup(seg[s], GPIO.OUT, initial=0)

zif=[16, 12, 7, 6]
for z in zif:
    GPIO.setup(z, GPIO.OUT, initial=1)

zahl=[
    "abcdef", #0
    "bc", #1
    "abdeg", #2
    "abcdg", #3
    "bcfg", #4
    "acdfg", #5
    "acdefg", #6
    "abc", #7
    "abcdefg", #8
    "abcdfg", #9
    ]

z = [0,0,0,0]
print("Strg+C beendet das Programm")

def za():
    for i in range(4):
        for s in "abcdefg":
            GPIO.output(seg[s], 0)
        GPIO.output(zif[i], 0)
        for s in zahl[z[i]]:
            GPIO.output(seg[s], 1)
        time.sleep(0.001)
        GPIO.output(zif[i], 1)

try:
    while True:
        s = input("Bitte vierstellige Zahl eingeben:")
        s = s.zfill(4)
        for i in range(4):
```

```

    if s[i].isdigit():
        z[i] = int(s[i])
    else:
        z[i] = 0
    sek = time.time()
    while time.time() <= sek + 2:
        za()

```

```

except KeyboardInterrupt:
    GPIO.cleanup()

```

So funktioniert es

Große Teile des Programms sind vom 10. Tag bekannt. Alles, was die Initialisierung der GPIO-Ports und die Liste zur Darstellung der Ziffern auf den sieben Segmenten der Anzeige betrifft, ist aus dem letzten Programm übernommen.

```
z = [0,0,0,0]
```

Die Liste `z[]` enthält im Verlauf des Programms vier Ziffern, die angezeigt werden sollen. Diese Liste wird am Anfang einmal initialisiert und mit vier Nullen gefüllt.

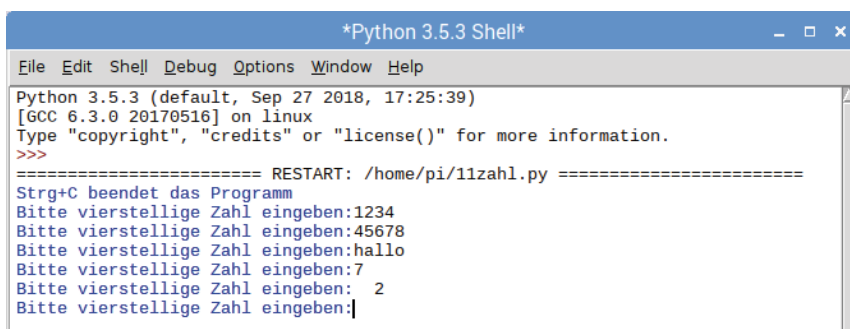
```

def za():
    for i in range(4):
        for s in "abcdefg":
            GPIO.output(seg[s], 0)
        GPIO.output(zif[i], 0)
        for s in zahl[z[i]]:
            GPIO.output(seg[s], 1)
        time.sleep(0.001)
        GPIO.output(zif[i], 1)

```

Die Zahlenanzeige ist jetzt in eine Funktion `za()` ausgelagert, die möglichst allgemein gehalten ist und deshalb auch für andere Programme verwendet werden kann. Sie zeigt nicht mehr, wie im letzten Programm, nacheinander die Ziffern 1 bis 4 an, sondern die vier Ziffern aus der Liste `z[]`.

Der Hauptteil des Programms ist völlig neu und läuft wieder als Endlosschleife. Der Benutzer wird aufgefordert, eine vierstellige Zahl einzugeben. Was auch immer der Benutzer tatsächlich eingibt, wird in eine gültige vierstellige Zahl umgewandelt, die danach etwa 2 Sekunden lang angezeigt wird. Anschließend erscheint erneut die Aufforderung zur Eingabe einer Zahl.



```

*Python 3.5.3 Shell*
File Edit Shell Debug Options Window Help
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: /home/pi/11zahl.py =====
Strg+C beendet das Programm
Bitte vierstellige Zahl eingeben:1234
Bitte vierstellige Zahl eingeben:45678
Bitte vierstellige Zahl eingeben:hallo
Bitte vierstellige Zahl eingeben:7
Bitte vierstellige Zahl eingeben: 2
Bitte vierstellige Zahl eingeben:|

```

Das Programm darf auch bei ungültigen Eingaben nicht einfach abbrechen.

```

try:
    while True:
        s = input("Bitte vierstellige Zahl eingeben:")

```

Die Endlosschleife wird in diesem Programm nicht zur Anzeige der Ziffern verwendet, sondern für die Benutzereingabe. Sie läuft also nicht viele Male pro Sekunde durch, sondern nur einmal für jede eingegebene Zahl.

Der Benutzer wird aufgefordert, eine vierstellige Zahl einzugeben. Die Funktion `input()` übernimmt die Eingabe im Klartext, ohne sie auszuwerten, und speichert sie als Zeichenkette in der Variablen `s`. Diese Zeichenkette kann beliebig lang sein und neben Ziffern beliebige andere Zeichen enthalten. Ungültige oder fehlende Zahlen sollen einfach als `0` angezeigt werden.

```
s = s.zfill(4)
```

Um das Problem abzufangen, dass der Benutzer möglicherweise weniger als vier Zeichen eingibt oder sogar einfach nur die [Enter]-Taste drückt, wird die Zeichenkette bis auf vier Stellen am Anfang mit Nullen aufgefüllt. Damit ist die Zeichenkette `s` immer mindestens vier Stellen lang.

```
for i in range(4):
```

Jetzt beginnt eine Schleife, die für jede der vier Stellen der Anzeige eine gültige Ziffer ermittelt und in die Liste `z[]` schreibt. Dabei werden die ersten vier Zeichen der Zeichenkette `s` ausgewertet. Ab dem fünften Zeichen werden alle Zeichen ignoriert.

```
    if s[i].isdigit():
        z[i] = int(s[i])
```

Ist das aus der Zeichenkette gelesene Zeichen wirklich eine Ziffer, wird deren Zahlenwert an der entsprechenden Stelle in der Liste `z[]` abgespeichert. Um das zu prüfen, wird die Methode `isdigit()` verwendet, die für jede Zeichenkettenvariable automatisch verfügbar ist. Sie ergibt `True`, wenn alle Zeichen der Zeichenkette – die in diesem Fall nur aus einem Zeichen besteht – Ziffern sind.

Die Funktion `int()` konvertiert ein Zeichen oder eine Fließkommazahl in eine Ganzzahl. In diesem Fall wird das gerade aus der Zeichenkette gelesene Zeichen in die entsprechende Ziffer umgewandelt.

```
    else:
        z[i] = 0
```

Im anderen Fall, wenn nämlich an der gerade gelesenen Stelle der Zeichenkette ein anderes Zeichen steht, weil der Benutzer sich bei der Eingabe nicht an die Regeln gehalten hat, wird an der entsprechenden Stelle in der Liste `z[]` eine `0` eingetragen.

Die auf diese Weise festgelegte Zahl, deren Ziffern sich jetzt einzeln in vier Elementen der Liste `z[]` befinden, soll 2 Sekunden lang angezeigt werden. Danach beginnt die Schleife von Neuem und fordert den Benutzer zur Eingabe auf.

Da zur Darstellung der Zahl auf der vierstelligen Sieben-Segment-Anzeige permanent in schneller Folge eine eigene Programmfunktion laufen muss, kann für die Wartezeit nicht einfach `time.sleep()` verwendet werden. Eine `while`-Schleife lässt 2 Sekunden lang immer wieder die Funktion `za()` zur Zahlenanzeige laufen und beendet danach die Hauptschleife.

```
    sek = time.time()
    while time.time() <= sek + 2:
        za()
```

In der Variablen `sek` wird die aktuelle Zeit in Sekunden gespeichert, die die Funktion `time.time()` jederzeit auf die Hundertstelsekunde genau ausgibt.

Zur Berechnung von Zeitangaben in Programmen bezieht man sich am besten immer auf die Systemzeit des Raspberry Pi und versucht nicht, zum Beispiel die Dauer einer Schleife selbst zu berechnen und daraus Zeiträume zu ermitteln. Durch Hintergrundprozesse eines Multitasking-Betriebssystems können diese Ergebnisse schnell verfälscht werden. Die Differenz zwischen zwei absoluten Zeitangaben ist dagegen eindeutig.

Die `while`-Schleife fragt ab, ob seit dem Speichern der Zeit zwei oder mehr Sekunden vergangen sind. Solange dies nicht der Fall ist, wird immer wieder die Funktion `za()` zur Zahlenanzeige aufgerufen.

Nach den 2 Sekunden endet diese Schleife, und damit endet auch die übergeordnete `while`-Schleife. Die Anzeige geht aus, und der Benutzer wird aufgefordert, eine neue Zahl einzugeben.

12. Tag

12. Tag

Heute im Adventskalender

• 1 GPIO-Verbindungskabel

Der Schaltungsaufbau ist der gleiche wie an den letzten Tagen.

Countdown bis Weihnachten

Das Programm des 12. Tags zeigt auf der Sieben-Segment-Anzeige an, wie viele Tage es noch bis Weihnachten sind.

Das Programm

Das Programm `12countdown.py` verwendet ein paar neue Python-Elemente zur Datumsberechnung.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time, datetime

GPIO.setmode(GPIO.BCM)

seg={'a':21, 'b':8, 'c':11, 'd':26, 'e':19, 'f':20,
'g':13}
for s in "abcdefg":
    GPIO.setup(seg[s], GPIO.OUT, initial=0)

zif=[16, 12, 7, 6]
for z in zif:
    GPIO.setup(z, GPIO.OUT, initial=1)

zahl=[
    "abcdef", #0
    "bc", #1
    "abdeg", #2
    "abcdg", #3
    "bcfg", #4
    "acdfg", #5
    "acdefg", #6
    "abc", #7
    "abcdefg", #8

    "abcdfg", #9
    ]

z=[0,0,0,0]
t=datetime.date(2019, 12, 24) - datetime.date.today()
z[0]=int(t.days/1000)
z[1]=int(t.days%1000/100)
z[2]=int(t.days%100/10)
z[3]=int(t.days%10)
print(z)
print("Strg+C beendet das Programm")

try:
    while True:
        for i in range(4):
            for s in "abcdefg":
                GPIO.output(seg[s], 0)
            GPIO.output(zif[i], 0)
            for s in zahl[z[i]]:
                GPIO.output(seg[s], 1)
            time.sleep(0.001)
            GPIO.output(zif[i], 1)
except KeyboardInterrupt:
    GPIO.cleanup()
```

So funktioniert es

Die Initialisierung der GPIO-Pins und die Liste zur Darstellung der Ziffern auf den sieben Segmenten der Anzeige werden aus dem letzten Programm übernommen.

```
import time, datetime
```

Am Anfang wird zusätzlich die Bibliothek `datetime` für Datumsberechnungen importiert.

```
t=datetime.date(2019, 12, 24) - datetime.date.today()
```

Die Funktion `datetime.date()` erzeugt ein Datumsobjekt mit einem bestimmten Datum. Diese Zeile errechnet die Differenz zweier Datumsobjekte, den Zeitraum zwischen dem 24.12.2019 und dem heutigen Tag. Das Objekt mit dem Ergebnis wird in der Variable `t` gespeichert.

```
z[0]=int(t.days/1000)
z[1]=int(t.days%1000/100)
z[2]=int(t.days%100/10)
z[3]=int(t.days%10)
```

Danach werden die vier anzuzeigenden Ziffern in den vier Elementen der Liste `z[]` gespeichert. `t.days` liefert die Tage in dem Datumsobjekt. Die Funktion `int()` speichert nur den ganzzahligen Teil der Berechnungsergebnisse.

- Die erste Ziffer `z[0]` ergibt sich durch Division der Tage durch 1000.
- Die zweite Ziffer `z[1]` wird mit dem Modulo-Operator `%` berechnet. Er ergibt den unteilbaren Rest bei einer Ganzzahldivision. Das Ergebnis wird durch 100 geteilt und ergibt die zweite Ziffer der anzuzeigenden Zahl.
- Auf die gleiche Weise wird die dritte Ziffer `z[2]` errechnet.
- Die vierte Ziffer `z[3]` ist der unteilbare Rest bei der Division durch 10.

```
print(z)
```

Diese Zeile zeigt die vier Ziffern in der Python-Shell an. Sie ist zur Berechnung nicht nötig und kann auch weggelassen werden. Anschließend wird die Zahl mit Hilfe der bereits bekannten Endlosschleife auf der Sieben-Segment-Anzeige dargestellt.

13. Tag

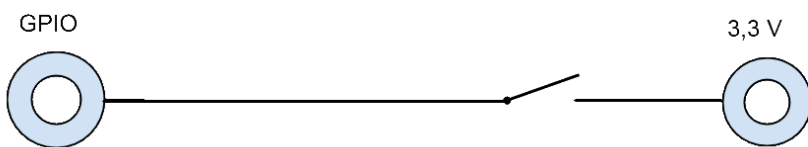
Heute im Adventskalender

- 1 Taster
- 1 Widerstand 10 kOhm

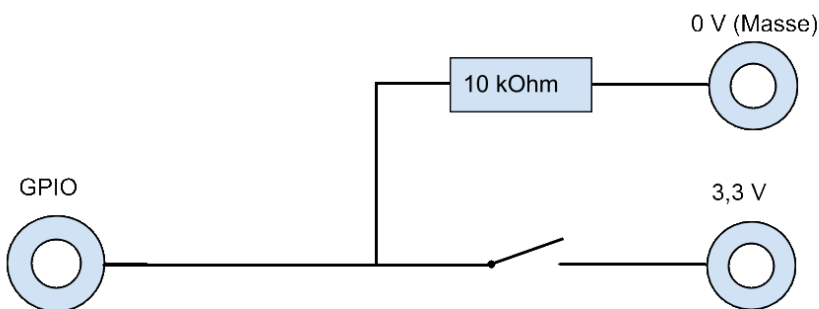
Das Anschlusschema der Sieben-Segment-Anzeige ist das gleiche wie an den letzten Tagen.

Taster an einem GPIO-Pin anschließen

GPIO-Pins können nicht nur Daten ausgeben, zum Beispiel 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.



Taster an einem GPIO-Eingang



Taster mit Pulldown-Widerstand an einem GPIO-Eingang

Liegt auf einem als Eingang definierten GPIO-Pin ein +3,3-V-Signal an, wird es als logisch `True` bzw. `1` ausgewertet. So können Sie also über einen Taster den jeweiligen GPIO-Pin mit dem +3,3-V-Anschluss des Raspberry Pi verbinden, was Sie aber bei älteren Raspberry-Pi-Modellen auf keinen Fall tun dürfen! Der GPIO-Pin würde dadurch überlastet. Beim Raspberry Pi 3 und Raspberry Pi 3 B+ sind Schutzwiderstände eingebaut und daher keine extern angeschlossenen Schutzwiderstände mehr nötig.

In den meisten Fällen funktioniert diese simple Schaltung bereits, allerdings hätte der GPIO-Pin 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-Pins bei geöffnetem Taster wieder

nach unten auf 0 V. Da der Widerstand sehr hoch ist, besteht, solange der Taster gedrückt ist, keine Kurzschlussgefahr. Im gedrückten Zustand des Tasters sind +3,3 V und die Masseleitung direkt über diesen Widerstand verbunden.

Weihnachts-Countdown mit Taster

Das Programm des 13. Tags zeigt, wie viele Tage es noch bis Weihnachten sind. Drückt man auf den Taster, startet ein Countdown, der bis 0 herunterzählt.

Bauteile: 1 Steckbrett SYB-46, 1 Sieben-Segment-Anzeige, 7 220-Ohm-Widerstände (rot-rot-braun), 1 10-kOhm-Widerstand (braun-schwarz-orange), 1 Taster, 14 GPIO-Verbindungskabel

Die in der Abbildung linke Kontaktleiste des Tasters ist mit dem GPIO-Pin 18 und über einen 10-kOhm-Pulldown-Widerstand (Braun-Schwarz-Orange) mit der Masseleitung auf der unteren Kontaktschiene des Steckbretts verbunden. Die in der Abbildung rechte Kontaktleiste des Tasters ist mit der +3,3-V-Leitung des Raspberry Pi verbunden.

Der Schaltungsaufbau für die Sieben-Segment-Anzeige ist der gleiche wie an den letzten Tagen.

Das Programm

Das Programm `13countdown.py` zeigt wieder die Anzahl der Tage bis Weihnachten an. Beim Druck auf den Taster läuft ein Countdown herunter bis zur 0.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time, datetime
```

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

```

seg={'a':21, 'b':8, 'c':11, 'd':26, 'e':19, 'f':20, 'g':13}
for s in "abcdefg":
    GPIO.setup(seg[s], GPIO.OUT, initial=0)

zif=[16, 12, 7, 6]
for z in zif:
    GPIO.setup(z, GPIO.OUT, initial=1)

zahl=[
    "abcdef", #0
    "bc", #1
    "abdeg", #2
    "abcdg", #3
    "bcfg", #4
    "acdfg", #5
    "acdefg", #6
    "abc", #7
    "abcdefg", #8
    "abcdfg", #9
    ]

taster=18
GPIO.setup(taster, GPIO.IN, GPIO.PUD_DOWN)

z=[0,0,0,0]

def za():
    for i in range(4):
        for s in "abcdefg":
            GPIO.output(seg[s], 0)
        GPIO.output(zif[i], 0)
        for s in zahl[z[i]]:
            GPIO.output(seg[s], 1)
        time.sleep(0.001)
        GPIO.output(zif[i], 1)

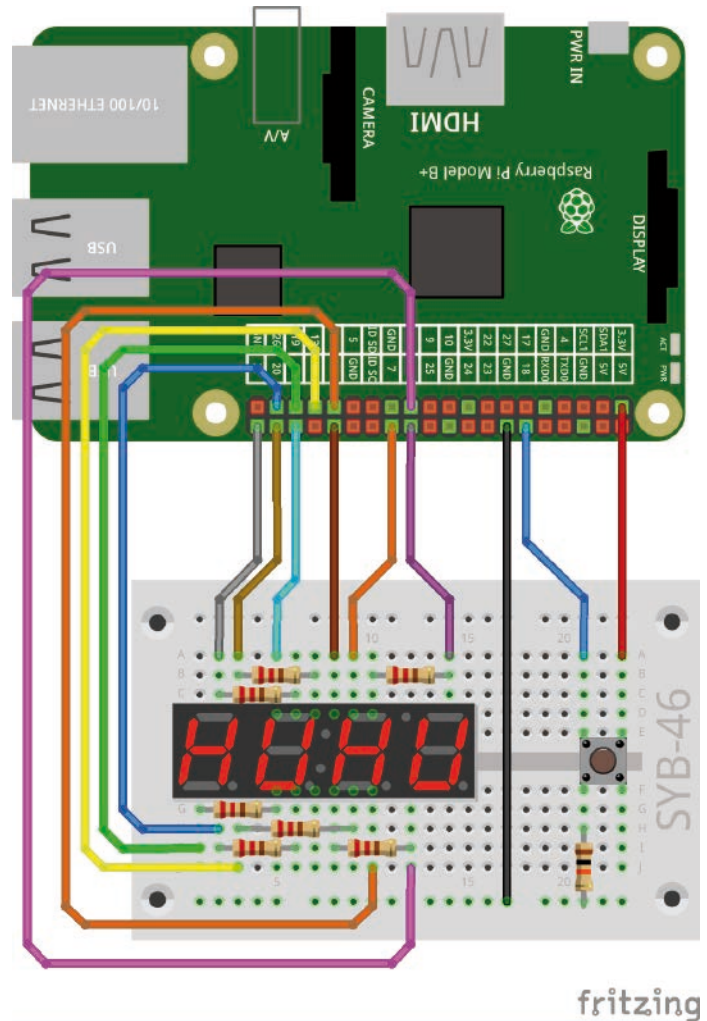
t=datetime.date(2019, 12, 24) - datetime.date.today()
x=t.days
z[0]=int(x/1000)
z[1]=int(x%1000/100)
z[2]=int(x%100/10)
z[3]=int(x%10)

print("Zum Start des Countdowns Taste drücken")
while GPIO.input(taster)==0:
    za()

while x>=0:
    z[0]=int(x/1000)
    z[1]=int(x%1000/100)
    z[2]=int(x%100/10)
    z[3]=int(x%10)
    for j in range(100):
        za()
    x-=1

GPIO.cleanup()

```



Taster mit Pulldown-Widerstand.

So funktioniert es

Der Taster am Pin 18 wird als GPIO-Eingang definiert. Die Pin-Nummer wird in der Variable `taster` gespeichert.

```
taster=18
GPIO.setup(taster, GPIO.IN, GPIO.PUD_DOWN)
```

Das Programm verwendet wieder die aus einem früheren Programm bekannte Funktion `za()` zur Anzeige einer vierstelligen Zahl auf der Sieben-Segment-Anzeige.

```
t=datetime.date(2019, 12, 24) - datetime.date.today()
x=t.days
z[0]=int(x/1000)
z[1]=int(x%1000/100)
z[2]=int(x%100/10)
z[3]=int(x%10)
```

Die Anzahl der Tage bis Weihnachten wird zunächst in einer neuen Variable `x` gespeichert und dann als einzelne Ziffern in den vier Feldern der Liste `z[]`.

```
while GPIO.input(taster)==0:
    za()
```

Eine Endlosschleife ruft, solange der Taster nicht gedrückt ist, immer wieder die Funktion `za()` auf, um diese Zahl anzuzeigen.

```
while x>=0:
    z[0]=int(x/1000)
    z[1]=int(x%1000/100)
    z[2]=int(x%100/10)
    z[3]=int(x%10)
    for j in range(100):
        za()
    x-=1
```

Wurde der Taster gedrückt, startet eine neue Schleife, die den Wert `x` in jedem Durchlauf um 1 herunterzählt. Dabei werden jedes Mal die Werte für die Liste `z[]` neu berechnet. Anschließend ruft eine innere Schleife die Funktion `za()` hundertmal hintereinander auf, was jede Zahl eine Zehntelsekunde lang leuchten lässt, bevor die nächste Zahl angezeigt wird.

Wird durch das Herunterzählen die 0 erreicht, erscheint sie noch auf der Anzeige. Danach werden die GPIO-Ports geschlossen und das Programm beendet.

14. Tag

Heute im Adventskalender

• 1 Taster

Das Anschlussschema der Sieben-Segment-Anzeige ist das gleiche wie an den letzten Tagen.

Taster ohne externe Pulldown-Widerstände anschließen

Aktuelle Raspberry-Pi-Modelle verfügen über eingebaute Pulldown-Widerstände, die softwareseitig ein- und ausgeschaltet werden können. Damit spart man sich den externen Widerstand. Der Taster kann direkt zwischen +3,3V und dem GPIO-Pin angeschlossen werden, wie das Experiment des 14. Tags erklärt.

Weihnachts-Countdown mit zwei Tastern

Auch das Programm des 14. Tags zeigt, wie viele Tage es noch bis Weihnachten sind. Drückt man auf den ersten Taster, startet ein Countdown, der bis 0 herunterzählt. Am Ende kann der Countdown mit dem gleichen Taster erneut gestartet werden. Der zweite Taster beendet das Programm. Er muss etwas länger gedrückt werden, da, je nachdem, wo das Programm gerade steht, mehrere Schleifen kurz hintereinander abgebrochen werden müssen.

Bauteile: 1 Steckbrett SYB-46, 1 Sieben-Segment-Anzeige, 7 220-Ohm-Widerstände (rot-rot-braun), 2 Taster, 14 GPIO-Verbindungskabel, 2 Drahtbrücken

Das Programm

Das Programm `14countdown.py` basiert auf dem Programm des 13. Tags und enthält noch weitere Schleifen und `break`-Anweisungen zum Abbruch der Schleifen.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time, datetime

GPIO.setmode(GPIO.BCM)

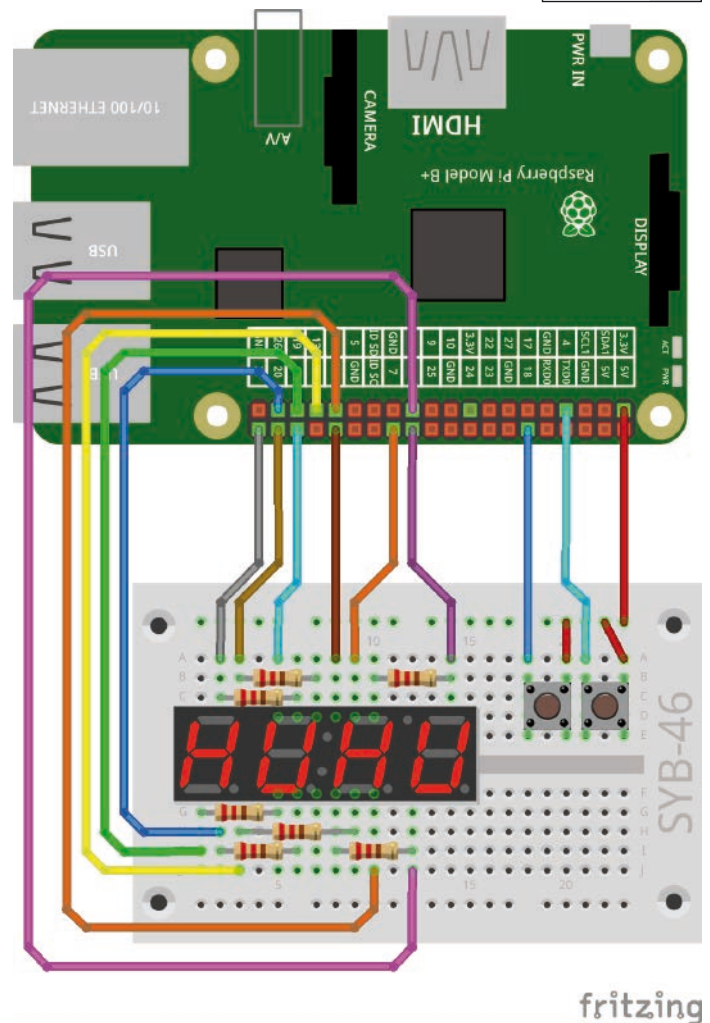
seg={'a':21, 'b':8, 'c':11, 'd':26, 'e':19, 'f':20, 'g':13}
for s in "abcdefg":
    GPIO.setup(seg[s], GPIO.OUT, initial=0)

zif=[16, 12, 7, 6]
for z in zif:
    GPIO.setup(z, GPIO.OUT, initial=1)

zahl=[
    "abcdef", #0
    "bc", #1
    "abdeg", #2
    "abcdg", #3
    "bcfg", #4
    "acdfg", #5
    "acdefg", #6
    "abc", #7
    "abcdefg", #8
    "abcdfg", #9
]

taste1=18
GPIO.setup(taste1, GPIO.IN, GPIO.PUD_DOWN)
taste2=4
```

14. Tag



Zwei Taster ohne externe Pulldown-Widerstände

fritzing

```

GPIO.setup(taste2, GPIO.IN, GPIO.PUD_DOWN)

z=[0,0,0,0]

def za():
    for i in range(4):
        for s in "abcdefg":
            GPIO.output(seg[s], 0)
        GPIO.output(zif[i], 0)
        for s in zahl[z[i]]:
            GPIO.output(seg[s], 1)
        time.sleep(0.001)
        GPIO.output(zif[i], 1)

print("Zum Start des Countdowns und zum Neustart Taste 1 drücken")
print("Zum Beenden Taste 2 drücken")
while True:
    t=datetime.date(2019, 12, 24) - datetime.date.today()
    x=t.days
    z[0]=int(x/1000)
    z[1]=int(x%1000/100)
    z[2]=int(x%100/10)
    z[3]=int(x%10)
    if GPIO.input(taste2)==1:
        break

    while GPIO.input(taste1)==0:
        za()
        if GPIO.input(taste2)==1:
            break

    while x>=0:
        z[0]=int(x/1000)
        z[1]=int(x%1000/100)
        z[2]=int(x%100/10)
        z[3]=int(x%10)
        for j in range(100):
            za()
        x-=1
        if GPIO.input(taste2)==1:
            break

    while GPIO.input(taste1)==0:
        za()
        if GPIO.input(taste2)==1:
            break

GPIO.cleanup()

```

So funktioniert es

Die Initialisierung der GPIO-Pins und die Liste zur Darstellung der Ziffern auf den sieben Segmenten der Anzeige werden aus dem letzten Programm übernommen.

```
taste1=18
GPIO.setup(taste1, GPIO.IN, GPIO.PUD_DOWN)
taste2=4
GPIO.setup(taste2, GPIO.IN, GPIO.PUD_DOWN)
```

Anschließend werden die GPIO-Pins der beiden Taster initialisiert. Der Parameter `GPIO.PUD_DOWN` bei der Initialisierung der GPIO-Pins für die Taster schaltet den eingebauten Pulldown-Widerstand ein. Dadurch kann man sich den externen Pulldown-Widerstand sparen.

Das ganze Programm läuft jetzt in einer Endlosschleife, da der Countdown durch Drücken einer der Taster immer wieder wiederholt werden kann.

```
while True:
    t=datetime.date(2019, 12, 24) - datetime.date.today()
    x=t.days
    z[0]=int(x/1000)
    z[1]=int(x%1000/100)
    z[2]=int(x%100/10)
    z[3]=int(x%10)
    if GPIO.input(taste2)==1:
        break
```

Die Endlosschleife und auch alle untergeordneten Schleifen prüfen in jedem Durchlauf, ob die Taste 2 gedrückt ist. Ist dies der Fall, wird die Schleife über eine `break` Anweisung abgebrochen.

```
while GPIO.input(taste1)==0:
    za()
    if GPIO.input(taste2)==1:
        break
```

Nachdem die Anzahl der Tage bis Weihnachten berechnet wurde, wird diese Zahl so lange auf der Sieben-Segment-Anzeige angezeigt, wie die Taste 1 nicht gedrückt ist. Auch diese Schleife fragt die Taste 2 ab. Ist diese gedrückt, wird die Schleife abgebrochen. Danach folgt aber die nächste Schleife. Um alle Schleifen bis einschließlich der äußeren Endlosschleife abzubrechen, drücken Sie die Taste - ähnlich wie einen Reset-Taster - ein paar Millisekunden lang.

```
while x>=0:
    z[0]=int(x/1000)
    z[1]=int(x%1000/100)
    z[2]=int(x%100/10)
    z[3]=int(x%10)
    for j in range(100):
        za()
    x-=1
    if GPIO.input(taste2)==1:
        break
```

Wenn die Taste 1 gedrückt wurde, wird die Schleife regulär beendet und die nächste Schleife folgt, die den Countdown bis zur 0 herunterzählt. Auch sie kann jetzt durch Drücken der Taste 2 abgebrochen werden.

```
while GPIO.input(taste1)==0:
    za()
    if GPIO.input(taste2)==1:
        break
```

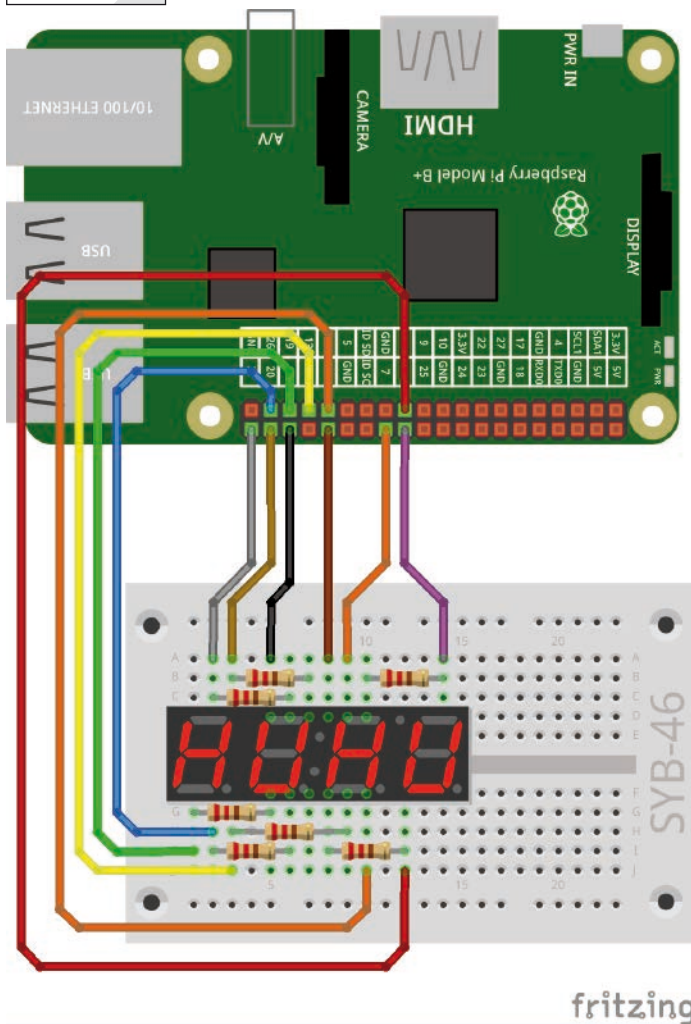
Zum Schluss stehen alle vier Ziffern auf 0. Eine weitere Schleife zeigt diese Zahl an, bis die Taste 1 gedrückt wird. In diesem Fall startet die Endlosschleife wieder von vorne mit der Berechnung der bis Weihnachten verbleibenden Tage.

15. Tag

15. Tag

Heute im Adventskalender

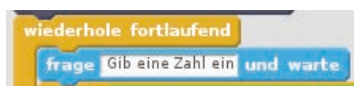
- 1 GPIO-Verbindungskabel



Alle sieben Segmente der Sieben-Segment-Anzeige angeschlossen.. Die Ziffern sind einzeln mit GPIO-Pins verbunden.



GPIO-Pins für die Ziffern initialisieren



Die Endlosschleife erwartet am Anfang eine Benutzereingabe.

Das Anschlussschema der Sieben-Segment-Anzeige ist das gleiche wie an den letzten Tagen.

Zahlen mit Scratch auf der Sieben-Segment-Anzeige

Das Programm des 15. Tags zeigt, wie sich mit Scratch Ziffern auf einer Sieben-Segment-Anzeige darstellen lassen. Scratch ist leider auch im Turbo-Modus nicht schnell genug, per Zeitmultiplex alle vier Ziffern der Anzeige zu steuern. Das Programm zeigt deshalb immer nur eine einzelne Ziffer an.

Bauteile: 1 Steckbrett SYB-46, 1 Sieben-Segment-Anzeige, 7 220-Ohm-Widerstände (rot-rot-braun), 11 GPIO-Verbindungskabel

Anschlussschema der Sieben-Segment-Anzeige

Die folgende Tabelle zeigt, welche Pins der 7-Segment-Anzeige mit welchen GPIO-Pins verbunden sind. Der grau hinterlegte Pin wird in diesem Programm nicht verwendet.

Pin Sieben-Segment-Anzeige	Steckbrett	Segment / Ziffer	GPIO-Pin
1	5 F-J	E	19
2	6 F-J	D	26
3	7 F-J	DP	5
4	8 F-J	C	11
5	9 F-J	G	13
6	10 F-J	4	6
7	10 A-E	B	8
8	9 A-E	3	7
9	8 A-E	2	12
10	7 A-E	F	20
11	6 A-E	A	21
12	5 A-E	1	16

Das Programm

Das Scratch 2 Programm 15zahl.sb2 erwartet vom Benutzer eine Ziffer, die dann auf der Sieben-Segment-Anzeige dargestellt wird.

Das Programm verwendet den gleichen Schaltungsaufbau wie die Programme der letzten Tage. Um die Kathoden der einzelnen Segmente nicht je nach Verwendung mit Masse oder +3,3V verdrahtet zu müssen, schaltet das Programm am Anfang die ersten drei Ziffern an den GPIO-Pins 16, 12 und 7 auf **High** und damit aus und die rechte Ziffer am GPIO-Pin 6 auf **Low** und damit ein.

Die Anoden der sieben Segmente brauchen nicht vorab initialisiert zu werden. Sie werden bei der Anzeige einer Ziffer automatisch initialisiert.

Als Nächstes startet eine Endlosschleife, die als Erstes eine Benutzereingabe erwartet.

Der Block **frage ... und warte** von der Blockpalette **Fühlen** blendet ein Eingabefeld auf der Scratch-Bühne ein. Die aktuelle Figur, im Beispiel die Katze; zeigt die Frage an. Das Programm läuft erst weiter, wenn der Benutzer etwas eingegeben hat.

Nacheinander fragen jetzt zehn Blöcke diese Eingabe ab. Dabei wird mit dem Block **Zeichen ... von ...** von der Blockpalette **Operatoren** in Kombination mit dem Block **... = ...** nur das erste Zeichen berücksichtigt, da immer nur ein Zeichen angezeigt werden kann.

Der Block **Antwort** aus der Blockpalette **Fühlen** enthält automatisch die eingegebene Zeichenkette.

Ist das erste Zeichen die Ziffer 0, werden die GPIO-Pins 21, 8, 11, 26, 19, 20 eingeschaltet und der GPIO-Pin 13 auf Low gesetzt und ausgeschaltet.



Zustand der GPIO-Pins, wenn der Benutzer eine 0 eingegeben hat.

Ähnliche Abfragen folgen für alle weiteren Ziffern. Dabei werden immer alle sieben Segmente geschaltet, nicht nur die benötigten, damit keine der letzten dargestellten Ziffern übrigbleibt.



Die Scratch-Katze fragt nach einer Zahl.

Blöcke duplizieren
 Beim Bau eines Scratch-Programms brauchen Sie ähnliche Blockkombinationen nicht jedes Mal neu anzulegen. Klicken Sie mit der rechten Maustaste auf den ersten Block, der dupliziert werden soll. Wählen Sie dann im Menü **Duplizieren**. Alle darunter hängenden Blöcke werden automatisch mit dupliziert. Die duplizierten Blöcke können dann wieder passender Stelle im Programm eingefügt werden.

Gibt der Benutzer keine Ziffer, sondern andere Zeichen ein, ändert sich die zuletzt auf der Anzeige dargestellte Zahl nicht.



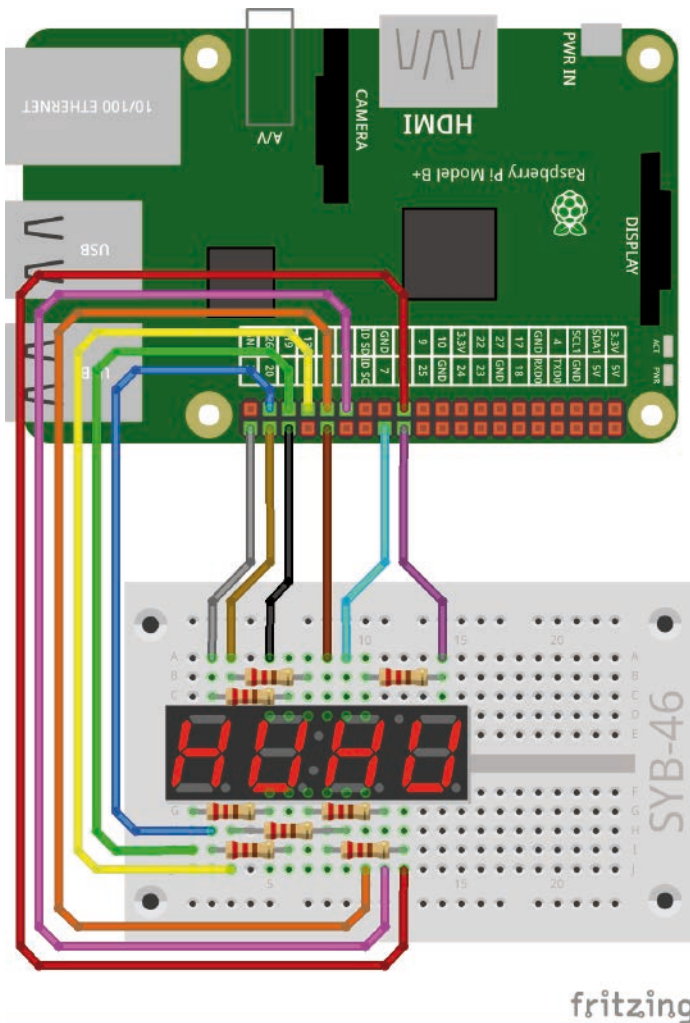
Das ganze Programm im Überblick

16. Tag

16. Tag

Heute im Adventskalender

• 1 Widerstand 220 Ohm



Alle sieben Segmente und der Dezimalpunkt der Sieben-Segment-Anzeige sind angeschlossen. Die Ziffern sind einzeln mit GPIO-Pins verbunden.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

seg={'a':21, 'b':8, 'c':11, 'd':26, 'e':19, 'f':20, 'g':13}
for s in "abcdefg":
    GPIO.setup(seg[s], GPIO.OUT, initial=0)

zif=[16, 12, 7, 6]
for z in zif:
    GPIO.setup(z, GPIO.OUT, initial=1)

dp = 5
GPIO.setup(dp, GPIO.OUT, initial=0)

zahl=[
    "abcdef", #0
    "bc", #1
    "abdeg", #2
    "abcdg", #3
```

Digitaluhr

Im Experiment des 16. Tags bauen wir eine funktionsfähige Digitaluhr, die die aktuelle Zeit auf der Sieben-Segment-Anzeige anzeigt. Die Digitaluhr kann automatisch auf dem Raspberry Pi laufen, ohne dass Monitor und Tastatur angeschlossen sind.

Der Raspberry Pi verfügt im Gegensatz zu einem PC über keine eingebaute batteriegepufferte Uhr, sondern holt seine Zeit immer von einem Zeitserver im Internet, der die aktuelle Zeit für jede Zeitzone der Welt jederzeit in höchster Präzision liefert.

Bauteile: 1 Steckbrett SYB-46, 1 Sieben-Segment-Anzeige, 8 220-Ohm-Widerstände (rot-rot-braun), 12 GPIO-Verbindungskabel

Anschlussschema der Sieben-Segment-Anzeige

Die folgende Tabelle zeigt, welche Pins der Sieben-Segment-Anzeige mit welchen GPIO-Pins verbunden sind. Diesmal werden alle Segmente, auch der Dezimalpunkt, verwendet.

Pin Sieben-Segment-Anzeige	Steckbrett	Segment / Ziffer	GPIO-Pin
1	5 F-J	E	19
2	6 F-J	D	26
3	7 F-J	DP	5
4	8 F-J	C	11
5	9 F-J	G	13
6	10 F-J	4	6
7	10 A-E	B	8
8	9 A-E	3	7
9	8 A-E	2	12
10	7 A-E	F	20
11	6 A-E	A	21
12	5 A-E	1	16

Das Programm

Das Programm `16uhr.py` basiert auf den Python-Programmen der letzten Tage mit ein paar Ergänzungen.

```

"bcfg", #4
"acdfg", #5
"acdefg", #6
"abc", #7
"abcdefg", #8
"abcdfg", #9
]

z = [0,0,0,0]
print("Strg+C beendet das Programm")

def za():
    for i in range(4):
        for s in "abcdefg":
            GPIO.output(seg[s], 0)
        GPIO.output(zif[i], 0)
        for s in zahl[z[i]]:
            GPIO.output(seg[s], 1)
        if i == 1:
            GPIO.output(dp, 1)
        else:
            GPIO.output(dp, 0)
        time.sleep(0.001)
        GPIO.output(zif[i], 1)

try:
    while True:
        zeit = time.localtime()
        h = zeit.tm_hour
        m = zeit.tm_min
        z[0] = int(h / 10)
        z[1] = h % 10
        z[2] = int(m / 10)
        z[3] = m % 10
        while time.localtime().tm_min == m:
            za()

except KeyboardInterrupt:
    GPIO.cleanup()

```

So funktioniert es

Große Teile des Programms werden Ihnen bekannt vorkommen. Das meiste, was die Initialisierung der GPIO-Ports und die Liste zur Darstellung der Ziffern auf den sieben Segmenten der Anzeige betrifft, wurde aus früheren Programmen übernommen. Zur Darstellung des Dezimalpunkts zwischen der zweiten und dritten Ziffer sind Ergänzungen nötig.

```

dp = 5
GPIO.setup(dp, GPIO.OUT, initial=0)

```

Nach der Definition der GPIO-Pins für Segmente und Ziffern wird der GPIO-Pin 5 für den Dezimalpunkt eingerichtet. Da es sich wie bei den Segmenten um eine gemeinsame Anode für alle vier Ziffern handelt, wird sie am Anfang auf 0 gesetzt, um die Dezimalpunkte auszuschalten.

Nachdem die Segmente einer Ziffer eingeschaltet werden, wird innerhalb der Funktion `za()`, wenn die zweite Ziffer gerade leuchtet, der Dezimalpunkt mit dazugeschaltet.

```

if i == 1:
    GPIO.output(dp, 1)
else:
    GPIO.output(dp, 0)

```

Steht der Schleifenzähler auf 1 - die erste Ziffer hat die Nummer 0 -, wird der GPIO-Ausgang für den Dezimalpunkt auf 1 gesetzt und damit eingeschaltet. Bei allen anderen Ziffern wird er auf 0 gesetzt und damit ausgeschaltet.

Danach beginnt eine Endlosschleife, die die aktuelle Uhrzeit ermittelt und die Ziffern in die Liste `z[]` schreibt, um sie mit der Funktion `za()` darzustellen.



Dezimalpunkt zur Trennung von Stunden und Minuten

```
try:
    while True:
        zeit = time.localtime()
```

In jedem Durchlauf wird, unabhängig davon, wie lange er dauert, 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.

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

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

Die folgenden Zeilen ermitteln aus der Zeitangabe die einzelnen Ziffern für die Anzeige und schreiben diese in die Liste `z[]`, aus der sie dann von der Funktion `za()` gelesen werden.

```
z[0] = int(h / 10)
```

Die erste Ziffer der Anzeige zeigt die Zehnerstelle der Stunden. Dazu wird die aktuelle Stundenangabe aus der Variablen `h` durch 10 geteilt und der Ganzzahlwert ermittelt. Das Ergebnis kann 0, 1 oder 2 sein.

```
z[1] = h % 10
```

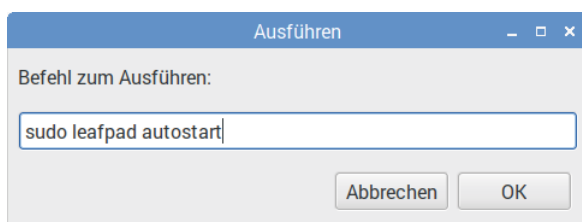
Die zweite Ziffer der Anzeige zeigt die Einerstelle der Stunden. Diese wird mit dem Modulo-Operator `%` errechnet. Modulo ermittelt den bei einer Ganzzahldivision entstehenden unteilbaren Rest. Bei einer Division durch 10 ergibt der Modulo-Operator immer die letzte Ziffer des Dividenden. Da immer Ganzzahlwerte errechnet werden, ist keine `int()` Funktion nötig.

```
z[2] = int(m / 10)
z[3] = m % 10
```

Auf die gleiche Weise werden an die dritte und die vierte Stelle der Liste `z[]` die beiden Ziffern der aktuellen Minutenangabe geschrieben.

```
while time.localtime().tm_min == m:
    za()
```

Solange die aktuelle Minute noch gleich der gespeicherten Minute ist, wird die Funktion `za()` zur Zahlenausgabe ständig wiederholt. Springt die Minutenangabe auf die nächste Minute um, beginnt die Hauptschleife von Neuem, ermittelt die Ziffern der aktuellen Uhrzeit und lässt diese dann wieder bis zur nächsten Minute anzeigen.



autostart Datei mit Superuserrechten bearbeiten

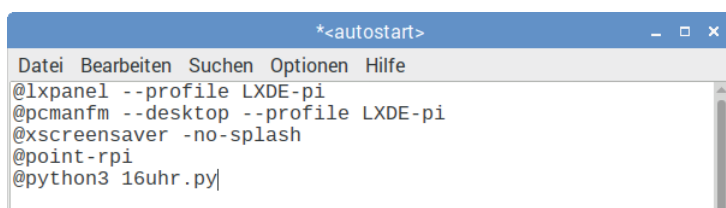
Digitaluhr automatisch starten

Dieses Programm kann aus dem Raspberry Pi eine Digitaluhr machen, die automatisch startet, ohne dass der Raspberry Pi eine Tastatur oder einen Monitor braucht.

Ohne Tastatur und Monitor muss das Programm automatisch gestartet werden, da der Benutzer weder Python-Shell noch Kommandozeile aufrufen kann.



Öffnen Sie im Dateimanager im Verzeichnis `/etc/xdg/lxsession/LXDE-pi` mit dem Texteditor die Textdatei `autostart`. Da diese Datei nur mit Superuserrechten bearbeitet werden kann, wechseln Sie mit dem Dateimanager in das Verzeichnis und wählen dann im Menü **Werkzeuge / Ausführen eines Befehls im aktuellen Ordner...** Geben Sie hier die abgebildete Befehlsfolge ein.



Der Autostart-Aufruf für die Uhr

Fügen Sie am Ende der Datei die abgebildete Zeile zum Aufruf des Programms mit dem Python-Kommandozeileninterpreter hinzu und setzen Sie vor die Zeile `point-rpi` ein `@`-Zeichen.

Damit wird kurz nach dem Booten des Raspberry Pi die aktuelle Uhrzeit automatisch auf der Sieben-Segment-Anzeige angezeigt.

Um den automatischen Start abzuschalten, löschen Sie die Zeile wieder, oder Sie tragen ganz am Anfang der Zeile ein `#`-Zeichen ein:

```
# @python3 16uhr.py
```

Die automatisch gestartete Uhr lässt sich nicht über die Tastenkombination `[Strg]+[C]` beenden, da kein Python-Shell-Fenster existiert.

17. Tag

Heute im Adventskalender

• 1 GPIO-Verbindungskabel

Das Anschlussschema der Sieben-Segment-Anzeige ist das gleiche wie am vorherigen Tag, obwohl die Ziffer ganz rechts nicht benutzt wird.

IP-Adresse des Raspberry Pi anzeigen

Wer einen Raspberry Pi ohne Tastatur und Monitor nur über das Netzwerk betreibt, braucht dessen IP-Adresse, um die SSH-Verbindung aufbauen zu können. Diese Adresse kann man mit Netzwerkscanner-Software herausfinden. Sie lässt sich auch über das Netzwerksymbol auf dem Desktop anzeigen. Wesentlich interessanter ist es, die IP-Adresse mit einer Sieben-Segment-Anzeige direkt auf dem Raspberry Pi anzeigen zu lassen.

Bauteile: 1 Steckbrett SYB-46, 1 Sieben-Segment-Anzeige, 8 220-Ohm-Widerstände (rot-rot-braun), 12 GPIO-Verbindungskabel

Das Programm

Das Programm `17ip-adresse.py` stellt die IP-Adresse auf der Sieben-Segment-Anzeige dar.

Die Sieben-Segment-Anzeige kann vier Ziffern gleichzeitig anzeigen. Eine IP-Adresse besteht aber aus vier Blöcken mit je drei Ziffern. Um diese darzustellen, lassen wir auf der Anzeige nacheinander jeden Zahlenblock etwa 1 Sekunde anzeigen. Um das Ende der IP-Adresse und den Beginn der nächsten Anzeigeschleife deutlich zu kennzeichnen, sollen nach dem vierten Zahlenblock drei Dezimalpunkte der Anzeige kurz aufblinken. Danach startet die Anzeige der IP-Adresse wieder mit dem ersten Zahlenblock.

Zahlenblöcke, die nur aus einer oder zwei Ziffern bestehen, werden für ein einheitliches Erscheinungsbild ebenfalls dreistellig angezeigt und am Anfang mit Nullen ergänzt. Diese Schreibweise ist bei der Angabe von IP-Adressen genauso gültig.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time, os

GPIO.setmode(GPIO.BCM)

seg={'a':21, 'b':8, 'c':11, 'd':26, 'e':19, 'f':20, 'g':13}
for s in "abcdefg":
    GPIO.setup(seg[s], GPIO.OUT, initial=0)

zif=[16, 12, 7, 6]
for z in zif:
    GPIO.setup(z, GPIO.OUT, initial=1)

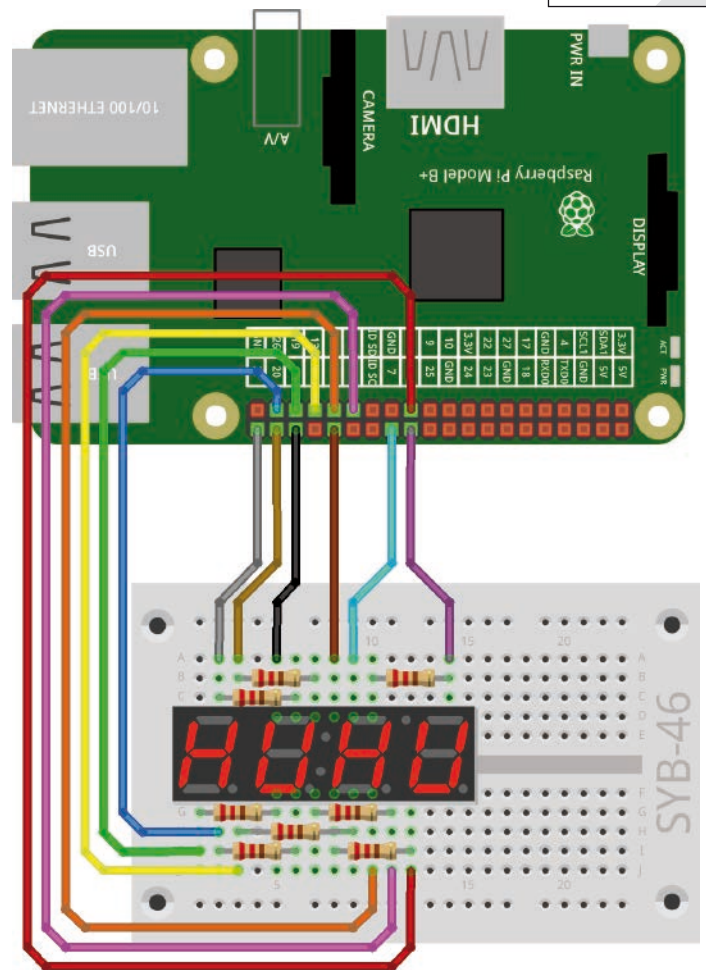
dp = 5
GPIO.setup(dp, GPIO.OUT, initial=0)

zah1=[
    "abcdef", #0
    "bc", #1
    "abdeg", #2
    "abcdg", #3
    "bcfg", #4
    "acdfg", #5
```



Anzeigefolge einer IP-Adresse auf der Sieben-Segment-Anzeige

17. Tag



fritzing

Alle sieben Segmente und der Dezimalpunkt der Sieben-Segment-Anzeige sind angeschlossen. Die Ziffern sind einzeln mit GPIO-Pins verbunden.

```

"acdefg", #6
"abc", #7
"acdefg", #8
"abcdfg", #9
]
z = [0,0,0]
ip = os.popen("hostname -I").readline()[:-2].split(".")
print("Strg+C beendet das Programm")
def za():
    for i in range(3):
        for s in "acdefg":
            GPIO.output(seg[s], 0)
            GPIO.output(zif[i], 0)
        for s in zahl[z[i]]:
            GPIO.output(seg[s], 1)
            time.sleep(0.001)
            GPIO.output(zif[i], 1)
def blink():

```

```

    for s in "acdefg":
        GPIO.output(seg[s], 0)
    for k in range(3):
        GPIO.output(zif[k], 0)
    GPIO.output(dp, 1)
    time.sleep(0.5)
    GPIO.output(dp, 0)
try:
    while True:
        for j in ip:
            for k in range(3):
                z[k] = int(j.zfill(3)[k])
                sek = time.time()
                while time.time() <= sek + 1:
                    za()
                    blink()
except KeyboardInterrupt:
    GPIO.cleanup()

```

So funktioniert es

Große Teile des Programms werden Ihnen bekannt vorkommen. Vieles basiert auf den früheren Python-Programmen mit der Sieben-Segment-Anzeige. Die Funktionen zum Auslesen der IP-Adresse sowie einige Methoden der Zeichenkettenverarbeitung sind neu.

```
import time, os
```

Beim Import der Bibliotheken am Anfang wird zusätzlich das Modul `os` für Betriebssystemfunktionen importiert.

```
zif=[16, 12, 7, 6]
for z in zif:
    GPIO.setup(z, GPIO.OUT, initial=1)
```

Die Liste der GPIO-Anschlüsse für die vier Ziffern bleibt unverändert. Damit die vierte, nicht benutzte Stelle der Anzeige im Programm keinen undefinierten Status haben kann, wird sie am Anfang mit initialisiert und ausgeschaltet.

```
z = [0,0,0]
```

Die Liste `z[]`, in der während des Programmablaufs die darzustellenden Ziffern gespeichert werden, enthält aber nur noch drei Elemente.

```
ip = os.popen("hostname -I").readline()[:-2].split(".")
```

Diese Zeile ermittelt die IP-Adresse zunächst als Zeichenfolge. Die Funktion `os.popen()` führt einen beliebigen Kommandozeilenbefehl aus. Die Methode `readline()` liest aus diesem Ergebnis eine Zeile als Zeichenfolge aus.

Die vierteilige IP-Adresse wird an den Punkten in vier Blöcke getrennt, die als vier einzelne Zeichenketten in der Liste `ip[]` gespeichert werden. Dazu wird die Methode `split()` verwendet, die in jeder Zeichenkette zur Verfügung steht. Sie benötigt als Parameter das Trennzeichen, an dem die Zeichenkette aufgespalten werden soll, in diesem Fall den Punkt. Dieses Trennzeichen selbst erscheint in keiner der entstehenden Zeichenfolgen.

Lautet die IP-Adresse zum Beispiel

```
"192.168.2.124"
```

entsteht daraus diese Liste `ip[]`:

```
('192', '168', '2', '124')
```

Die vier Elemente der Liste sind weiterhin Zeichenketten, keine Zahlen.

Die Funktion `za()` zur Zahlenausgabe entspricht den letzten Programmen. Nur die Zeilen zur Anzeige des Dezimalpunkts zwischen der zweiten und der dritten Ziffer bei der Uhr fallen weg.

```
def blink():
    for s in "abcdefg":
        GPIO.output(seg[s], 0)
    for k in range(3):
        GPIO.output(zif[k], 0)
    GPIO.output(dp, 1)
    time.sleep(0.5)
    GPIO.output(dp, 0)
```

Zusätzlich wird eine Funktion `blink()` definiert, die die drei Dezimalpunkte der verwendeten Ziffern 0,5 Sekunden lang aufblincken lässt. Dazu werden zunächst alle sieben Segmente auf 0 gesetzt und damit ausgeschaltet. Danach werden die Kathoden der ersten drei Ziffern auf 0 und der Dezimalpunkt auf 1 gesetzt.

Jetzt leuchten die drei Dezimalpunkte. Nach einer Wartezeit von 0,5 Sekunden wird der GPIO-Pin des Dezimalpunkts wieder auf 0 gesetzt, und damit werden die LEDs wieder ausgeschaltet. Das Hauptprogramm besteht wieder aus einer Endlosschleife, die zyklisch die vier Zahlenblöcke der IP-Adresse anzeigt.

```
for j in ip:
```

Dazu läuft eine Schleife über die vier Elemente der Liste `ip[]`, in denen diese Zahlenblöcke gespeichert sind.

```
for k in range(3):
```

In jedem Block läuft eine innere Schleife dreimal, um jeweils drei Ziffern darzustellen. Diese werden in den drei Elementen der Liste `z[]` abgelegt, die die Funktion `za()` später ausliest.

```
z[k] = int(j.zfill(3)[k])
```

Diese Zeile schreibt in jedem Durchlauf in das aktuelle Element der Liste `z[]` den Zahlenwert einer Ziffer. Dieser wird mit der Funktion `int()` aus einer als Zeichen gespeicherten Ziffer gelesen.

Aus der äußeren Schleife übernimmt der Zähler `j` einen Zahlenblock der IP-Adresse. Die Methode `zfill()`, die in jeder Zeichenkette zur Verfügung steht, und bereits in einem früheren Programm verwendet wurde, ergänzt eine Zeichenkette auf eine bestimmte Länge, die im Parameter (hier 3) angegeben ist, wobei die Originalzeichenkette im Ergebnis rechts steht. Links werden eventuell fehlende Stellen mit Nullen aufgefüllt. Auf diese Weise werden in der Liste `z[]` immer drei Ziffern abgelegt, auch wenn ein Zahlenblock der IP-Adresse nur aus einer oder zwei Ziffern besteht.

So werden aus dem weiter oben als Beispiel verwendeten Feld `ip[]`

```
('192', '168', '2', '124')
```

in vier Durchläufen jeweils diese Felder `z[]`:

```
(1, 9, 2)
```

```
(1, 6, 8)
```

```
(0, 0, 2)
```

```
(1, 2, 4)
```

Die Funktion `za()` liest später das Feld `z[]` aus und stellt die drei Ziffern auf der Sieben-Segment-Anzeige dar. Eine `while`-Schleife lässt 1 Sekunde lang immer wieder die Funktion `za()` zur Zahlenanzeige laufen und beendet danach die Hauptschleife.

```
sek = time.time()
while time.time() <= sek + 1:
    za()
```

In der Variablen `sek` wird die aktuelle Zeit in Sekunden gespeichert, die die Funktion `time.time()` jederzeit auf die Hundertstelsekunde genau ausgibt. Die `while`-Schleife fragt ab, ob seit dem Speichern der Zeit eine oder mehr Sekunden vergangen sind. Solange dies nicht der Fall ist, wird immer wieder die Funktion `za()` zur Zahlenanzeige aufgerufen. Nach Ablauf der Sekunde startet die äußere Schleife erneut mit dem nächsten Zahlenblock der IP-Adresse.

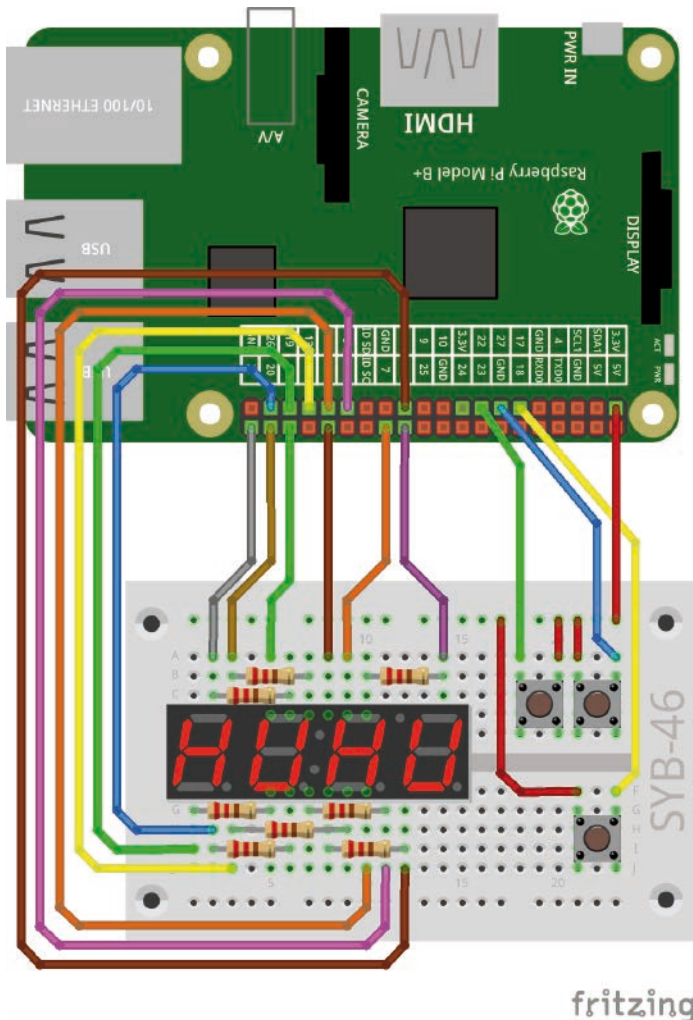
Nachdem diese Schleife viermal durchgelaufen ist, wird die Funktion `blink()` aufgerufen, die die Dezimalpunkte einmal kurz aufblincken lässt, um das Ende der IP-Adresse anzuzeigen. Danach startet die Endlosschleife erneut. Das Ganze wiederholt sich, bis der Benutzer mit `[Strg]+[C]` abbricht. Dieses Programm kann ähnlich wie das Programm mit der Uhr vom 16. Tag beim Start des Raspberry Pi automatisch gestartet werden, um die IP-Adresse anzuzeigen, auch wenn kein Bildschirm angeschlossen ist.

18. Tag

Heute im Adventskalender

• 1 Taster

Das Anschlusschema der Sieben-Segment-Anzeige ist das gleiche wie an den vorherigen Tagen.



Sieben-Segment-Anzeige und drei Tasten

Stoppuhr

Das Experiment des 18. Tags ist eine Stoppuhr mit drei Tasten. Die erste Taste startet die Stoppuhr, die zweite stoppt auf Zehntelsekunden genau und die dritte setzt die gemessene Zeit wieder auf 0 zurück.

Bauteile: 1 Steckbrett SYB-46, 1 Sieben-Segment-Anzeige, 8 220-Ohm-Widerstände (rot-rot-braun), 3 Taster, 16 GPIO-Verbindungskabel, 3 Drahtbrücken (unterschiedliche Längen)

Das Programm

Das Programm 18stoppuhr.py lässt eine Stoppuhr auf der Sieben-Segment-Anzeige laufen. Die Uhr misst die Zeit in Zehntelsekunden, deshalb leuchtet hinter der dritten Ziffer der Dezimalpunkt. Die Tasten sind an den GPIO-Pins 22, 27 und 17 angeschlossen.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

seg={'a':21, 'b':8, 'c':11, 'd':26, 'e':19, 'f':20, 'g':13}
for s in "abcdefg":
    GPIO.setup(seg[s], GPIO.OUT, initial=0)

zif=[16, 12, 7, 6]
for z in zif:
    GPIO.setup(z, GPIO.OUT, initial=1)

dp = 5
GPIO.setup(dp, GPIO.OUT, initial=0)

t1=22
t2=27
t3=17
GPIO.setup(t1, GPIO.IN, GPIO.PUD_DOWN)
GPIO.setup(t2, GPIO.IN, GPIO.PUD_DOWN)
GPIO.setup(t3, GPIO.IN, GPIO.PUD_DOWN)

zahl=[
    "abcdef", #0
    "bc", #1
    "abdeg", #2
    "abcdg", #3
    "bcfg", #4
    "acdfg", #5
    "acdefg", #6
    "abc", #7
    "abcdefg", #8
    "abcdfg", #9
]

print("Start: Taste 1")
print("Stopp: Taste 2")
print("Reset: Taste 3")
print("Strg+C beendet das Programm")
```



```

def za():
    for i in range(4):
        for s in "abcdefg":
            GPIO.output(seg[s], 0)
        GPIO.output(zif[i], 0)
        for s in zahl[z[i]]:
            GPIO.output(seg[s], 1)
        if i == 2:
            GPIO.output(dp, 1)
        else:
            GPIO.output(dp, 0)
        time.sleep(0.001)
        GPIO.output(zif[i], 1)

try:
    while True:
        z = [0,0,0,0]
        while GPIO.input(t1)==0:
            za()
        start = time.time()
        while GPIO.input(t2)==0:
            zeit = time.time() - start
            z[0] = int(zeit % 1000 / 100)
            z[1] = int(zeit % 100 / 10)
            z[2] = int(zeit % 10)
            z[3] = int(zeit * 10 % 10)
            while int((time.time() - start) * 10 % 10) == z[3]:
                za()
        while GPIO.input(t3)==0:
            za()

except KeyboardInterrupt:
    GPIO.cleanup()

```

So funktioniert es

Das Programm basiert auf bereits bekannten Elementen, auch die Initialisierung der GPIO-Pins für die Sieben-Segment-Anzeige läuft gleich wie an den vorherigen Tagen.

```

t1=22
t2=27
t3=17
GPIO.setup(t1, GPIO.IN, GPIO.PUD_DOWN)
GPIO.setup(t2, GPIO.IN, GPIO.PUD_DOWN)
GPIO.setup(t3, GPIO.IN, GPIO.PUD_DOWN)

```

Zusätzlich zu den GPIO-Pins der Sieben-Segment-Anzeige werden drei Pins als Eingänge mit Pulldown-Widerständen für die Taster initialisiert.

```

print("Start: Taste 1")
print("Stopp: Taste 2")
print("Reset: Taste 3")
print("Strg+C beendet das Programm")

```

Vor dem eigentlichen Start des Programms erscheint eine kurze Erklärung der Tasten.

```

try:
    while True:
        z = [0,0,0,0]

```

Die Hauptschleife des Programms setzt am Anfang jedes Durchlaufs die Stoppuhr auf 0. Die vier Ziffern in der Liste `z[]` werden auf 0 gesetzt.

```

    while GPIO.input(t1)==0:
        za()

```

So lange die Taste 1 nicht gedrückt ist, zeigt die Funktion `za()` vier Nullen auf der Sieben-Segment-Anzeige an.

```

if i == 2:
    GPIO.output(dp, 1)
else:
    GPIO.output(dp, 0)

```

Diese Funktion ist etwas verändert. Immer wenn die dritte Ziffer leuchtet, wird zusätzlich der Dezimalpunkt eingeschaltet. Bei den anderen Ziffern bleibt er aus.

```
start = time.time()
```

Drückt der Benutzer die Taste 1, endet die erste `while`-Schleife. Die aktuelle Zeit in Sekunden wird in der Variable `start` gespeichert. Aus dieser Zeit wird jetzt immer die aktuelle Zeit der Stoppuhr errechnet, die durch Drücken der Taste 1 gestartet wurde.

Die Funktion `time.time()` gibt die aktuelle Zeit als Unix Timestamp an.

Was ist der Unix Timestamp?

Der Unix Timestamp, auch als Unix Epoche bezeichnet, ist ein allgemein in Unix und Linux verwendetes Zeitformat, das die Anzahl von Sekunden angibt, die seit dem 01.01.1970 00:00 UTC vergangen sind. Dabei handelt es sich um eine zehnstellige Zahl. Nach dem Komma folgen Bruchteile von Sekunden. Der Unix Timestamp am 01.12.2019 00:00 UTC lautet 1575158400.

```

while GPIO.input(t2)==0:
    zeit = time.time() - start
    z[0] = int(zeit % 1000 / 100)
    z[1] = int(zeit % 100 / 10)
    z[2] = int(zeit % 10)
    z[3] = int(zeit * 10 % 10)

```

Die nächste Schleife läuft, bis der Benutzer die Stoptaste 2 drückt. In jedem Durchlauf wird in der Variable `zeit` die seit dem Start der Stoppuhr abgelaufene Zeit durch Subtraktion der gespeicherten Startzeit von der aktuellen Zeit berechnet.

In den vier Elementen der Liste `z[]` werden ganze Sekunden als dreistellige Zahl und Zehntelsekunden an der vierten Stelle errechnet und gespeichert.

```

while int((time.time() - start) * 10 % 10) == z[3]:
    za()

```

Jetzt startet eine Schleife, die mit der Funktion `za()` die in der Liste `z[]` gespeicherten Ziffern so lange anzeigt, bis seit der zuletzt angezeigten Zeit eine Zehntelsekunde vergangen ist. Erst wenn eine Zehntelsekunde abgelaufen ist, startet die übergeordnete `while`-Schleife den nächsten Durchlauf. Jetzt werden neue Werte für die Ziffern in der Liste `z[]` ermittelt - vorausgesetzt, die Stoptaste 2 wurde noch nicht gedrückt.

```

while GPIO.input(t3)==0:
    za()

```

Wurde die Stoptaste 2 gedrückt, endet die Schleife, und es beginnt eine weitere `while`-Schleife, die darauf wartet, dass der Benutzer die Rest-Taste 3 drückt. Solange dies nicht der Fall ist, werden die zuletzt gespeicherten Ziffern mit der Funktion `za()` in der Liste `z[]` angezeigt.

Beim Drücken der Taste 3 endet auch diese Schleife, und die Hauptschleife startet ihren nächsten Durchlauf, in dem als erstes die vier Ziffern wieder auf 0 gesetzt werden.

19. Tag

Heute im Adventskalender

• 1 Widerstand 20 MOhm (rot-schwarz-blau)

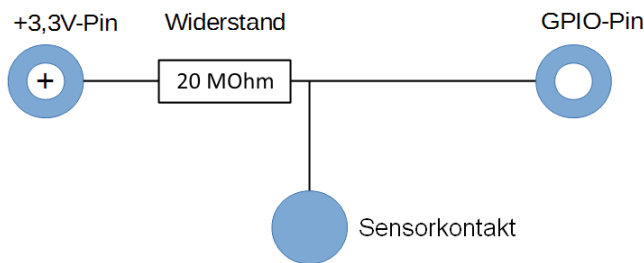
• 1 Knete

Das Anschlusschema der Sieben-Segment-Anzeige ist das gleiche wie an den vorherigen Tagen.

Sensorkontakt aus Knete

Ampeln, Türöffner, Lichtschalter und Automaten werden heute oft mit Sensorkontakten gesteuert, die man nur zu berühren braucht. Taster, die wirklich gedrückt werden müssen, werden immer seltener.

Der als Eingang geschaltete GPIO-Pin ist über einen extrem hochohmigen Widerstand (20 MOhm) mit +3,3 V verbunden, sodass ein schwaches, aber eindeutig als 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 Low-Pegel. Berührt dieser Mensch einen Sensorkontakt, wird das schwache High-Signal von dem deutlich stärkeren Low-Pegel der Hand überlagert und zieht den GPIO-Pin auf Low-Pegel. Der eingebaute Pull-down-Widerstand am GPIO-Eingang muss dazu ausgeschaltet sein.



Sensorkontakt an einem GPIO-Eingang

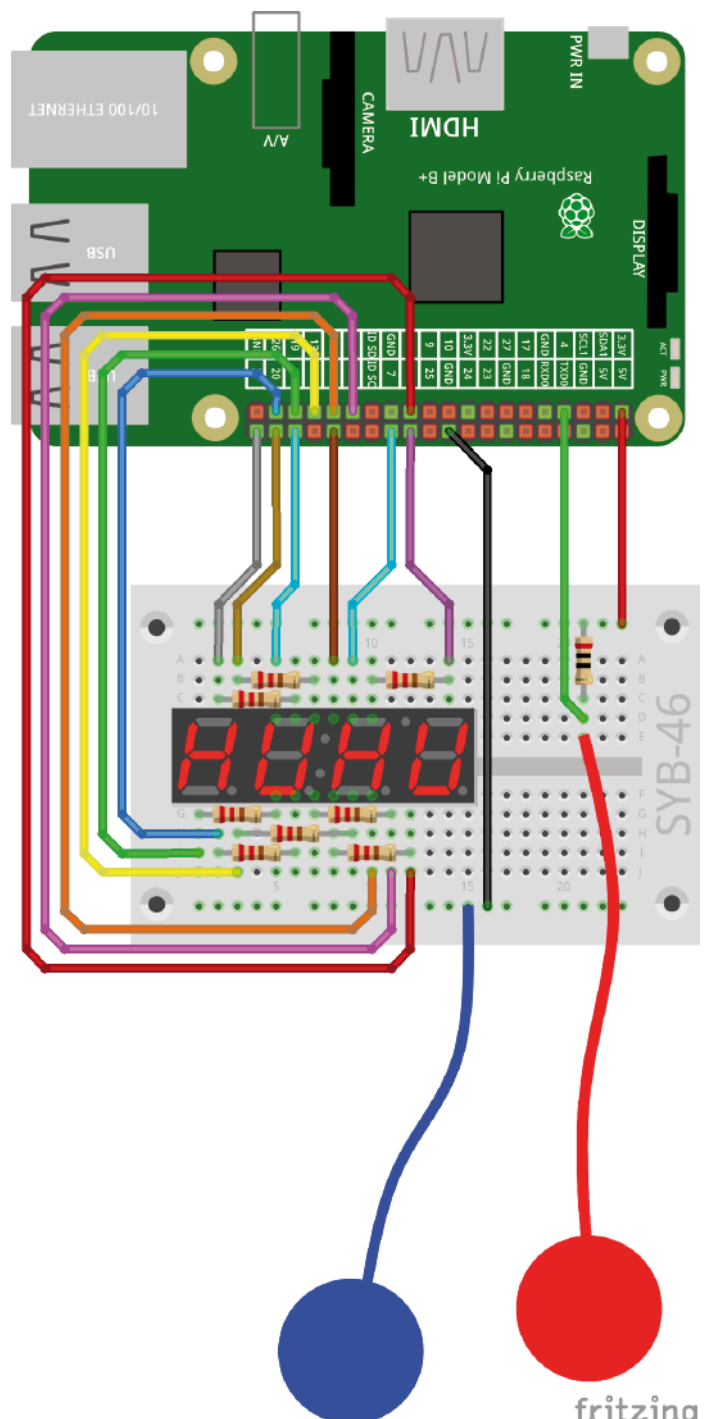
Wie hoch allerdings der Widerstand zwischen Hand und Masse wirklich ist, hängt von vielen Dingen ab, unter anderem von Schuhen und Fußböden. Barfuss im nassen Gras ist die Verbindung zur Masse der Erde am besten, aber auch auf Steinfußboden funktioniert es meistens gut. Holzfußböden isolieren stärker, Kunststoffbodenbeläge sind oft sogar positiv aufgeladen. Für den Fall, dass der Sensorkontakt nicht funktioniert, enthält die Schaltung einen zweiten Knetekontakt, der mit der Masseschiene des Steckbretts verbunden ist. Berühren Sie diesen und den eigentlichen Sensor gleichzeitig, ist die Masseverbindung auf jeden Fall hergestellt.

Knete leitet den Strom etwa so gut wie menschliche Haut. Sie lässt sich leicht in jede beliebige Form bringen, und ein Knetekontakt fasst sich viel besser an als ein einfaches Stück Draht. Die Fläche, mit der die Hand den Kontakt berührt, ist deutlich größer. So kommt es nicht so leicht zu einem „Wackelkontakt“. Stecken Sie ein Stück abisolierten Schaltdraht in ein Stück Knete. Das andere Drahtende stecken Sie in das Steckbrett.

Stoppuhr mit Sensorkontakt

Das Experiment des 19. Tags steuert eine Stoppuhr über einen einfachen Sensorkontakt.

Bauteile: 1 Steckbrett SYB-46, 1 Sieben-Segment-Anzeige, 8 220-Ohm-Widerstände (rot-rot-braun), 1 20-MOhm-Widerstand (rot-schwarz-blau), 2 Knetekontakte, 15 GPIO-Verbindungskabel



Sieben-Segment-Anzeige und Sensorkontakte

19. Tag

Das Programm

Das Programm `19stoppuhr.py` ähnelt dem Programm des 18. Tags, aber mit dem Unterschied, dass für Start, Stopp und Reset nur ein einziger Sensorkontakt verwendet wird.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

seg={'a':21, 'b':8, 'c':11, 'd':26, 'e':19, 'f':20, 'g':13}
for s in "abcdefg":
    GPIO.setup(seg[s], GPIO.OUT, initial=0)

zif=[16, 12, 7, 6]
for z in zif:
    GPIO.setup(z, GPIO.OUT, initial=1)

dp = 5
GPIO.setup(dp, GPIO.OUT, initial=0)

k1=4
GPIO.setup(k1, GPIO.IN)

zahl=[
    "abcdef", #0
    "bc", #1
    "abdeg", #2
    "abcdg", #3
    "bcfg", #4
    "acdfg", #5
    "acdefg", #6
    "abc", #7
    "abcdefg", #8
    "abcdfg", #9
    ]

print("Start, Stopp, Reset: Sensorkontakt berühren")
print("Strg+C beendet das Programm")

def za():
    for i in range(4):
        for s in "abcdefg":
            GPIO.output(seg[s], 0)
        GPIO.output(zif[i], 0)
        for s in zahl[z[i]]:
            GPIO.output(seg[s], 1)
        if i == 2:
            GPIO.output(dp, 1)
        else:
            GPIO.output(dp, 0)
        time.sleep(0.001)
        GPIO.output(zif[i], 1)

try:
    while True:
        z = [0,0,0,0]
        while GPIO.input(k1)==1:
            za()
        while GPIO.input(k1)==0:
            pass
        start = time.time()
        while GPIO.input(k1)==1:
            zeit = time.time() - start
            z[0] = int(zeit % 1000 / 100)
```

```

z[1] = int(zeit % 100 / 10)
z[2] = int(zeit % 10)
z[3] = int(zeit * 10 % 10)
while int((time.time() - start) * 10 % 10) == z[3]:
    za()
while GPIO.input(k1)==0:
    pass
while GPIO.input(k1)==1:
    za()
while GPIO.input(k1)==0:
    pass

except KeyboardInterrupt:
    GPIO.cleanup()

```

So funktioniert es

Das Programm basiert auf bereits bekannten Elementen, auch die Initialisierung der GPIO-Pins für die Sieben-Segment-Anzeige läuft wie an den vorherigen Tagen.

```

k1=4
GPIO.setup(k1, GPIO.IN)

```

Zusätzlich zu den GPIO-Pins der Sieben-Segment-Anzeige wird ein Pin als Eingang ohne Pulldown-Widerstand für den Sensorkontakt initialisiert.

```

try:
    while True:
        z = [0,0,0,0]
        while GPIO.input(k1)==1:
            za()

```

Da ein Sensorkontakt beim Berühren auf Low-Pegel springt, läuft diesmal die erste `while`-Schleife innerhalb der Endlosschleife so lange, wie der Eingang `k1` High-Pegel liefert.

Da in dieser Programmversion keine drei verschiedenen Tasten für Start, Stopp und Reset, sondern nur ein Sensor verwendet wird, muss verhindert werden, dass eine längere Berührung mehrere Aktionen hintereinander auslöst.

```

while GPIO.input(k1)==0:
    pass

```

Dazu läuft eine weitere Schleife so lange, wie der Eingang `k1` Low-Pegel liefert, der Sensorkontakt also berührt wird. Innerhalb der Schleife passiert gar nichts. Python verwendet dazu die Anweisung `pass`, die einfach nichts tut. Damit lassen sich solche Warteschleifen leicht realisieren. Die nächste Aktion im Programm findet erst statt, nachdem der Benutzer den Sensorkontakt wieder losgelassen hat. Als auffälliges Zeichen dafür schaltet sich die Anzeige während der Berührung des Sensorkontakts aus, da sie nur leuchtet, während die Funktion `za()` läuft.

Hat der Benutzer den Sensorkontakt wieder losgelassen, läuft das Programm weiter und die Stoppuhr beginnt zu laufen.

```

while GPIO.input(k1)==1:
    zeit = time.time() - start

```

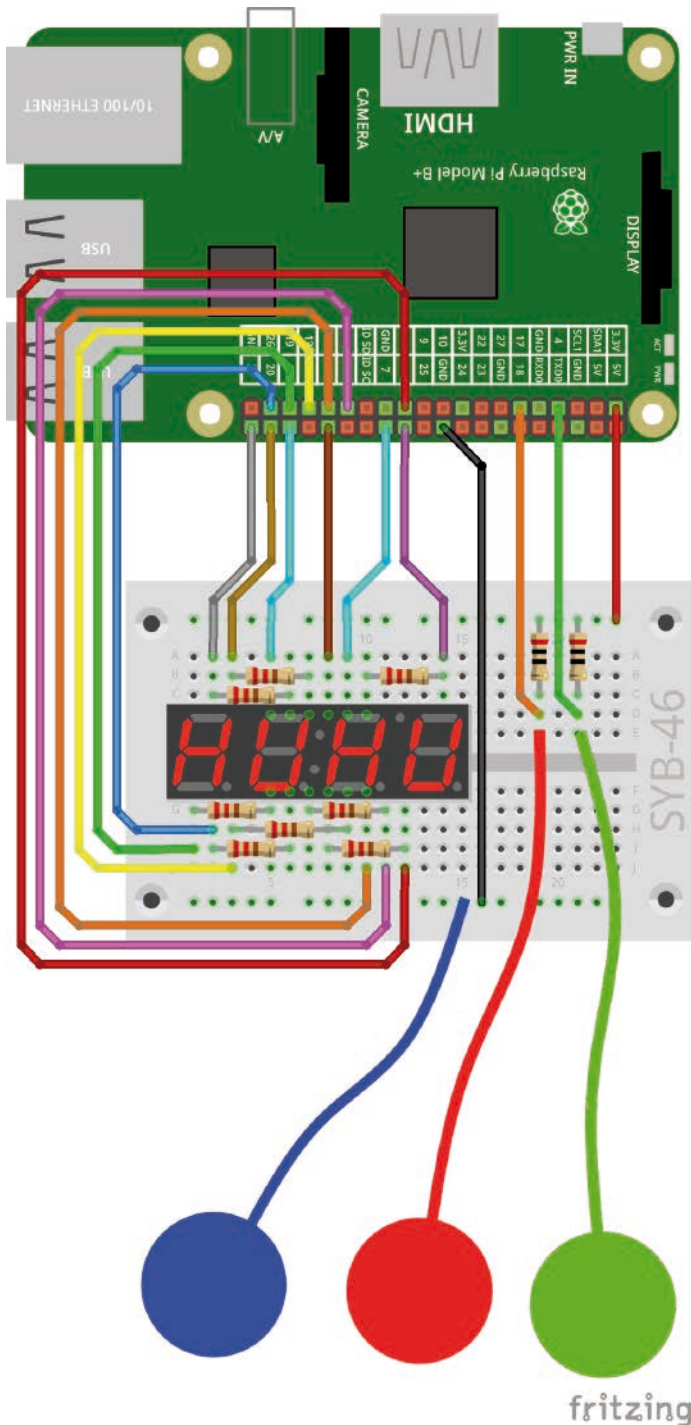
Auch hier läuft eine `while`-Schleife so lange, wie der Eingang `k1` High-Pegel liefert, der Sensorkontakt also nicht berührt wird. Nach dieser Schleife und auch nach dem Reset auf 0 wartet das Programm jedes Mal, bis der Benutzer den Sensorkontakt wieder losgelassen hat.

20. Tag

Heute im Adventskalender

• 1 Widerstand 20 MOhm (rot-schwarz-blau)

Das Anschlusschema der Sieben-Segment-Anzeige ist das gleiche wie an den vorherigen Tagen.



Sieben-Segment-Anzeige und Sensorkontakte.

Zähler mit Sensorkontakten

Das Experiment des 20. Tags ist ein einfacher Zähler auf der Sieben-Segment-Anzeige, der mit zwei Sensorkontakten gesteuert wird. Der eine Sensorkontakt zählt den Zähler hoch, der andere herunter. Die Sensorkontakte sind an den GPIO-Pins 17 und 4 angeschlossen.

Bauteile: 1 Steckbrett SYB-46, 1 Sieben-Segment-Anzeige, 8 220-Ohm-Widerstände (rot-rot-braun), 2 20-MOhm-Widerstände (rot-schwarz-blau), 3 Knetkontakte, 16 GPIO-Verbindungskabel

Das Programm

Das Programm `20zaehler.py` zählt auf der Sieben-Segment-Anzeige von 0 aufwärts bis maximal 9999 oder wieder abwärts bis 0. Diesmal wird nicht gewartet, bis der Benutzer den Sensorkontakt wieder loslässt. Durch längere Berührung soll automatisch weiter gezählt werden.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

seg={'a':21, 'b':8, 'c':11, 'd':26, 'e':19, 'f':20, 'g':13}
for s in "abcdefg":
    GPIO.setup(seg[s], GPIO.OUT, initial=0)

zif=[16, 12, 7, 6]
for z in zif:
    GPIO.setup(z, GPIO.OUT, initial=1)

dp=5
GPIO.setup(dp, GPIO.OUT, initial=0)

k1=17
k2=4
GPIO.setup(k1, GPIO.IN)
GPIO.setup(k2, GPIO.IN)

zahl=[
    "abcdef", #0
    "bc", #1
    "abdeg", #2
    "abcdg", #3
    "bcfg", #4
    "acdfg", #5
    "acdefg", #6
    "abc", #7
    "abcdefg", #8
    "abcdfg", #9
]

z=[0,0,0,0]
x=0

print("Sensorkontakt 1 erhöht die Zahl")
print("Sensorkontakt 2 verringert die Zahl")
print("Strg+C beendet das Programm")
```

```

def za(n):
    z[0] = int(n / 1000)
    z[1] = int(n % 1000 / 100)
    z[2] = int(n % 100 / 10)
    z[3] = int(n % 10)
    for i in range(4):
        for s in "abcdefg":
            GPIO.output(seg[s], 0)
        GPIO.output(zif[i], 0)
        for s in zahl[z[i]]:
            GPIO.output(seg[s], 1)
        time.sleep(0.001)
        GPIO.output(zif[i], 1)

try:
    while True:
        za(x)
        if GPIO.input(k1)==0 and x>0:
            x-=1
        if GPIO.input(k2)==0 and x<9999:
            x+=1

except KeyboardInterrupt:
    GPIO.cleanup()

```

So funktioniert es

Auch dieses Programm basiert zu großen Teilen auf früheren Programmen. Die Funktionen zur Initialisierung der Sieben-Segment-Anzeige sind gleich.

```

k1=17
k2=4
GPIO.setup(k1, GPIO.IN)
GPIO.setup(k2, GPIO.IN)

```

Für die beiden Sensorkontakte werden zwei GPIO-Pins als Eingänge ohne Pulldown-Widerstand initialisiert.

```
x=0
```

Bei den Definitionen wird noch eine neue Variable *x* auf 0 gesetzt. Sie enthält später im Programm den wert des Zählers, der angezeigt wird.

In diesem Programm wird die Berechnung der vier Ziffern in die Funktion `za()` ausgelagert, was das Hauptprogramm vereinfacht. Diese Funktion enthält dazu einen Parameter, der beim Funktionsaufruf übergeben wird.

```

def za(n):
    z[0] = int(n / 1000)
    z[1] = int(n % 1000 / 100)
    z[2] = int(n % 100 / 10)
    z[3] = int(n % 10)
    for i in range(4):
        for s in "abcdefg":
            GPIO.output(seg[s], 0)
        GPIO.output(zif[i], 0)
        for s in zahl[z[i]]:
            GPIO.output(seg[s], 1)
        time.sleep(0.001)
        GPIO.output(zif[i], 1)

```

Der Parameter *n* in der Funktionsdefinition ist eine vierstellige Zahl, die angezeigt werden soll. Die ersten vier Zeilen der Funktion ermitteln aus der Zahl die einzelnen Ziffern. Danach läuft die Funktion wie in früheren Programmen und stellt diese vier Ziffern auf der Sieben-Segment-Anzeige dar.

```
try:
    while True:
        za(x)
```

Die Hauptschleife des Programms zeigt in jedem Durchlauf über die Funktion `za(x)` den aktuellen Wert des Zählers `x` auf der Sieben-Segment-Anzeige an.

```
    if GPIO.input(k1)==0 and x>0:
        x-=1
```

Wird der Sensorkontakt `k1` berührt und ist der Zähler noch größer als 0, wird er um 1 verringert. Auf diese Weise wird sichergestellt, dass der Zähler keine negativen Werte erreicht.

```
    if GPIO.input(k2)==0 and x<9999:
        x+=1
```

Wird der Sensorkontakt `k2` berührt und ist der Zähler noch kleiner als 9999, wird er um 1 erhöht. Auf diese Weise wird sichergestellt, dass der Zähler keine fünfstelligen Werte erreicht, die nicht mehr dargestellt werden könnten.

Danach startet die Hauptschleife neu und zeigt wieder den aktuellen Wert des Zählers an.

Der Zähler zählt zu schnell

Wenn Sie das Programm ausprobieren, werden Sie feststellen, dass der Zähler bereits auf kürzeste Berührungen reagiert und es kaum möglich ist, einzelne Schritte zu zählen.

Natürlich könnte man, wie im letzten Programm, jedes Mal warten, dass der Benutzer den Sensorkontakt wieder loslässt. Dann wäre es aber nicht möglich, durch längeres Berühren schnell weit zu zählen.

Das Programm `20zaehler02.py` enthält nach jeder Veränderung des Zählers eine kurze Wartezeit, um bei kurzer Berührung auch einzelne Schritte zählen zu können.

```
try:
    while True:
        za(x)
        if GPIO.input(k1)==0 and x>0:
            x-=1
            time.sleep(0.04)
        if GPIO.input(k2)==0 and x<9999:
            x+=1
            time.sleep(0.04)
```

Verändern Sie je nach Bedarf diese Wartezeiten, um komfortabel zählen zu können.

21. Tag

Heute im Adventskalender

• Downloadcode

Das Anschlussschema der Sieben-Segment-Anzeige ist das gleiche wie an den vorherigen Tagen.

Der Raspberry Pi erzeugt Töne

Der Raspberry Pi kann über einen HDMI-Monitor, einen externen Lautsprecher oder Kopfhörer an der analogen 3,5-mm-Klinkenbuchse Musik abspielen. Bei Computermonitoren mit DVI-Anschluss, die am HDMI-Ausgang angeschlossen sind, muss am analogen Ausgang ein Lautsprecher angeschlossen werden, da das Audiosignal nicht über das DVI-Kabel übertragen wird. Klicken Sie mit der rechten Maustaste oben rechts auf das Lautsprechersymbol, um den Audioausgang auszuwählen.



Audioausgang auswählen

Bauteile: 1 Steckbrett SYB-46, 1 Sieben-Segment-Anzeige, 8 220-Ohm-Widerstände (rot-rot-braun), 12 GPIO-Verbindungskabel

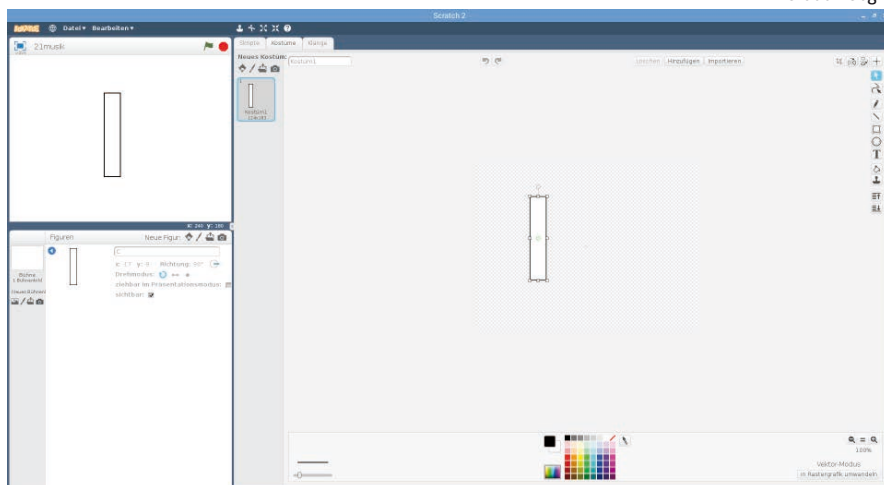
Das Programm

Das Scratch-2-Programm `21musik.sb2` zeigt ein einfaches Klavier. Beim Anklicken einer Taste ertönt der entsprechende Ton, und die Sieben-Segment-Anzeige zeigt den Ton als Buchstaben an.

Starten Sie ein neues Projekt in Scratch und löschen Sie als Erstes die Katze. In diesem Projekt brauchen wir diverse andere Objekte, genauer gesagt, für jede Klaviertaste ein Objekt, die Katze aber nicht.

Zeichnen Sie die erste weiße Klaviertaste. Schalten Sie dazu im Grafikbereich unten rechts auf Vektorgrafik um. Zeichnen Sie mit schwarzer Farbe ein nicht gefülltes schmales Rechteck.

Wählen Sie danach die weiße Farbe, klicken Sie auf das Farbeimer-Symbol und füllen Sie das Rechteck damit aus.

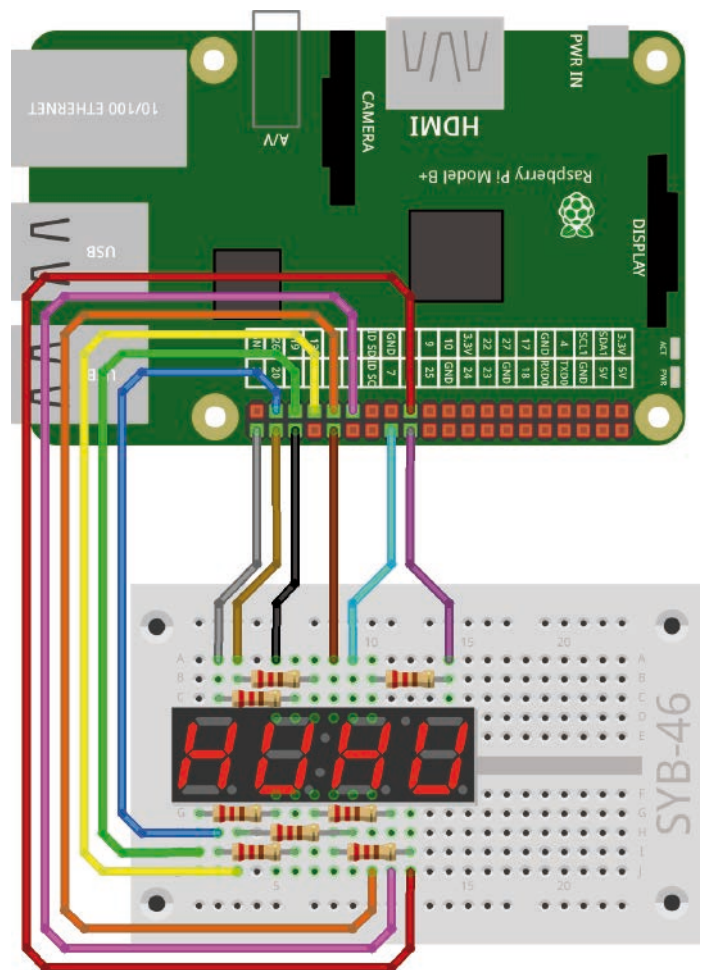


Die erste Klaviertaste

Geben Sie der Taste gleich noch einen Namen. Bei vielen Objekten in einem Programm kommt man sonst schnell durcheinander. Klicken Sie dazu auf das blaue *i*-Symbol dieser Taste in der Figurenliste und ändern Sie im Namensfeld den Namen `c`. Das ist der Ton, den die erste Taste spielen soll.

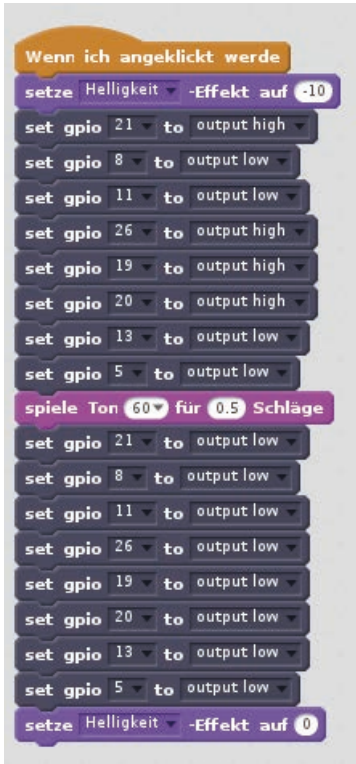
Jede Taste bekommt jetzt ihre eigenen Programmblöcke. Diese sind für alle Tasten ähnlich und spielen beim Anklicken der Taste einen bestimmten Ton ab. Bauen Sie das Programm für die erste Taste zusammen, dann können Sie es mit den anderen Tasten duplizieren und brauchen es nur noch zu verändern.

21. Tag



fritzing

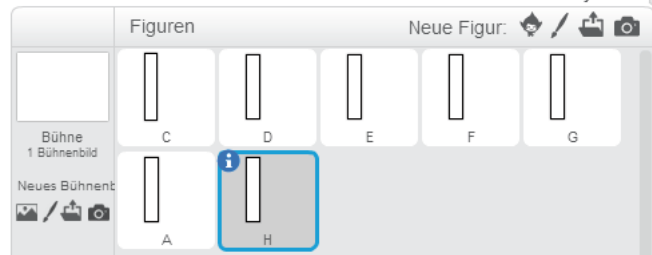
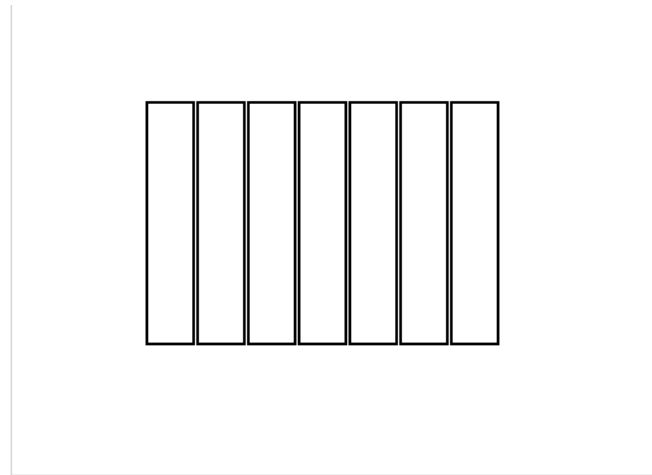
Sieben-Segment-Anzeige ohne zusätzliche Bauelemente



Die Scratch-Blöcke für die Taste C



Klavier zur Auswahl der Töne

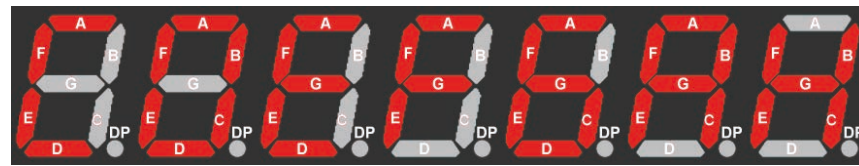


Die weißen Tasten des Klaviers

Die Programmblöcke starten nicht bei Klick auf das grüne Fähnchen, sondern jedes Mal, wenn die entsprechende Taste angeklickt wird. Markieren Sie in der Figurenliste die Taste **C** und ziehen Sie aus der Blockpalette **Ereignisse** den Block **Wenn ich angeklickt werde** ins Skriptfenster.

Alle Blöcke darunter werden immer dann ausgeführt, wenn die Figur - hier die Taste **C** - angeklickt wird. Damit man deutlich sieht, dass eine Klaviertaste gedrückt wurde, soll sie kurz etwas abgedunkelt erscheinen. Hängen Sie dazu aus der Blockpalette **Aussehen** den Block **setze ...Effekt auf...** an das Programm, wählen Sie im Listenfeld **Helligkeit** und tragen Sie in das Zahlenfeld **-10** ein. Damit verdunkelt sich die Figur.

Anschließend werden die sieben Segmente der Sieben-Segment-Anzeige so geschaltet, dass sie den Namen des Tons als Buchstaben anzeigen. Sieben-Segment-Anzeigen sind für die Darstellung von Buchstaben nur bedingt geeignet. Mit etwas Fantasie lassen sich die für das Klavier benötigten Buchstaben aber erkennen.



Die Buchstaben der verwendeten Töne auf der Sieben-Segment-Anzeige

Um den richtigen Ton zu erzeugen, bauen Sie den Block **spiele Ton... für... Schläge** von der Blockpalette **Klang** in das Programm ein.

Klicken Sie einfach einmal auf diesen Block im Programmfenster, ertönt der erste Ton. Klicken Sie dann in das Zahlenfeld hinter **Ton**. Es erscheint eine Klaviatur, auf der Sie den gewünschten Ton auswählen können. Die erste Klaviertaste **C** soll den Ton **C(60)** spielen.

Danach werden alle sieben Segmente der Sieben-Segment-Anzeige ausgeschaltet. Die Taste soll auch wieder normal und nicht abgedunkelt dargestellt werden. Hängen Sie dazu einen weiteren Block **setze Helligkeit-Effekt auf...** an das Programm und setzen dort den Wert wieder auf **0**.

Unser Klavier soll sieben weiße und fünf schwarze Tasten haben. Die nächsten sechs weißen Tasten können Sie einfach aus der ersten duplizieren. Klicken Sie dazu mit der rechten Maustaste auf die Figur **c** in der Figurenliste und wählen Sie im Menü **Duplizieren**.

Benennen Sie die duplizierte Taste in **D** um und schieben Sie sie mit winzigem Abstand rechts neben die erste Taste. Duplizieren Sie auf die gleiche Weise weitere Tasten und nennen Sie sie **E, F, G, A** und **H**.

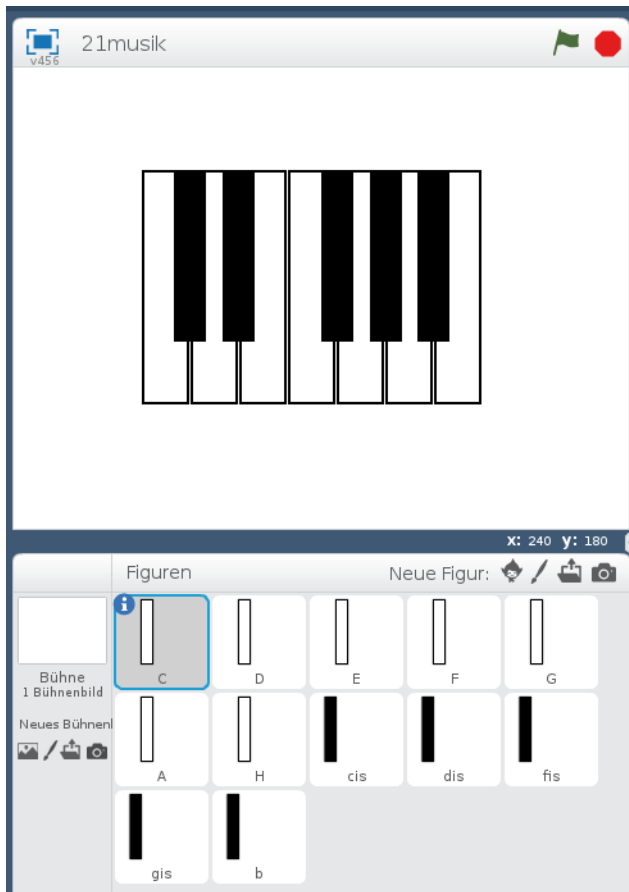
Ändern Sie nach dem Duplizieren bei allen Tasten die Blöcke zur Darstellung der Segmente so, dass der jeweilige Buchstabe angezeigt wird. Wählen Sie außerdem die passenden Töne für die Tasten. Welche Taste welchen Ton spielen soll, ist in der Klaviertastenabbildung des Blocks **spiele Ton... für... Schläge** gut zu erkennen. Die Bezeichnungen der Töne lauten in Scratch manchmal etwas anders als im deutschen Sprachgebrauch.

Ton	Tastensfarbe	Nummer	Scratch
c	weiß	60	Mittleres C
cis	schwarz	61	C#
d	weiß	62	D
dis	schwarz	63	Eb
e	weiß	64	E
f	weiß	65	F
fis	schwarz	66	F#
g	weiß	67	G
gis	schwarz	68	G#
a	weiß	69	A
b	schwarz	70	Bb
h	weiß	71	B

Zeichnen Sie die erste schwarze Taste als ausgefülltes Rechteck und etwas kleiner als die weißen Tasten. Verwenden Sie auch hier den Modus **Vektorgrafik**. Schieben Sie diese Taste zwischen die ersten beiden weißen Tasten, sodass die oberen Kanten miteinander abschließen.

Die Programmblöcke der schwarzen Tasten sind weitgehend gleich, wie bei den weißen Tasten. Die schwarzen Klaviertasten müssen beim Anklicken etwas heller und nicht dunkler werden, damit sie gut zu erkennen sind. Setzen Sie hier die Helligkeit auf 40 statt auf -10. Als Buchstaben für die Töne verwenden wir jeweils den Buchstaben der weißen Taste links daneben und schalten zur Unterscheidung zusätzlich den Dezimalpunkt ein.

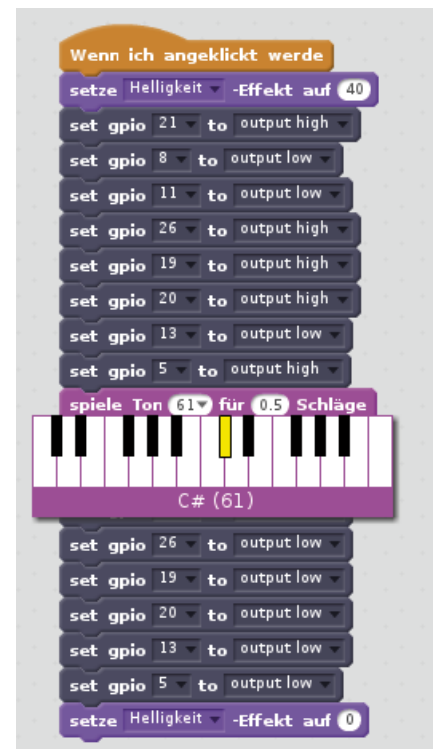
Benennen Sie diese Figur in `cis` um und duplizieren Sie daraus vier weitere schwarze Tasten. Bauen Sie diese wie abgebildet zwischen den weißen Tasten ein und nennen Sie sie `dis`, `fis`, `gis` und `b`.



Die weißen und schwarzen Tasten des Klaviers

Passen Sie dann noch die Programmblöcke der Tasten entsprechend an. Jetzt können Sie Musik auf dem Klavier spielen.

Über den Downloadcode heute im Adventskalender können Sie sich Noten für ein Weihnachtslied zum Nachspielen herunterladen.



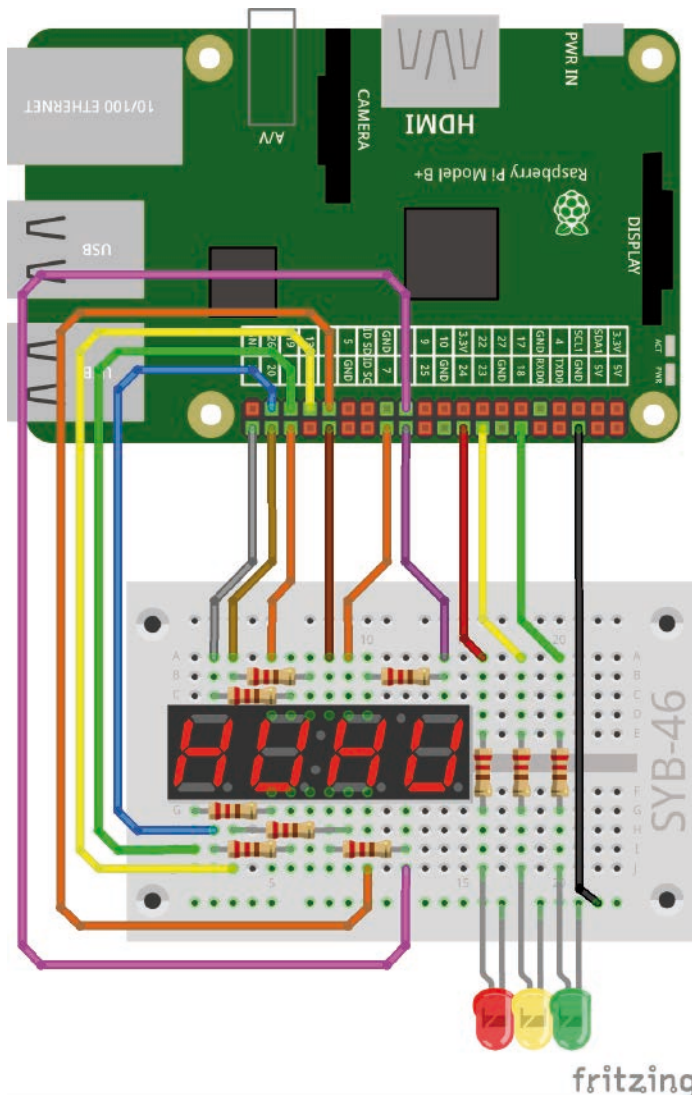
Die Scratch-Blöcke für die schwarze Taste C#

22. Tag

22. Tag

Heute im Adventskalender

• 2 Widerstände 220 Ohm



Sieben-Segment-Anzeige, und drei LEDs.

Das Anschlussschema der Sieben-Segment-Anzeige ist das gleiche wie an den vorherigen Tagen. Der Dezimalpunkt wird nicht verwendet.

Ampel mit Countdown

Das Experiment des 22. Tags simuliert eine Ampel mit drei LEDs. Während der Grünphase zeigt ein Countdown, wie viel Zeit man noch hat.

Bauteile: 1 Steckbrett SYB-46, 1 Sieben-Segment-Anzeige, 1 LED rot, 1 LED gelb, 1 LED grün, 10 220-Ohm-Widerstände (rot-rot-braun), 15 GPIO-Verbindungskabel

Anschlussschema der Sieben-Segment-Anzeige

Die folgende Tabelle zeigt, welche Pins der Sieben-Segment-Anzeige mit welchen GPIO-Pins verbunden sind. Der grau hinterlegte Pin wird in diesem Programm nicht verwendet.

Pin Sieben-Segment-Anzeige	Steckbrett	Segment / Ziffer	GPIO-Pin
1	5 F-J	E	19
2	6 F-J	D	26
3	7 F-J	DP	5
4	8 F-J	C	11
5	9 F-J	G	13
6	10 F-J	4	6
7	10 A-E	B	8
8	9 A-E	3	7
9	8 A-E	2	12
10	7 A-E	F	20
11	6 A-E	A	21
12	5 A-E	1	16

Die LEDs der Ampel sind an den GPIO-Pins 24, 23 und 18 angeschlossen.

Das Programm

Das Scratch-2-Programm `22ampel.sb2` lässt beim Klick auf das grüne Fähnchen den Ampelzyklus laufen. Für den Countdown wird ein selbstdefinierter Block verwendet. Dieser kann auch jederzeit in anderen Programmen zur Darstellung einer Ziffer genutzt werden.

Der Anfang des Programms enthält bekannte Elemente. Nacheinander werden GPIO-Pins initialisiert:

- Die Pins 16, 12, 7 für die Kathoden der ersten drei Ziffern werden auf High gesetzt und diese Ziffern damit ausgeschaltet, die Kathode der vierten Ziffer am Pin 6 wird auf Low gesetzt, um diese Ziffer einzuschalten.
- Die Anoden der sieben Segmente: 21, 8, 11, 26, 19, 20, 13 werden auf Low gesetzt und alle Segmente damit ausgeschaltet.
- Die rote LED am Pin 24 wird eingeschaltet, die gelbe und die grüne LED an den Pins 23 und 18 werden ausgeschaltet.
- Nach einer halben Sekunde Wartezeit wird die gelbe LED am Pin 23 zusätzlich eingeschaltet.
- Nach einer weiteren halben Sekunde werden die rote und die gelbe LED ausgeschaltet und die grüne am Pin 18 eingeschaltet.

Danach wird es interessant.

Eine Variable *z* wird in einer Countdownschleife heruntergezählt. Sie enthält in jedem Durchlauf die anzuzeigende Zahl.

Variablen in Scratch

Variablen sind kleine Speicherplätze, in denen sich ein Programm eine Zahl oder irgendetwas anderes merkt. Wenn das Programm beendet wird, werden diese Variablenspeicher automatisch geleert. Variablen müssen in Scratch erst einmal mit einem Klick auf **Neue Variable** auf der Blockpalette **Daten** angelegt werden, bevor man sie benutzen kann. Anschließend können Sie das Symbol der neu angelegten Variablen aus der Blockpalette in ein dafür vorgesehenes Feld eines Blocks im Programm ziehen. Auf der Blockpalette stehen zusätzlich verschiedene Blöcke zum Auslesen und Verändern der Variablen zur Verfügung.

Legen Sie eine Variable **z** für den Zähler an.



Die neue Variable *z*



Neuen Block definieren

Für die Darstellung einer Ziffer auf der Sieben-Segment-Anzeige verwenden wir einen eigenen, selbstgebauten Block. Eigentlich wäre das für dieses Programm, in dem jede Ziffer nur einmal dargestellt wird, nicht nötig. Das Programm zeigt aber, wie Sie diese Technik in anderen Programmen einsetzen können. Selbstgebaute Blöcke in Scratch sind vergleichbar mit Funktionen in Python, die einmal definiert und dann immer wieder aufgerufen werden können.

Klicken Sie auf der Blockpalette **Weitere Blöcke** auf den Button **Neuer Block**. Jeder neue Block braucht einen Namen. Im Beispiel nennen wir den Block **ziffer**. Bauen Sie in diesen Block ein Zahlenfeld ein. Es soll später beim Aufruf die Ziffer enthalten, die der Block auf der Sieben-Segment-Anzeige anzeigen soll.

Nach einem Klick auf **OK** erscheint ein neuer Block **definiere ziffer number1**, der oben rund ist, also immer als Anfang für eine Folge von Blöcken dient. Der blaue Block **number1** enthält später die Zahl, die dem Block beim Aufruf mitgegeben wird.

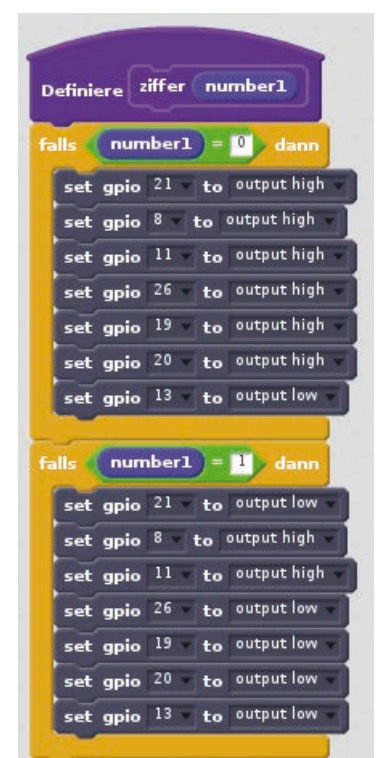
Hängen Sie unter diesen Block zehn **falls ... dann**-Abfragen, die jeweils prüfen, ob die Zahl **number1** gleich 0, 1, 2, 3, usw. ist. In jedem Fall werden die sieben Segmente der Sieben-Segment-Anzeige so geschaltet, dass sie die entsprechende Ziffer darstellen. Die Blöcke zum Schalten der Segmente entsprechen dem Programm des 15. Tags.

Sie können den Block **number1** einfach aus der Blockdefinition in die Felder der **... = ...**-Abfrageblöcke ziehen.

Auf der Blockpalette **Weitere Blöcke** erscheint ein Block **ziffer ...**. Diesen können Sie wie jeden Standardblock im Programm verwenden.



Initialisierung der GPIO-Pins



Selbstgebauten Block definieren



Der neue Block auf der Blockpalette

Im Hauptprogramm wird, nachdem die grüne LED der Ampel eingeschaltet wurde, als erstes der Countdown-Zähler **z** auf 9 gesetzt. Danach startet eine Schleife, die so oft läuft, bis der Zähler **z** kleiner als 0 ist.

In jedem Schleifendurchlauf wird der selbstgebaute Block **ziffer ...** mit der aktuellen Zahl **z** des Countdowns aufgerufen, damit diese Zahl angezeigt wird. Nach einer Wartezeit von 0,2 Sekunden wird die Variable **z** um 1 heruntergezählt, und die Schleife startet den nächsten Durchlauf.



Die Zählschleife im Programm

Nachdem der Countdown bei 0 angekommen ist, wird die gelbe LED der Ampel ein- und die grüne ausgeschaltet. Nach einer weiteren Wartezeit von 0,5 Sekunden wird die rote LED der Ampel ein- und die gelbe ausgeschaltet.

Damit ist das Programm zu Ende und kann mit einem Klick auf das grüne Fähnchen erneut gestartet werden.

```

Wenn angeklickt
  set gpio 16 to output high
  set gpio 12 to output high
  set gpio 7 to output high
  set gpio 6 to output low
  set gpio 21 to output low
  set gpio 8 to output low
  set gpio 11 to output low
  set gpio 26 to output low
  set gpio 19 to output low
  set gpio 20 to output low
  set gpio 13 to output low
  set gpio 24 to output high
  set gpio 23 to output low
  set gpio 18 to output low
  warte 0.5 Sek.
  set gpio 23 to output high
  warte 0.5 Sek.
  set gpio 24 to output low
  set gpio 23 to output low
  set gpio 18 to output high
  setze z auf 0
  wiederhole bis z = 10
    ziffer z
    warte 0.2 Sek.
    andere z um 1
  set gpio 23 to output high
  set gpio 18 to output low
  warte 0.5 Sek.
  set gpio 24 to output high
  set gpio 23 to output low
  
```

```

Definiere ziffer number1
  falls number1 = 0 dann
    set gpio 21 to output high
    set gpio 8 to output high
    set gpio 11 to output high
    set gpio 26 to output high
    set gpio 19 to output high
    set gpio 20 to output high
    set gpio 13 to output low
  falls number1 = 1 dann
    set gpio 21 to output low
    set gpio 8 to output high
    set gpio 11 to output high
    set gpio 26 to output low
    set gpio 19 to output low
    set gpio 20 to output low
    set gpio 13 to output low
  falls number1 = 2 dann
    set gpio 21 to output high
    set gpio 8 to output high
    set gpio 11 to output low
    set gpio 26 to output high
    set gpio 19 to output high
    set gpio 20 to output low
    set gpio 13 to output high
  falls number1 = 3 dann
    set gpio 21 to output high
    set gpio 8 to output high
    set gpio 11 to output high
    set gpio 26 to output high
    set gpio 19 to output low
    set gpio 20 to output low
    set gpio 13 to output high
  falls number1 = 4 dann
    set gpio 21 to output low
    set gpio 8 to output high
    set gpio 11 to output high
    set gpio 26 to output low
    set gpio 19 to output low
    set gpio 20 to output high
    set gpio 13 to output high
  falls number1 = 5 dann
    set gpio 21 to output high
    set gpio 8 to output low
    set gpio 11 to output high
    set gpio 26 to output high
    set gpio 19 to output low
    set gpio 20 to output high
    set gpio 13 to output high
  falls number1 = 6 dann
    set gpio 21 to output high
    set gpio 8 to output low
    set gpio 11 to output high
    set gpio 26 to output high
    set gpio 19 to output high
    set gpio 20 to output high
    set gpio 13 to output high
  falls number1 = 7 dann
    set gpio 21 to output high
    set gpio 8 to output high
    set gpio 11 to output high
    set gpio 26 to output low
    set gpio 19 to output low
    set gpio 20 to output low
    set gpio 13 to output low
  falls number1 = 8 dann
    set gpio 21 to output high
    set gpio 8 to output high
    set gpio 11 to output high
    set gpio 26 to output high
    set gpio 19 to output high
    set gpio 20 to output high
    set gpio 13 to output high
  falls number1 = 9 dann
    set gpio 21 to output high
    set gpio 8 to output high
    set gpio 11 to output high
    set gpio 26 to output high
    set gpio 19 to output low
    set gpio 20 to output high
    set gpio 13 to output high
  
```

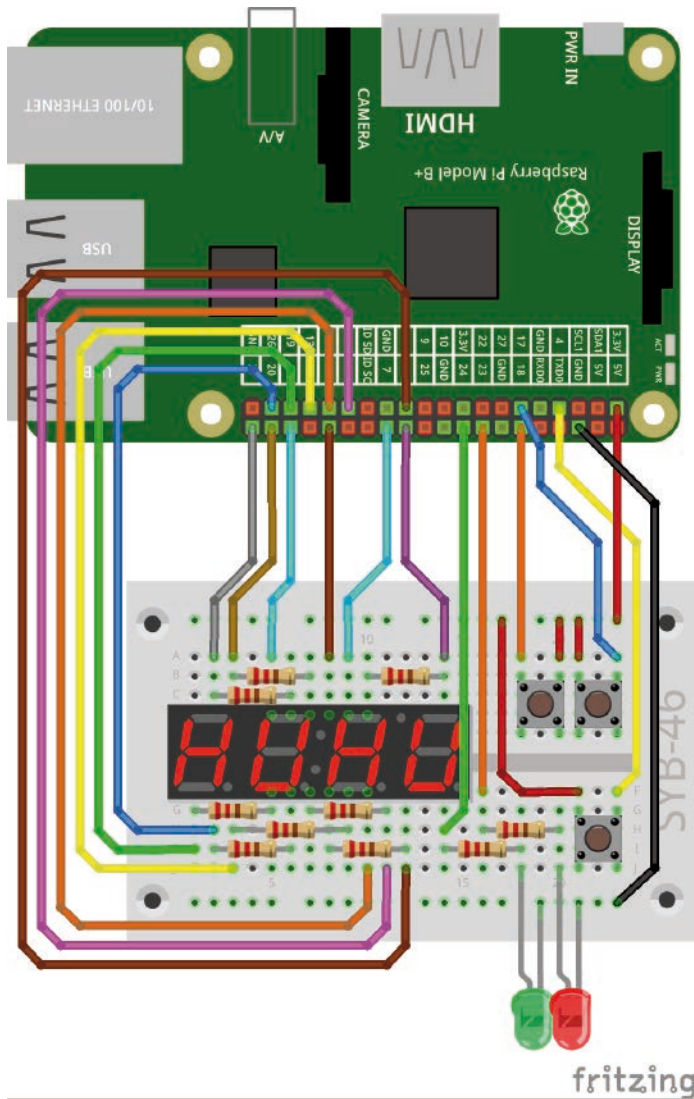
Das ganze Programm im Überblick

23. Tag

Heute im Adventskalender

· 3 GPIO-Verbindungskabel

Das Anschlusschema der Sieben-Segment-Anzeige ist das gleiche wie an den vorherigen Tagen. Der Dezimalpunkt wird auch verwendet.



Sieben-Segment-Anzeige, zwei LEDs und drei Tasten.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time, random

GPIO.setmode(GPIO.BCM)

seg={'a':21, 'b':8, 'c':11, 'd':26, 'e':19, 'f':20, 'g':13}
for s in "abcdefg":
    GPIO.setup(seg[s], GPIO.OUT, initial=0)

zif=[16, 12, 7, 6]
for z in zif:
    GPIO.setup(z, GPIO.OUT, initial=1)

dp=5
GPIO.setup(dp, GPIO.OUT, initial=0)
```

Zahlenraten mit drei Tasten

Das ist ein einfaches Ratespiel, in dem eine vom Computer zufällig gewählte Zahl vom Spieler in möglichst wenigen Schritten erraten werden soll. Die LEDs sind an den GPIO-Pins 24 und 23 angeschlossen, die Taster an den GPIO-Pins 18, 17 und 4.

Bauteile: 1 Steckbrett SYB-46, 1 Sieben-Segment-Anzeige, 1 LED rot, 1 LED grün, 9 220-Ohm-Widerstände (rot-rot-braun), 3 Taster, 18 GPIO-Verbindungskabel, 3 Drahtbrücken (unterschiedliche Längen)

Die LEDs sind an den GPIO-Pins 23 und 24 angeschlossen, die Taster an den Pins 18, 17 und 4.

Das Programm

Das Programm `23spiel.py` generiert eine zufällige Zahl zwischen 0 und 100, die der Spieler mit möglichst wenigen Tipps erraten muss. Mit zwei Tastern stellt der Spieler einen Tipp ein und gibt ihn mit dem dritten Taster ab.

Die beiden linken Ziffern der Anzeige zeigen die getippte Zahl, die beiden Ziffern rechts zeigen an, wie viele Versuche der Spieler bereits hatte. Die beiden LEDs zeigen, ob die gesuchte Zahl kleiner oder größer als der letzte Tipp ist.

Wie entstehen Zufallszahlen?

Gemeinhin denkt man, in einem Programm könne nichts zufällig geschehen – wie also 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. Sie 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 den mehrfachen Aufruf des gleichen Programms jederzeit reproduzieren. Nimmt man aber eine aus einigen dieser Ziffern zusammengebaute Zahl und teilt sie wiederum durch eine Zahl, die sich aus der aktuellen Uhrzeitsekunde oder dem Inhalt einer beliebigen Speicherstelle des Computers ergibt, kommt ein Ergebnis heraus, das sich nicht reproduzieren lässt und daher als Zufallszahl bezeichnet wird.


```

LED1=24
GPIO.setup(LED1, GPIO.OUT, initial=0)
LED2=23
GPIO.setup(LED2, GPIO.OUT, initial=0)

t1=18
t2=17
t3=4
GPIO.setup(t1, GPIO.IN, GPIO.PUD_DOWN)
GPIO.setup(t2, GPIO.IN, GPIO.PUD_DOWN)
GPIO.setup(t3, GPIO.IN, GPIO.PUD_DOWN)

zahl=[
    "abcdef", #0
    "bc", #1
    "abdeg", #2
    "abcdg", #3
    "bcfg", #4
    "acdfg", #5
    "acdefg", #6
    "abc", #7
    "abcdefg", #8
    "abcdfg", #9
]

z=[0,0,0,0]
x=0

print("Taste 1 verringert den Tipp")
print("Taste 2 erhöht den Tipp")
print("Taste 3 gibt den Tipp ab")
print("Strg+C beendet das Programm")

def za(n):
    z[0] = int(n / 1000)
    z[1] = int(n % 1000 / 100)
    z[2] = int(n % 100 / 10)
    z[3] = int(n % 10)
    for i in range(4):
        for s in "abcdefg":
            GPIO.output(seg[s], 0)
            GPIO.output(zif[i], 0)
            for s in zahl[z[i]]:

```

```

            GPIO.output(seg[s], 1)
        if i == 1:
            GPIO.output(dp, 1)
        else:
            GPIO.output(dp, 0)
            time.sleep(0.001)
            GPIO.output(zif[i], 1)

try:
    while True:
        y=random.randrange(100)
        t=0
        while True:
            za(x * 100 + t)
            if GPIO.input(t1)==1 and x>0:
                x-=1
                time.sleep(0.08)
            if GPIO.input(t2)==1 and x<99:
                x+=1
                time.sleep(0.08)
            if GPIO.input(t3)==1:
                t+=1
                if y<x:
                    GPIO.output(LED1, 1)
                    time.sleep(0.5)
                    GPIO.output(LED1, 0)
                if y>x:
                    GPIO.output(LED2, 1)
                    time.sleep(0.5)
                    GPIO.output(LED2, 0)
                if y==x:
                    for i in range(5):
                        GPIO.output(LED1, 1)
                        GPIO.output(LED2, 1)
                        for j in range(100):
                            za(x * 100 + t)
                        GPIO.output(LED1, 0)
                        GPIO.output(LED2, 0)
                        for j in range(100):
                            za(x * 100 + t)
                    break
except KeyboardInterrupt:
    GPIO.cleanup()

```

So funktioniert es

Am Anfang wird zusätzlich die Bibliothek `random` importiert, die Funktionen zum Generieren von Zufallszahlen enthält. Die GPIO-Pins für die Sieben-Segment-Anzeige werden nach bekanntem Schema initialisiert. Der Dezimalpunkt wird zur optischen Trennung der beiden Ziffern links von den beiden Ziffern rechts verwendet. Zusätzlich werden GPIO-Ausgänge für die beiden LEDs und Eingänge mit Pulldown-Widerständen für die drei Taster eingerichtet.

```

x=0

print("Taste 1 verringert den Tipp")
print("Taste 2 erhöht den Tipp")
print("Taste 3 gibt den Tipp ab")
print("Strg+C beendet das Programm")

```

Bevor das Programm startet, wird die Variable `x`, die später die vom Spieler getippte Zahl enthält, auf 0 gesetzt. Danach erscheint auf dem Bildschirm noch eine kurze Erklärung der Tasten.

```

def za(n):
    z[0] = int(n / 1000)
    z[1] = int(n % 1000 / 100)
    z[2] = int(n % 100 / 10)
    z[3] = int(n % 10)
    for i in range(4):
        for s in "abcdefg":
            GPIO.output(seg[s], 0)
        GPIO.output(zif[i], 0)
        for s in zahl[z[i]]:
            GPIO.output(seg[s], 1)
        if i == 1:
            GPIO.output(dp, 1)
        else:
            GPIO.output(dp, 0)
        time.sleep(0.001)
        GPIO.output(zif[i], 1)

```

Die Funktion `za(n)` zur Anzeige einer vierstelligen Zahl ist ähnlich aufgebaut wie im Programm des 20. Tags. Zusätzlich leuchtet der Dezimalpunkt hinter der zweiten Ziffer.

```

try:
    while True:
        y=random.randrange(100)
        t=0

```

Das Hauptprogramm ist eine Endlosschleife, in der zu Beginn jedes Durchlaufs in der Variable `y` eine ganzzahlige Zufallszahl kleiner als 100 gespeichert wird. Diese Zahl muss der Spieler erraten. Zusätzlich wird die Variable `t` auf 0 gesetzt, in der später die Anzahl der Tipps gespeichert wird.

```

    while True:
        za(x * 100 + t)

```

Innerhalb der Schleife, die bei jedem Spiel einmal durchläuft, läuft eine weitere Endlosschleife. Diese zeigt in jedem Durchlauf die zuletzt eingestellte Zahl und die Anzahl der Tipps an. Dazu wird die Zahl `x` mit 100 multipliziert und die Tipps in der Variablen `t` addiert. Vor dem Dezimalpunkt erscheint jetzt die zweistellige Zahl und dahinter ebenfalls zweistellig die Anzahl der Versuche, die der Spieler bereits hatte.

```

    if GPIO.input(t1)==1 and x>0:
        x-=1
        time.sleep(0.08)

```

Drückt der Benutzer die Taste 1 und ist die eingestellte Zahl noch größer als 0, wird die Zahl um 1 verringert. Danach wartet das Programm 0,08 Sekunden, um Mehrfachaktionen durch versehentlich zu langes Drücken der Taste abzufangen. Bei Bedarf können Sie diese Wartezeit anpassen.

```

    if GPIO.input(t2)==1 and x<99:
        x+=1
        time.sleep(0.08)

```

Auf die gleiche Weise erhöht ein Druck auf die zweite Taste die Zahl, wenn diese noch kleiner als 99 ist.

```

    if GPIO.input(t3)==1:
        t+=1

```

Drückt der Spieler die dritte Taste, wird zunächst die Anzahl der Tipps um 1 erhöht. Dann gibt es drei Möglichkeiten:

```

    if y<x:
        GPIO.output(LED1, 1)
        time.sleep(0.5)
        GPIO.output(LED1, 0)

```

• Ist die gesuchte Zahl y kleiner als der abgegebene Tipp, leuchtet die linke von beiden LEDs eine halbe Sekunde lang.

```
if y>x:
    GPIO.output(LED2, 1)
    time.sleep(0.5)
    GPIO.output(LED2, 0)
```

• Ist die gesuchte Zahl y größer als der abgegebene Tipp, leuchtet die rechte von beiden LEDs eine halbe Sekunde lang.

```
if y==x:
    for i in range(5):
        GPIO.output(LED1, 1)
        GPIO.output(LED2, 1)
        for j in range(100):
            za(x * 100 + t)
        GPIO.output(LED1, 0)
        GPIO.output(LED2, 0)
        for j in range(100):
            za(x * 100 + t)
```

• Sind die gesuchte Zahl y und der abgegebene Tipp gleich, lässt eine Schleife beide LEDs fünfmal blinken. Während der Zeit, in der die LEDs leuchten oder ausgeschaltet sind, wird nicht einfach gewartet, sondern einhundert mal hintereinander die Funktion $za(n)$ aufgerufen und damit die richtig getippte Zahl und die Anzahl der Tipps angezeigt. 100 Aufrufe dieser Funktion dauern etwa 0,1 Sekunden.

```
break
```

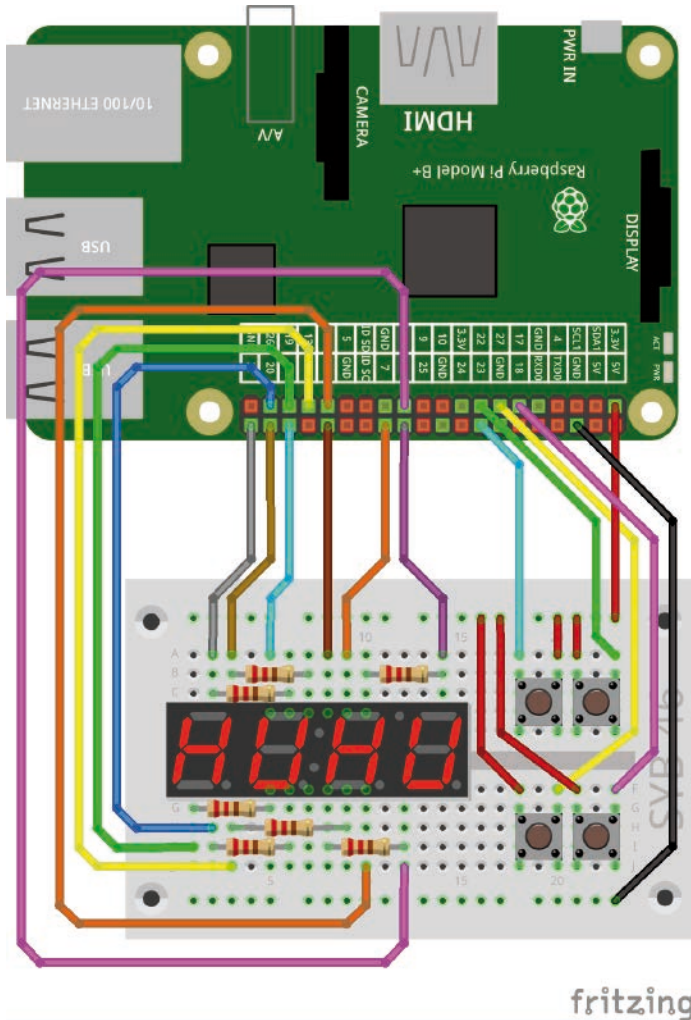
Anschließend wird die innere der beiden Schleifen abgebrochen, da das Spiel gelöst wurde. Die äußere Schleife startet ein neues Spiel, wobei der letzte Tipp gleich als Start für den nächsten Durchlauf verwendet wird. Man muss also nicht mehr von 0 an bis zur gewünschten Zahl hochzählen.

24. Tag

24. Tag

Heute im Adventskalender

- 1 Taster
- Downloadcode



Sieben-Segment-Anzeige und vier Tasten.

Das Anschlusschema der Sieben-Segment-Anzeige ist das gleiche wie an den vorherigen Tagen. Der Dezimalpunkt wird nicht verwendet.

Weihnachtslieder auf dem Raspberry Pi

Das Experiment des 24. Tags spielt beim Drücken einer Taste ein Weihnachtslied. Vier verschiedene Lieder stehen zur Auswahl. Über den Downloadcode im Adventskalender heute können Sie sich Weihnachtslieder im mp3-Format herunterladen. Die Taster sind an den GPIO-Pins 23, 22, 27 und 17 angeschlossen.

Bauteile: 1 Steckbrett SYB-46, 1 Sieben-Segment-Anzeige, 7 220-Ohm-Widerstände (rot-rot-braun), 4 Taster, 18 GPIO-Verbindungskabel, 4 Drahtbrücken (unterschiedliche Längen)

Anschlusschema der Sieben-Segment-Anzeige

Die folgende Tabelle zeigt, welche Pins der 7-Segment-Anzeige mit welchen GPIO-Pins verbunden sind. Der grau hinterlegte Pin wird in diesem Programm nicht verwendet.

Pin Sieben-Segment-Anzeige	Steckbrett	Segment / Ziffer	GPIO-Pin
1	5 F-J	E	19
2	6 F-J	D	26
3	7 F-J	DP	5
4	8 F-J	C	11
5	9 F-J	G	13
6	10 F-J	4	6
7	10 A-E	B	8
8	9 A-E	3	7
9	8 A-E	2	12
10	7 A-E	F	20
11	6 A-E	A	21
12	5 A-E	1	16

Das Programm

Das Programm 24Lieder.py spielt vier verschiedene Weihnachtslieder ab, je nachdem, welche Taste der Benutzer drückt.

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

GPIO.setmode(GPIO.BCM)

seg={'a':21, 'b':8, 'c':11, 'd':26, 'e':19, 'f':20, 'g':13}
for s in "abcdefg":
    GPIO.setup(seg[s], GPIO.OUT, initial=0)

zif=[16, 12, 7, 6]
for z in zif:
    GPIO.setup(z, GPIO.OUT, initial=1)

t1=23
t2=22
t3=27
```

```

t4=17
GPIO.setup(t1, GPIO.IN, GPIO.PUD_DOWN)
GPIO.setup(t2, GPIO.IN, GPIO.PUD_DOWN)
GPIO.setup(t3, GPIO.IN, GPIO.PUD_DOWN)
GPIO.setup(t4, GPIO.IN, GPIO.PUD_DOWN)

zahl=[
    "abcdef", #0
    "bc", #1
    "abdeg", #2
    "abcdg", #3
    "bcfg", #4
    "acdfg", #5
    "acdefg", #6
    "abc", #7
    "abcdefg", #8
    "abcdfg", #9
    ]

print("Strg+C beendet das Programm")

def za(n):
    for s in "abcdefg":
        GPIO.output(seg[s], 0)
    for i in range(4):
        GPIO.output(zif[i], 1)
    if n!=0:
        for s in zahl[n]:
            GPIO.output(seg[s], 1)
        GPIO.output(zif[n-1], 0)

try:
    while True:
        if GPIO.input(t1)==1:
            za(1)
            subprocess.Popen(["omxplayer", "lied1.mp3"])
            time.sleep(4)
            za(0)
        if GPIO.input(t2)==1:
            za(2)
            subprocess.Popen(["omxplayer", "lied2.mp3"])
            time.sleep(4)
            za(0)
        if GPIO.input(t3)==1:
            za(3)
            subprocess.Popen(["omxplayer", "lied3.mp3"])
            time.sleep(4)
            za(0)
        if GPIO.input(t4)==1:
            za(4)
            subprocess.Popen(["omxplayer", "lied4.mp3"])
            time.sleep(4)
            za(0)

except KeyboardInterrupt:
    GPIO.cleanup()

```

So funktioniert es

Zusätzlich zu den bereits bekannten Bibliotheken wird die Bibliothek `subprocess` importiert. Damit ist es möglich, Linux-Kommandozeilenprogramme aus einem Python-Programm heraus zu starten.

```
def za(n):
    for s in "abcdefg":
        GPIO.output(seg[s], 0)
    for i in range(4):
        GPIO.output(zif[i], 1)
    if n!=0:
        for s in zahl[n]:
            GPIO.output(seg[s], 1)
        GPIO.output(zif[n-1], 0)
```

Die Sieben-Segment-Anzeige soll die Nummer des Liedes anzeigen. Dabei ist die verwendete Funktion `za(n)` so verändert, dass die 1 auf der ersten Ziffer, die 2 auf der zweiten usw. angezeigt werden. Das ist nicht unbedingt nötig, sondern nur ein grafischer Effekt. Um die Anzeige komplett auszuschalten, übergibt man eine 0 beim Funktionsaufruf.

```
try:
    while True:
        if GPIO.input(t1)==1:
            za(1)
            subprocess.Popen(["omxplayer", "lied1.mp3"])
            time.sleep(4)
            za(0)
```

Die Hauptschleife des Programms fragt nacheinander alle vier Tasten ab. Wurde die Taste 1 gedrückt, zeigt die Anzeige die Ziffer 1 an. Dann wird das Lied `lied1.mp3` abgespielt. Dazu wird die Funktion `subprocess.Popen()` verwendet, die in diesem Fall den kommandozeilenbasierten Medienplayer `omxplayer` aufruft.

Nach vier Sekunden schaltet sich die Anzeige aus. Der Medienplayer läuft in einem eigenen Prozess. Das Lied kann also durchaus noch länger laufen. Das Programm wartet nicht. Auf die gleiche Weise lassen sich auch die anderen Lieder abspielen.

Durch Umbenennen der Dateien können Sie auch andere Lieder abspielen lassen. Der Download enthält mehr als nur vier Weihnachtslieder.

Frohe Weihnachten!

Vorsichtsmaßnahmen

Auf keinen Fall irgendwelche GPIO-Pins miteinander verbinden und abwarten, was passiert.

Nicht alle GPIO-Pins lassen sich frei programmieren. Einige sind für die Stromversorgung und andere Zwecke fest eingerichtet.

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 Schutzwiderstand dazwischengeschaltet werden.

Für Logiksignale immer Pin 1 verwenden, der +3,3 V liefert und bis 50 mA belastet werden kann. Pin 6 ist die Masseleitung für Logiksignale.

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

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.

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.

MAKERFACTORY

distributed by Conrad Electronic SE
Klaus-Conrad-Str. 1, 92240 Hirschau

www.makerfactory.com

© 2019 Franzis Verlag GmbH, Richard-Reitzner-Allee 2, 85540 Haar

Alle Rechte vorbehalten, auch die der fotomechanischen Wiedergabe und der Speicherung in elektronischen Medien. Das Erstellen und Verbreiten von Kopien auf Papier, auf Datenträgern oder im Internet, insbesondere als PDF, ist nur mit ausdrücklicher Genehmigung des Verlags gestattet und wird widrigenfalls strafrechtlich verfolgt.

Die meisten Produktbezeichnungen von Hard- und Software sowie Firmennamen und Firmenlogos, die in diesem Werk genannt werden, sind in der Regel gleichzeitig auch eingetragene Warenzeichen und sollten als solche betrachtet werden. Der Verlag folgt bei den Produktbezeichnungen im Wesentlichen den Schreibweisen der Hersteller.

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.