CE

Inhaltsverzeichnis

1 Von der Betriebssysteminstallation bis zum ersten Python-Programm

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

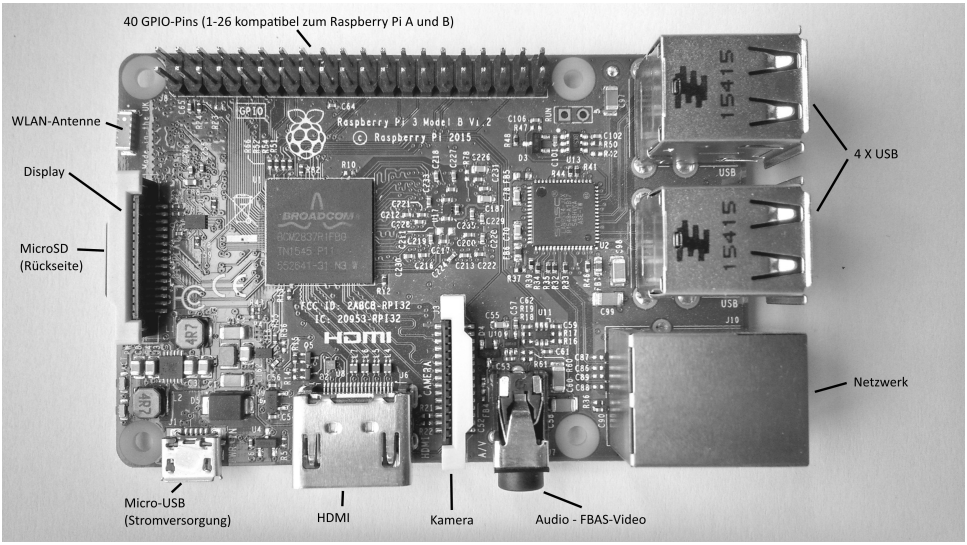

Abb. 1.1: Der Raspberry Pi und seine Hardwareanschlüsse

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

1.1 Was braucht man?

Der Raspberry Pi ist trotz seiner winzigen Größe ein vollwertiger Computer. Um ihn nutzen zu können, braucht man wie bei einem »normalen« PC noch einiges an Zubehör – ein Betriebssystem, Stromversorgung, Netzwerk, Monitor, Tastatur und diverse Anschlusskabel.

1.1.1 Micro-USB-Handyladegerät

Für den Raspberry Pi reicht jedes moderne Handynetzteil. Ältere Ladegeräte aus den Anfangszeiten der USB-Ladetechnik sind noch zu schwach. Schließt man leistungshungrige USB-Geräte wie externe Festplatten ohne eigene Stromversorgung an, ist ein stärkeres Netzteil erforderlich. Das Netzteil muss 5 V und mindestens 2.000 mA. Der eingebaute Leistungsregler verhindert ein »Durchbrennen« bei zu starken Netzteilen.

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

1.1.2 Speicherkarte

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

1.1.3 Tastatur

Jede gängige Tastatur mit USB-Anschluss kann genutzt werden. Kabellose Tastaturen funktionieren manchmal nicht, da sie zu viel Strom oder spezielle Treiber benötigen. Haben Sie keine andere Tastatur zur Verfügung, benötigen Sie einen USB-Hub mit separater Stromversorgung zum Betrieb einer Funktastatur.

1.1.4 Maus

Eine Maus mit USB-Anschluss wird nur benötigt, wenn man auf dem Raspberry Pi ein Betriebssystem mit grafischer Benutzeroberfläche verwendet, wie auch für die Experimente in diesem Lernpaket.

1.1.5 Netzwerkkabel

Zur Verbindung mit dem Router im lokalen Netzwerk wird ein Netzwerkkabel benötigt. Der Raspberry Pi 3 verfügt dafür auch über ein eingebautes WLAN-Modul. Ohne Internetzugang sind viele Funktionen des Raspberry Pi nicht sinnvoll nutzbar.

1.1.6 HDMI-Kabel

Der Raspberry Pi kann per HDMI-Kabel an Monitoren oder Fernsehern angeschlossen werden. Zum Anschluss an Computermonitoren mit DVI-Anschluss gibt es spezielle HDMI-Kabel oder Adapter.

1.1.7 Audiokabel

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

1.1.8 Gelbes FBAS-Videokabel

Steht kein HDMI-Monitor zur Verfügung, kann der Raspberry Pi mit einem analogen Videokabel an einen klassischen Fernseher angeschlossen werden, wobei die Bildschirmauflösung allerdings sehr gering ist. Der analoge Video-Ausgang ist mit dem Audio-Ausgang kombiniert. Zum Anschluss an die typische gelbe Buchse

eines Fernsehers ist ein Adapterkabel mit einem 3,5-mm-Klinkenstecker erforderlich. Für Fernseher ohne gelben FBAS-Eingang gibt es Adapter von FBAS auf SCART. Die grafische Oberfläche lässt sich in analoger Fernsehauflösung nur mit Einschränkungen bedienen.

1.2 Raspbian-Betriebssystem installieren

Der Raspberry Pi wird ohne Betriebssystem ausgeliefert. Anders als bei PCs, die fast alle Windows verwenden, empfiehlt sich für den Raspberry Pi ein speziell angepasstes Linux. Windows würde auf der sparsamen Hardware gar nicht laufen.

Raspbian heißt die Linux-Distribution, die vom Hersteller des Raspberry Pi empfohlen und unterstützt wird. Raspbian basiert auf Debian-Linux, einer der bekanntesten Linux-Distributionen, die unter anderem Grundlage der populären Linux-Varianten Ubuntu und Knoppix ist. Was bei PCs die Festplatte ist, ist beim Raspberry Pi eine Speicherkarte. Auf dieser befinden sich das Betriebssystem und die Daten, von dieser Speicherkarte bootet der Raspberry Pi auch.

1.2.1 Speicherkarte im PC vorbereiten

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

Verwenden Sie am besten fabrikneue Speicherkarten, da diese vom Hersteller bereits optimal vorformatiert sind. Sie können aber auch eine Speicherkarte verwenden, die vorher bereits in einer Digitalkamera oder einem Smartphone genutzt wurde. Diese Speicherkarten sollten vor der Verwendung für den Raspberry Pi neu formatiert werden. Theoretisch können Sie dazu die Formatierungsfunktionen von Windows verwenden. Deutlich besser ist die Software »SDFormatter« der SD Association. Damit werden die Speicherkarten für optimale Performance formatiert. Dieses Tool können Sie sich bei *www.sdcard.org/downloads/formatter_4* kostenlos herunterladen.

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

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

1.2.2 Der Software-Installer NOOBS

»New Out Of Box Software« (NOOBS) ist ein besonders einfacher Installer für Raspberry-Pi-Betriebssysteme. Hier braucht sich der Benutzer nicht mehr wie früher selbst mit Image-Tools und Bootblöcken auseinanderzusetzen, um eine bootfähige Speicherkarte einzurichten. NOOBS bietet verschiedene Betriebssysteme zur Auswahl, wobei man beim ersten Start direkt auf dem Raspberry Pi das gewünschte Betriebssystem auswählen kann, das dann bootfähig auf der Speicherkarte installiert wird. Laden Sie sich das etwa 1,2 GB große

Installationsarchiv für NOOBS auf der offiziellen Downloadseite *www.raspberrypi.org/downloads* herunter und entpacken Sie es am PC auf eine mindestens 4 GB große Speicherkarte. Die Experimente in diesem Lernpaket wurden mit der NOOBS-Version 1.9.2 getestet. Ältere Versionen sind zu aktuellen Raspberry-Pi-Modellen nur eingeschränkt kompatibel.

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

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

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

1.2.3 Der erste Start auf dem Raspberry Pi

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

1.3 Fast wie Windows – die grafische Oberfläche LXDE

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

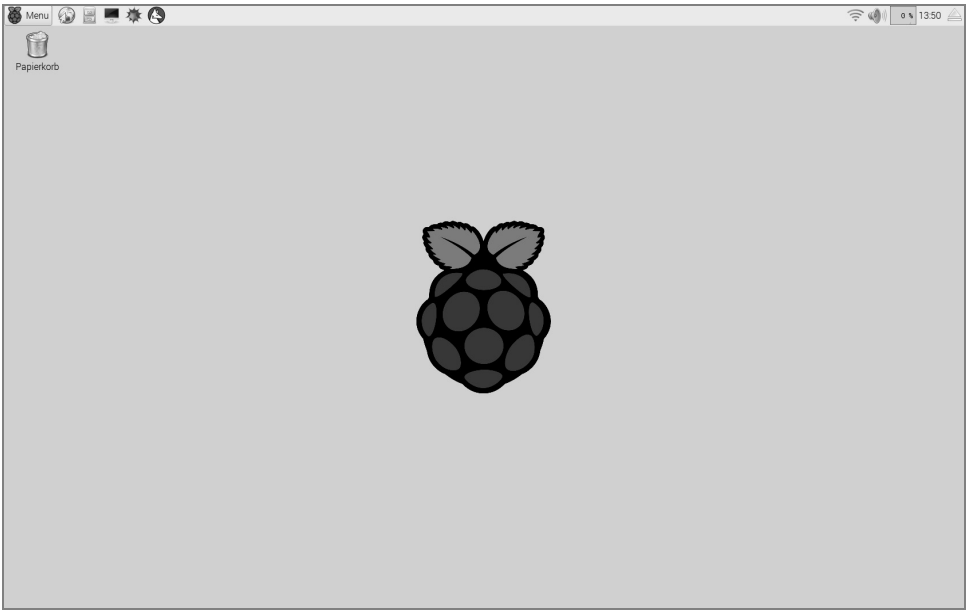

Abb. 1.2: Der LXDE-Desktop auf dem Raspberry Pi

Das Raspbian-Linux für den Raspberry Pi verwendet die Oberfläche LXDE (Lightweight X11 Desktop Environment), die einerseits sehr wenig Systemressourcen benötigt und andererseits mit ihrem Startmenü und dem Dateimanager der gewohnten Windows-Oberfläche sehr ähnelt – mit dem Unterschied, dass die Taskleiste am oberen Bildschirmrand liegt.

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

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

Raspberry Pi ausschalten
Theoretisch kann man bei dem Raspberry Pi einfach den Stecker ziehen, und er schaltet sich ab. Besser ist es jedoch, das System wie auf einem PC sauber herunterzufahren. Wählen Sie dazu im Menü Shutdown.

1.3.1 Uhrzeit einstellen

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

Wählen Sie den Punkt *Einstellungen/Raspberry Pi Confi guration.* Dieses Dialogfeld ersetzt das textbasierte Konfi gurationstool früherer Raspbian-Versionen. Klicken Sie auf der Registerkarte *Localisation* auf den Button *Set Timezone* und wählen Sie die hierzulande verwendete Zeitzone Europa/Berlin.

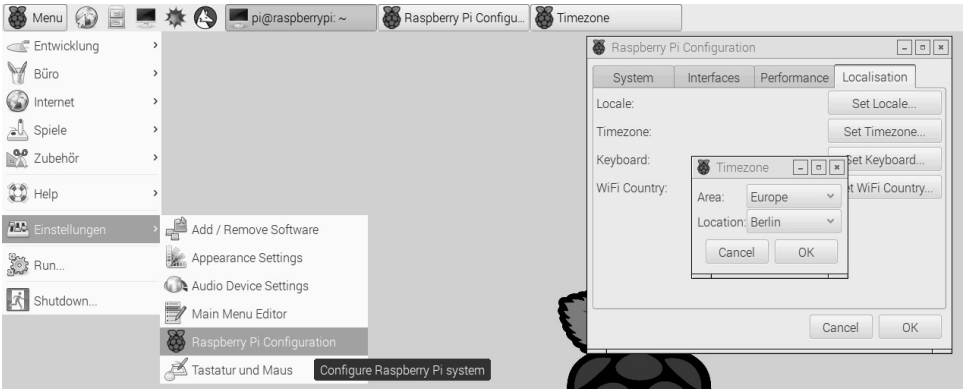

Abb. 1.3: Zeitzone einstellen

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

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

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

```
Di 22. Mär 08:00:00 CET 2016
```

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

1.3.2 Eigene Dateien auf dem Raspberry Pi speichern

Die Dateiverwaltung läuft unter Linux zwar etwas anders als unter Windows, ist aber auch nicht schwieriger. Raspbian bringt einen Dateimanager mit, der dem Windows-Explorer täuschend ähnlich sieht. Ein wichtiger Unterschied zu Windows: Linux trennt nicht strikt nach Laufwerken, alle Dateien befinden sich in einem gemeinsamen Dateisystem.

Unter Linux legt man alle eigenen Dateien grundsätzlich nur unterhalb des eigenen Home-Verzeichnisses ab. Hier heißt es `/home/pi` nach dem Benutzernamen `pi`. Linux verwendet den einfachen Schrägstrich zur Trennung von Verzeichnisebenen (`/`), nicht den von Windows bekannten Backslash (`\`). In diesem Verzeichnis werden Sie auch Ihre Python-Programme ablegen.

Der Dateimanager zeigt standardmäßig auch nur dieses Home-Verzeichnis an. Einige Programme legen dort automatisch Unterverzeichnisse an.

Wer wirklich alles sehen möchte, auch die Dateien, die den normalen Benutzer nichts angehen, schaltet den Dateimanager oben links von *Orte* auf *Verzeichnisbaum* um. Dann noch im Menü unter *Ansicht* die Option *Detailansicht* gewählt, und die Anzeige sieht so aus, wie man sie sich bei Linux vorstellt.

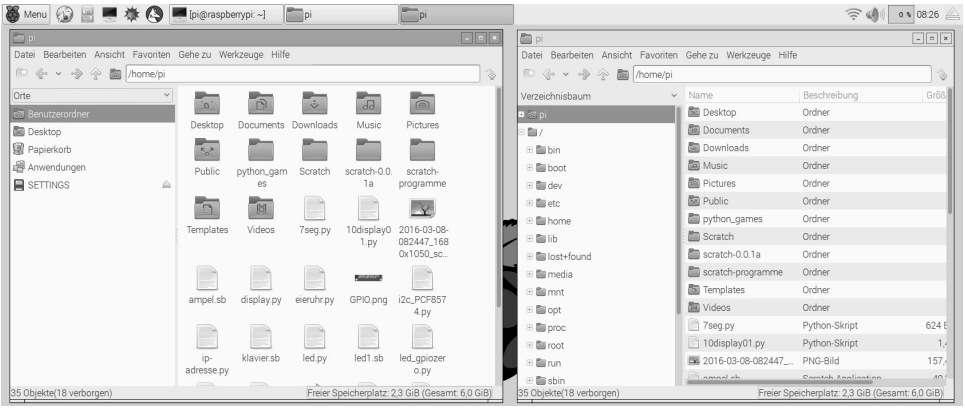

Abb. 1.4: Der Dateimanager auf dem Raspberry Pi in zwei verschiedenen Darstellungen.

Wie viel Platz ist auf der Speicherkarte frei?
Nicht nur Festplatten von PCs sind ständig voll – bei der Speicherkarte des Raspberry Pi kann das noch viel schneller gehen. Umso wichtiger ist es, den freien und insgesamt verfügbaren Platz auf der Speicherkarte immer im Blick zu haben. Die Statuszeile des Dateimanagers am unteren Fensterrand zeigt rechts den freien und belegten Speicherplatz auf der Speicherkarte.

1.4 Das erste Programm mit Python

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

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

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

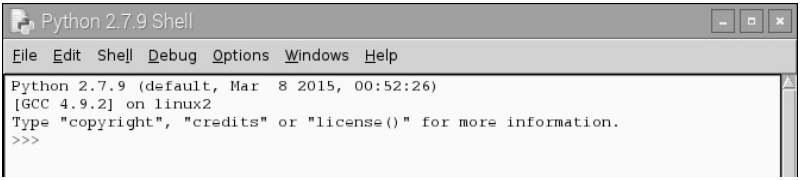

Abb. 1.5: Das Eingabefenster der Python-Shell.

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

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

Es erscheint sofort die richtige Antwort:

```
3
```

Auf diese Weise lässt sich Python als komfortabler Taschenrechner verwenden, was aber noch nichts mit Programmierung zu tun hat. Üblicherweise fangen Programmierkurse mit einem *Hallo Welt*-Programm an, das auf den Bildschirm den Satz »Hallo Welt« schreibt. Das ist in Python derart einfach, dass es sich nicht einmal lohnt, dafür eine eigene Überschrift einzufügen. Tippen Sie im Python-Shell-Fenster einfach folgende Zeile ein:

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

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

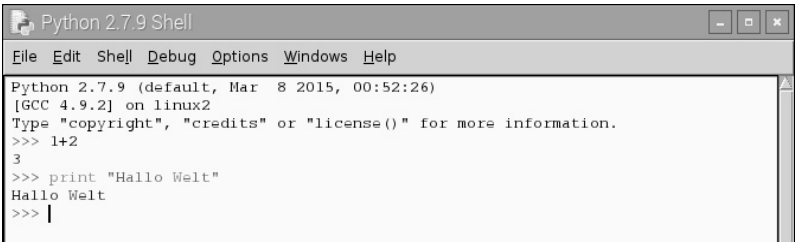

Abb. 1.6: »Hallo Welt« in Python (oberhalb ist noch die Ausgabe der Berechnung zu sehen).

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

Python-Flashcards

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

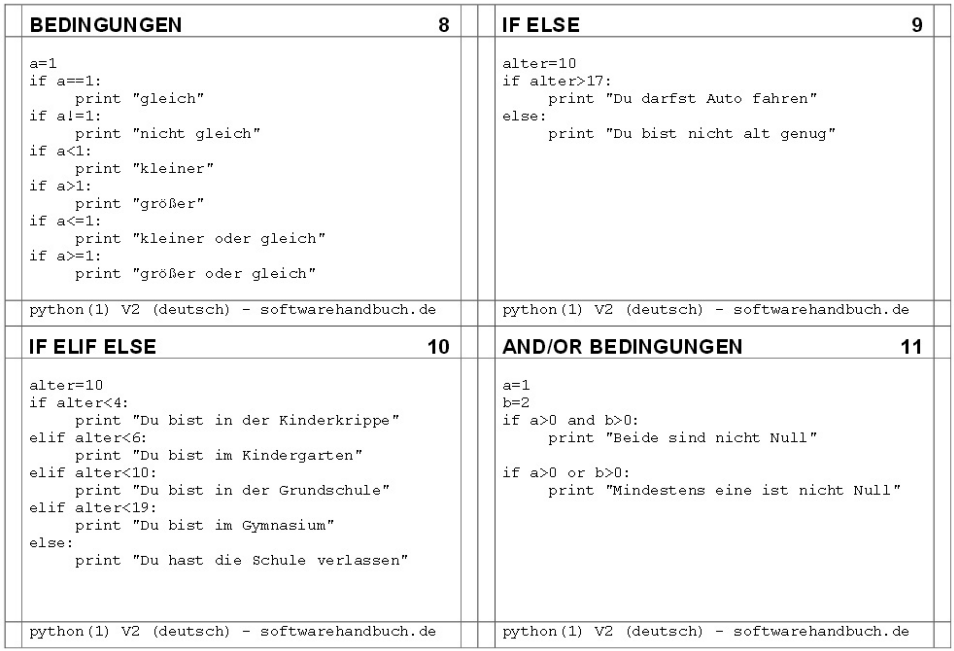

Abb. 1.7: Ausschnitt aus den Python-Flashcards.

1.4.1 Zahlenraten mit Python

Anstatt uns mit Programmiertheorie, Algorithmen und Datentypen aufzuhalten, schreiben wir gleich das erste kleine Spiel in Python, ein einfaches Ratespiel, in dem eine vom Computer zufällig gewählte Zahl vom Spieler in möglichst wenigen Schritten erraten werden soll.

1. Wählen Sie im Menü der Python-Shell *File/New Window*. Hier öffnet sich ein neues Fenster, in das Sie den folgenden Programmcode eintippen:

```
import random
zahl = random.randrange(1000); tipp = 0; i = 0
while tipp != zahl:
    tipp = input("Dein Tipp:")
    if zahl < tipp:
        print "Die gesuchte Zahl ist kleiner als ",tipp

    if zahl > tipp:
        print "Die gesuchte Zahl ist größer als ",tipp

    i += 1
print "Du hast die Zahl beim ",i,". Tipp erraten"
```

2. Speichern Sie die Datei über *File/Save As* als `spiel1.py` ab. Oder Sie laden sich die fertige Programmdatei bei *www.buch.cd* herunter und öffnen sie in der Python-Shell mit *File/Open*. Die Farbcodierung im Quelltext erscheint automatisch und hilft dabei, Tippfehler zu finden.

3. Bevor Sie das Spiel starten, müssen Sie noch eine Besonderheit der deutschen Sprache berücksichtigen, nämlich die Umlaute. Python läuft auf verschiedensten Computerplattformen, die Umlaute unterschiedlich codieren. Damit sie richtig dargestellt werden, wählen Sie im Menü *Options/Configure IDLE* und schalten auf der Registerkarte *General* die Option *Locale-defined* im Bereich *Default Source Encoding* ein.

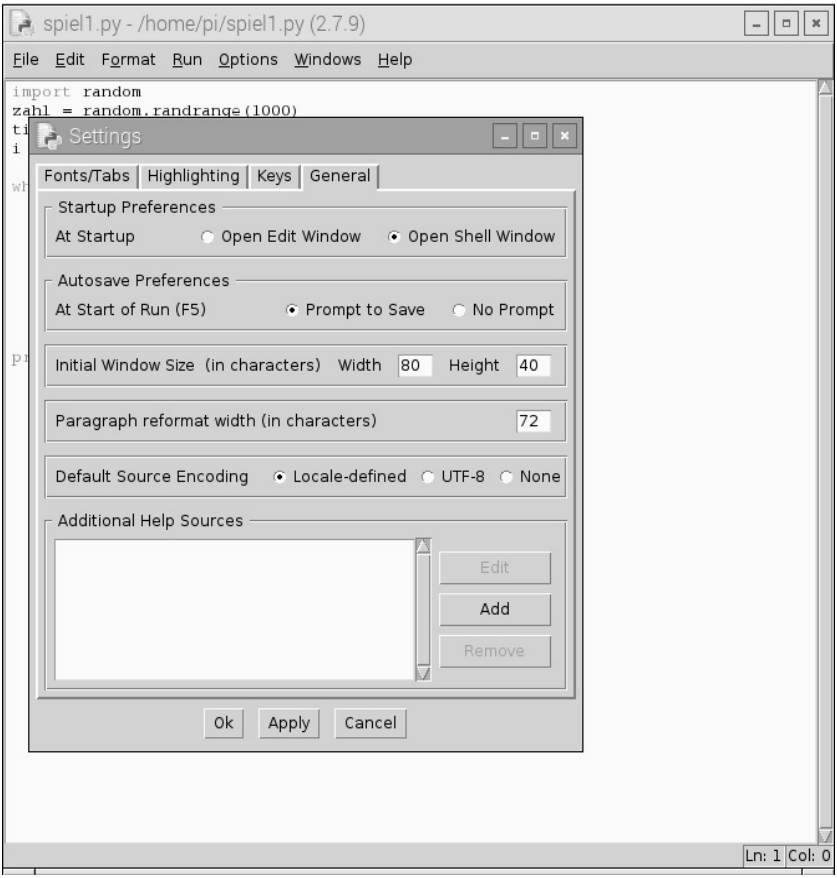

Abb. 1.8: Die richtige Einstellung zur Darstellung von Umlauten in Python.

4. Starten Sie jetzt das Spiel mit der Taste F5 oder dem Menüpunkt *Run/Run Module*.

5. Das Spiel verzichtet der Einfachheit halber auf jede grafische Oberfläche sowie auf erklärende Texte oder Plausibilitätsabfragen der Eingabe. Im Hintergrund generiert der Computer eine Zufallszahl zwischen 0 und 1.000. Geben Sie einfach einen Tipp ab, und Sie erfahren, ob die gesuchte Zahl größer oder kleiner ist. Mit weiteren Tipps tasten Sie sich an die richtige Zahl heran.

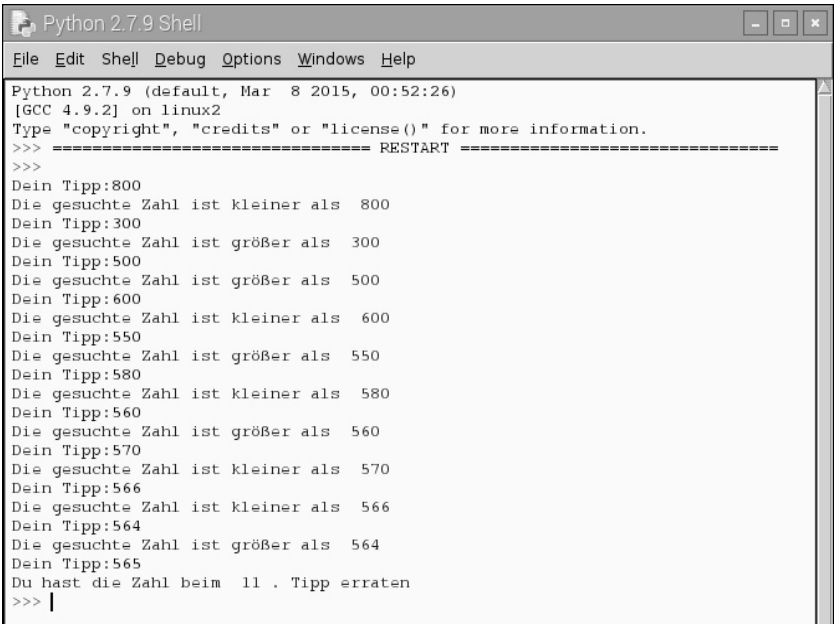

Abb. 1.9: Zahlenraten in Python.

1.4.2 So funktioniert es

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

`import random` Um die zufällige Zahl zu generieren, wird ein externes Python-Modul namens `random` importiert, das diverse Funktionen für Zufallsgeneratoren enthält.

`zahl = random.randrange(1000)` Die Funktion `randrange` aus dem Modul `random` generiert eine Zufallszahl in dem durch die Parameter begrenzten Zahlenbereich, hier zwischen 0 und 999. Der Parameter der Funktion `random.randrange()` gibt die Anzahl möglicher Zufallszahlen an, mit 0 beginnend also immer die erste Zahl, die nicht erreicht wird. Das Gleiche gilt auch für Schleifen und ähnliche Funktionen in Python.

Diese Zufallszahl wird in der Variablen `zahl` gespeichert. Variablen sind in Python Speicherplätze, die einen beliebigen Namen haben und Zahlen, Zeichenfolgen, Listen oder andere Datenarten speichern können. Anders als in einigen anderen Programmiersprachen müssen sie nicht vorher deklariert werden.

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. Diese ändern sich auch ohne jede Regelmäßigkeit, wenn man den Divisor regelmäßig erhöht. Dieses Ergebnis ist zwar scheinbar zufällig, lässt sich aber durch ein identisches Programm oder den mehrfachen Aufruf des gleichen Programms jederzeit reproduzieren. Nimmt man jetzt aber eine aus einigen dieser Ziffern zusammengebaute Zahl und teilt sie wiederum durch eine Zahl, die sich aus der aktuellen Uhrzeitsekunde oder dem Inhalt einer beliebigen Speicherstelle des Rechners ergibt, kommt ein Ergebnis heraus, das sich nicht reproduzieren lässt und daher als Zufallszahl bezeichnet wird.

`tipp = 0` Die Variable `tipp` enthält später die Zahl, die der Benutzer tippt. Am Anfang ist sie `0`.

`i = 0` Die Variable `i` hat sich unter Programmierern als Zähler für Programmschleifendurchläufe eingebürgert. Hier wird sie verwendet, um die Anzahl der Tipps zu zählen, die der Benutzer brauchte, um die geheime Zahl zu erraten. Auch diese Variable steht am Anfang auf `0`.

`while tipp != zahl:` Das Wort `while` (englisch für »so lange wie«) leitet eine Programmschleife ein, die in diesem Fall so lange wiederholt wird, wie `tipp` (die Zahl, die der Benutzer tippt) ungleich der geheimen Zahl `zahl` ist. Python verwendet die Zeichenkombination `!=` für ungleich. Hinter dem Doppelpunkt folgt die eigentliche Programmschleife.

`tipp = input("Dein Tipp:")` Die Funktion `input` schreibt den Text `Dein Tipp:` und erwartet danach eine Eingabe, die in der Variablen `tipp` gespeichert wird.

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

`if zahl < tipp:` Wenn die geheime Zahl `zahl` kleiner ist als die vom Benutzer getippte Zahl `tipp`, dann ...

```
        print "Die gesuchte Zahl ist kleiner als ",tipp
```

... wird dieser Text ausgegeben. Am Ende steht hier die Variable `tipp`, damit die getippte Zahl im Text angezeigt wird. Trifft diese Bedingung nicht zu, wird die eingerückte Zeile einfach übergangen.

`if tipp < zahl`: Wenn die geheime Zahl `zahl` größer ist als die vom Benutzer getippte Zahl `tipp`, dann ...

```
        print "Die gesuchte Zahl ist größer als ",tipp
```

... wird ein anderer Text ausgegeben.

`i += 1` In jedem Fall – deshalb nicht mehr eingerückt – wird der Zähler `i`, der die Versuche zählt, um 1 erhöht. Diese Zeile mit dem Operator `+=` bedeutet das Gleiche wie `i = i + 1`.

```
print "Du hast die Zahl beim ",i,". Tipp erraten"
```

Diese Zeile ist mehr eingerückt, was bedeutet, dass auch die `while`-Schleife zu Ende ist. Trifft deren Bedingung nicht mehr zu, ist also die vom Benutzer getippte Zahl `tipp` nicht mehr ungleich (sondern gleich) der geheimen Zahl `zahl`, wird dieser Text ausgegeben, der sich aus zwei Satzteilen und der Variablen `i` zusammensetzt und so angibt, wie viele Versuche der Benutzer benötigte. Python-Programme brauchen keine eigene Anweisung zum Beenden. Sie enden einfach nach dem letzten Befehl bzw. nach einer Schleife, die nicht mehr ausgeführt wird und der keine weiteren Befehle folgen.

2 Die erste LED leuchtet am Raspberry Pi

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

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

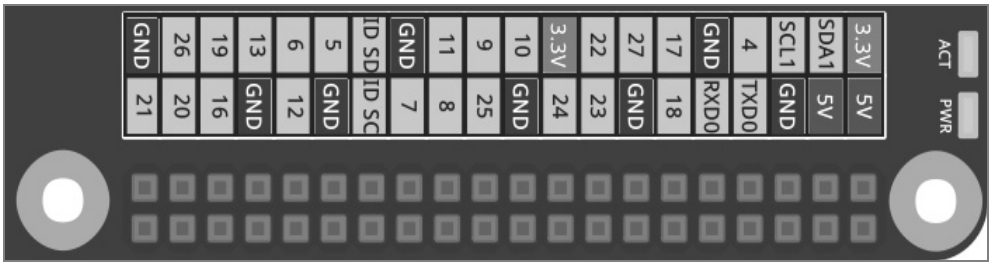

Abb. 2.1: Belegung der GPIO-Schnittstelle. Die abgerundete Ecke unten rechts entspricht der Ecke der Raspberry-Pi-Platine.

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

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

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

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

Pin 2 und Pin 4 liefert +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.

2.1 Bauteile im Paket

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

- 2x Steckplatine
- 1x LED rot
- 1x LED gelb
- 1x LED grün
- 1x LED blau
- 4x Taster
- 4x Widerstand 10 kOhm (Braun-Schwarz-Orange)
- 4x Widerstand 1 kOhm (Braun-Schwarz-Rot)
- 4x Widerstand 220 Ohm (Rot-Rot-Braun)
- 12x Verbindungskabel (Steckplatine – Raspberry Pi)

- ca. 1 m Schaltdraht

2.1.1 Steckplatinen

Für den schnellen Aufbau elektronischer Schaltungen sind zwei Steckplatinen im Paket enthalten. Hier können elektronische Bauteile direkt in ein Lochraster mit Standardabständen eingesteckt werden, ohne dass man löten muss. Bei diesen Platinen sind die äußeren Längsreihen mit Kontakten (X und Y) alle miteinander verbunden.

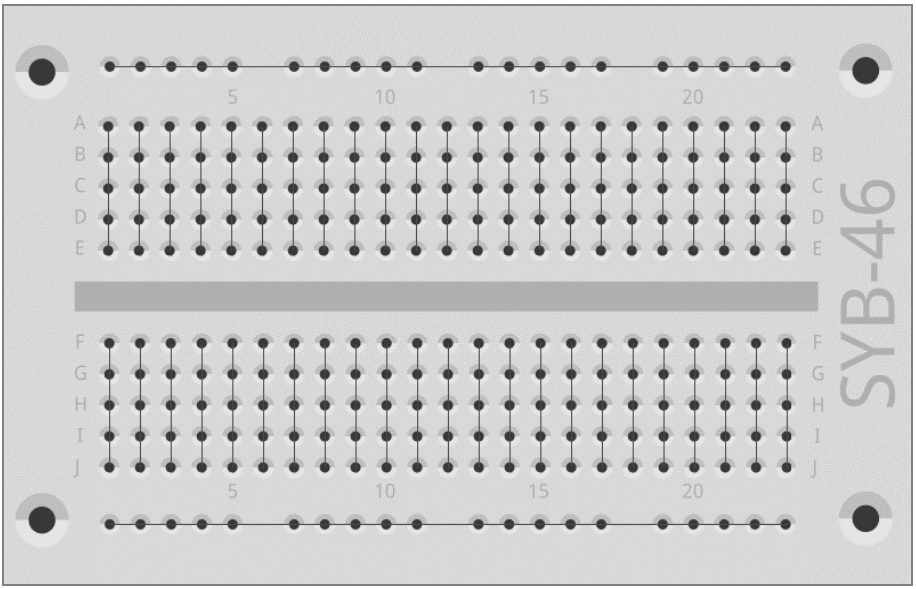

Abb. 2.2: Die Steckplatine aus dem Paket mit ihren eingebauten Verbindungen.

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

2.1.2 Verbindungskabel

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

Außerdem ist Schaltdraht im Lernpaket enthalten. Damit stellen Sie kurze Verbindungsbrücken her, mit denen Kontaktreihen auf der Steckplatine verbunden werden. Schneiden Sie den Draht mit einem kleinen Seitenschneider in den passenden Längen ab, wie bei den einzelnen Experimenten beschrieben. Um die Drähte besser in die Steckplatine stecken zu können, empfiehlt es sich, die Drähte leicht schräg abzuschnei-

den, sodass eine Art Keil entsteht. Entfernen Sie an beiden Enden auf einer Länge von etwa einem halben Zentimeter die Isolierung.

2.1.3 Widerstände und ihre Farbcodes

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

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

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

Tab. 2.1: Die Tabelle zeigt die Bedeutung der farbigen Ringe auf Widerständen.

Im Lernpaket sind Widerstände in drei verschiedenen Werten enthalten:

Wert	1. Ring (Zehner)	2. Ring (Einer)	3. Ring (Multipl.)	4. Ring (Toleranz)	Verwendung
220 Ohm	Rot	Rot	Braun	Gold	Vorwiderstand für LEDs
1 kOhm	Braun	Schwarz	Rot	Gold	Schutzwiderstand für GPIO-Eingänge
10 kOhm	Braun	Schwarz	Orange	Gold	Pull-down-Widerstand für

					GPIO-Eingänge

Tab. 2.2: Farbcodes der Widerstände im Lernpaket.

Achten Sie besonders bei den 1-kOhm- und 10-kOhm-Widerständen genau auf die Farben. Diese sind leicht zu verwechseln.

2.2 LED anschließen

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

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

Schließen Sie als Erstes wie auf dem Bild eine LED über einen 220-Ohm-Vorwiderstand (Rot-Rot-Braun) am +3,3-V-Anschluss (Pin 1) an und verbinden Sie den Minuspol der LED mit der Masseleitung (Pin 6).

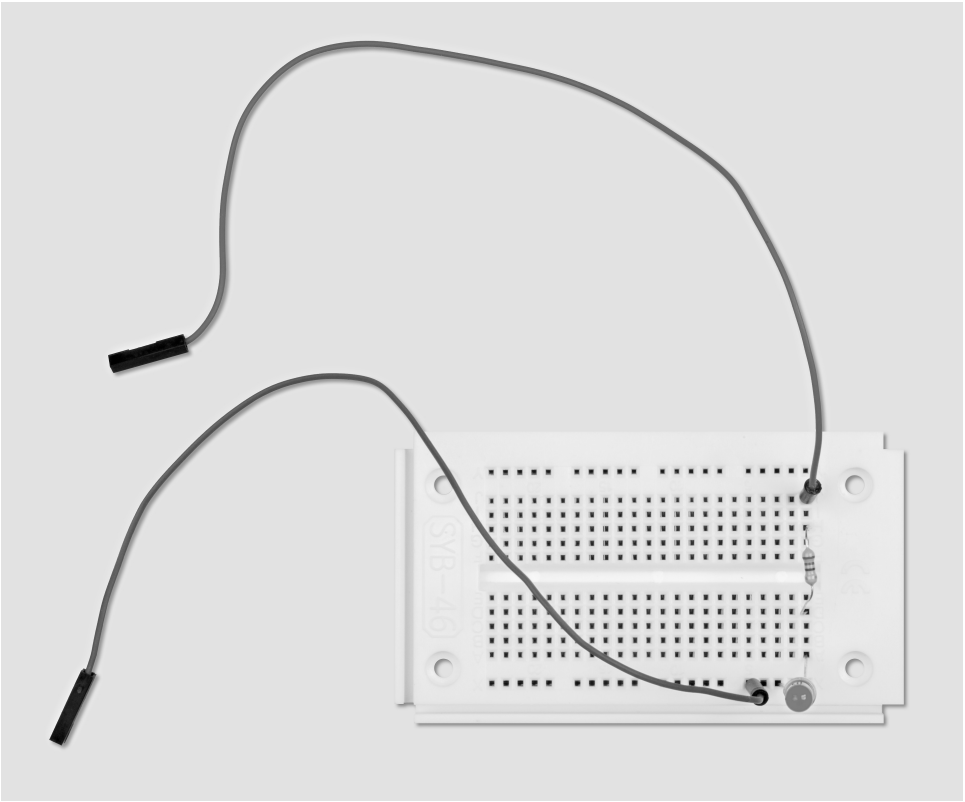

Abb. 2.3: Steckbrettaufbau, um eine LED anzuschließen.

Benötigte Bauteile:
1x Steckplatine
1x LED rot
1x 220-Ohm-Widerstand
2x Verbindungskabel

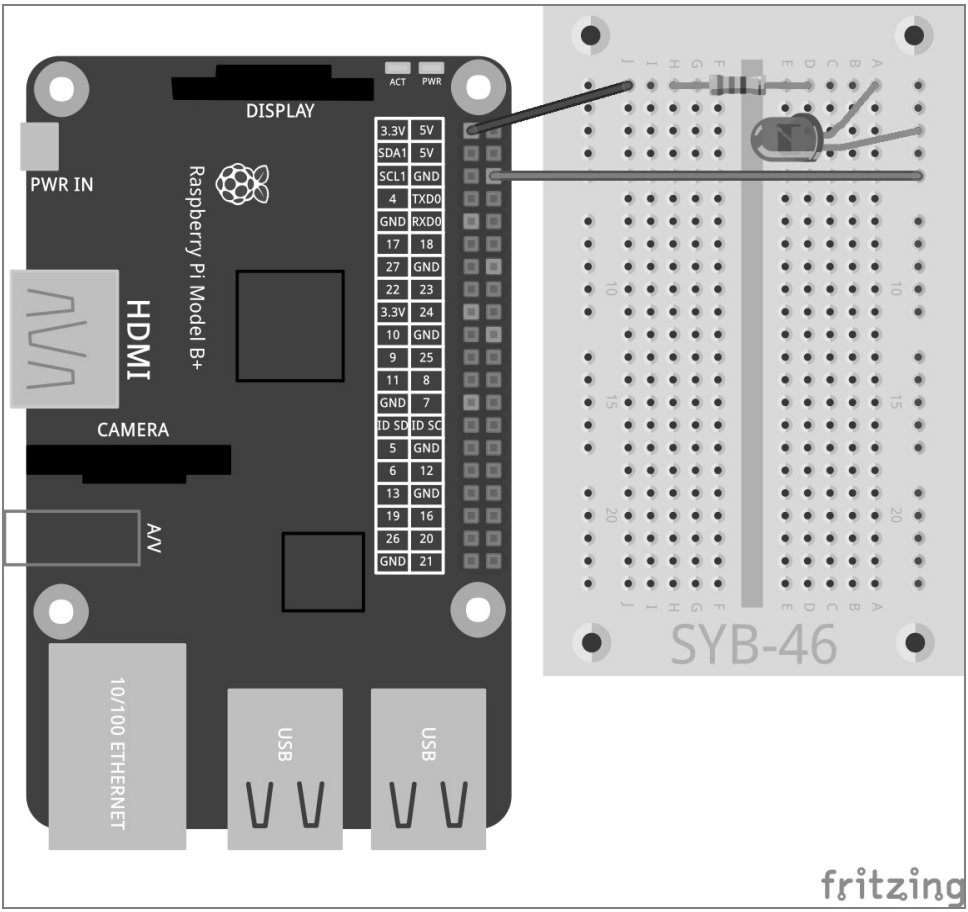

Abb. 2.4: Die erste LED am Raspberry Pi.

In diesem ersten Experiment wird der Raspberry Pi lediglich als Stromversorgung für die LED genutzt. Die LED leuchtet immer, man braucht keinerlei Software dazu.

Im nächsten Experiment setzen Sie einen Taster in die Zuleitung der LED. Die LED leuchtet jetzt nur, wenn dieser Taster gedrückt ist. Auch dafür braucht man keinerlei Software.

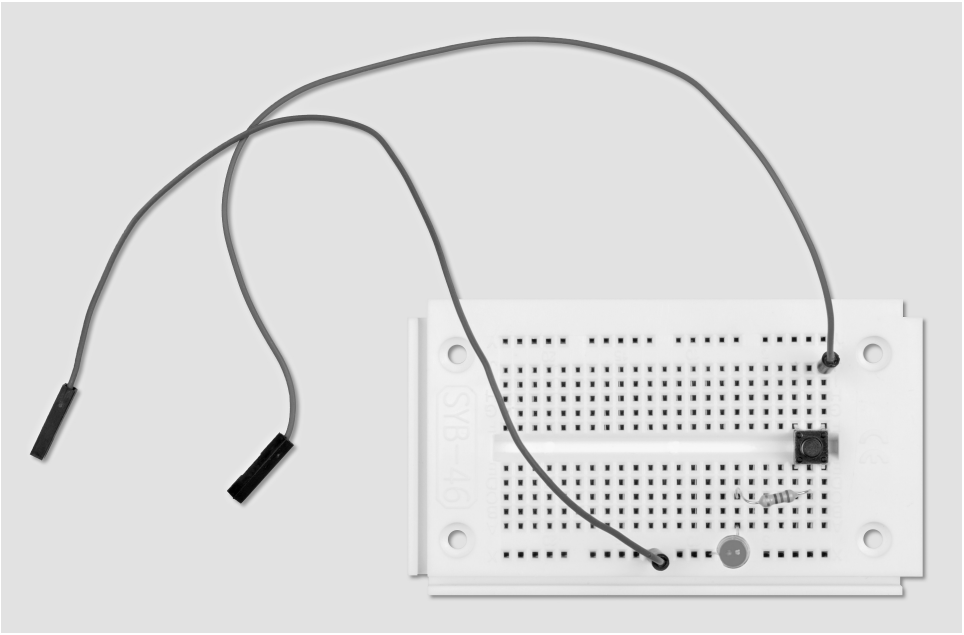

Abb. 2.5: Steckbrettaufbau für eine LED, die mit einem Taster geschaltet wird.

Benötigte Bauteile:
1x Steckplatine
1x LED rot
1x 220-Ohm-Widerstand
1x Taster
2x Verbindungskabel

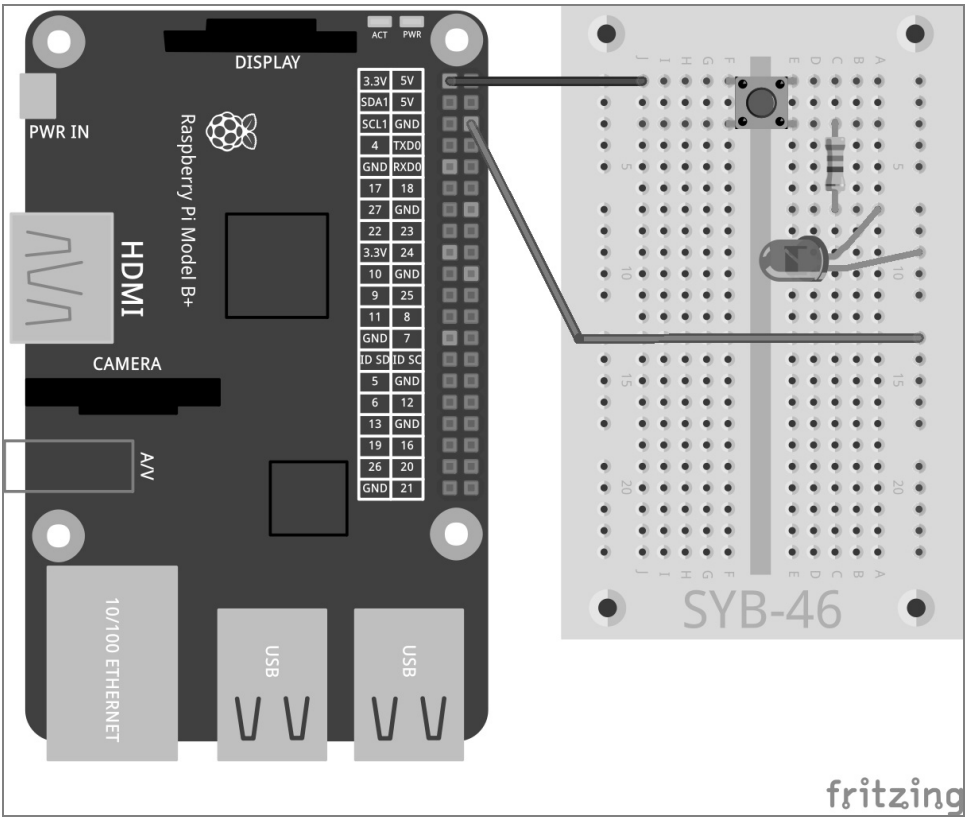

Abb. 2.6: LED mit Taster am Raspberry Pi.

2.3 GPIO mit Python

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

2.4 LED ein- und ausschalten

Schließen Sie wie auf dem nächsten Bild eine LED über einen 220-Ohm-Vorwiderstand (Rot-Rot-Braun) am GPIO-Port 25 (Pin 22) und nicht mehr direkt am +3,3-V-Anschluss an und verbinden Sie den Minuspol der LED über die Masseschiene der Steckplatine mit der Masseleitung des Raspberry Pi (Pin 6).

Benötigte Bauteile:
1x Steckplatine
1x LED rot
1x 220-Ohm-Widerstand
2x Verbindungskabel

Das nächste Programm `led.py` schaltet die LED für 2 Sekunden ein und danach wieder aus:

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
GPIO.setup(25, GPIO.OUT)
GPIO.output(25, 1)
time.sleep(2)
GPIO.output(25, 0)
GPIO.cleanup()
```

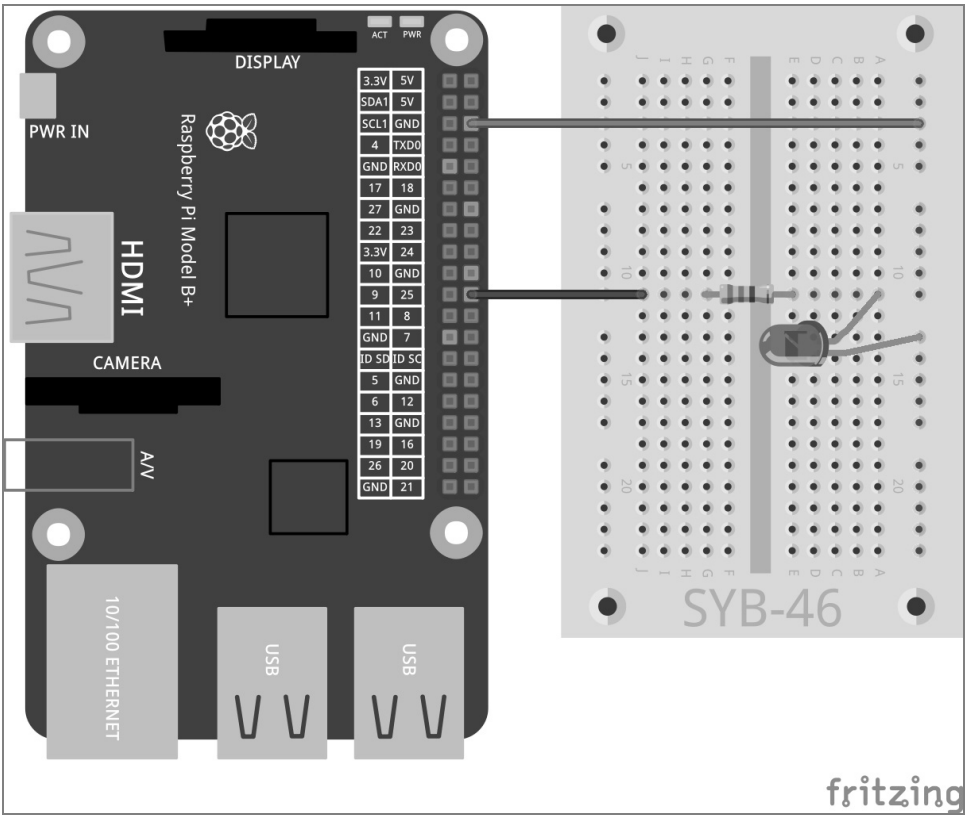

Abb. 2.7: Eine LED am GPIO-Port 25.

2.4.1 So funktioniert es

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

`import RPi.GPIO as GPIO` Die Bibliothek `RPi.GPIO` muss in jedem 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-Ports
Die Bibliothek `RPi.GPIO` unterstützt zwei verschiedene Methoden zur Bezeichnung der Ports. Im Modus `BCM` werden die bekannten GPIO-Portnummern genutzt, die auch auf Kommandozeilenebene oder in Shell-Skripten verwendet werden. Im alternativen Modus `BOARD` entsprechen die Bezeichnungen den Pin-Nummern von 1 bis 40 auf der Raspberry-Pi-Platine.

`GPIO.setup(25, GPIO.OUT)` Die Funktion `GPIO.setup` initialisiert einen GPIO-Port als Ausgang oder als Eingang. Der erste Parameter bezeichnet den Port je nach vorgegebenem Modus `BCM` oder `BOARD` mit seiner GPIO-Nummer oder Pin-Nummer. Der zweite Parameter kann entweder `GPIO.OUT` für einen Ausgang oder `GPIO.IN` für einen Eingang sein.

`GPIO.output(25, 1)` Auf dem soeben initialisierten Port wird eine `1` ausgegeben. Die dort angeschlossene LED leuchtet. Statt der `1` können auch die vordefinierten Werte `True` oder `GPIO.HIGH` ausgegeben werden.

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

`GPIO.output(25, 0)` Zum Ausschalten der LED gibt man den Wert `0` bzw. `False` oder `GPIO.LOW` auf dem GPIO-Port aus.

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

GPIO-Warnungen abfangen
Soll ein GPIO-Port konfiguriert werden, der nicht sauber zurückgesetzt wurde, sondern möglicherweise von einem anderen oder einem abgebrochenen Programm noch geöffnet ist, kommt es zu Warnmeldungen, die jedoch den Programmfluss nicht unterbrechen. Während der Programmentwicklung können diese Warnungen sehr nützlich sein, um Fehler zu entdecken. In einem fertigen Programm können sie einen unbedarften Anwender aber verwirren. Deshalb bietet die GPIO-Bibliothek mit `GPIO.setwarnings(False)` die Möglichkeit, diese Warnungen zu unterdrücken.

3 Verkehrsampel

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

Bauen Sie die abgebildete Schaltung auf der Steckplatine auf. Zur Ansteuerung der LEDs werden drei GPIO-Ports und eine gemeinsame Masseleitung verwendet. Die GPIO-Portnummern im BCM-Modus sind auf dem Raspberry Pi in der Zeichnung abgebildet.

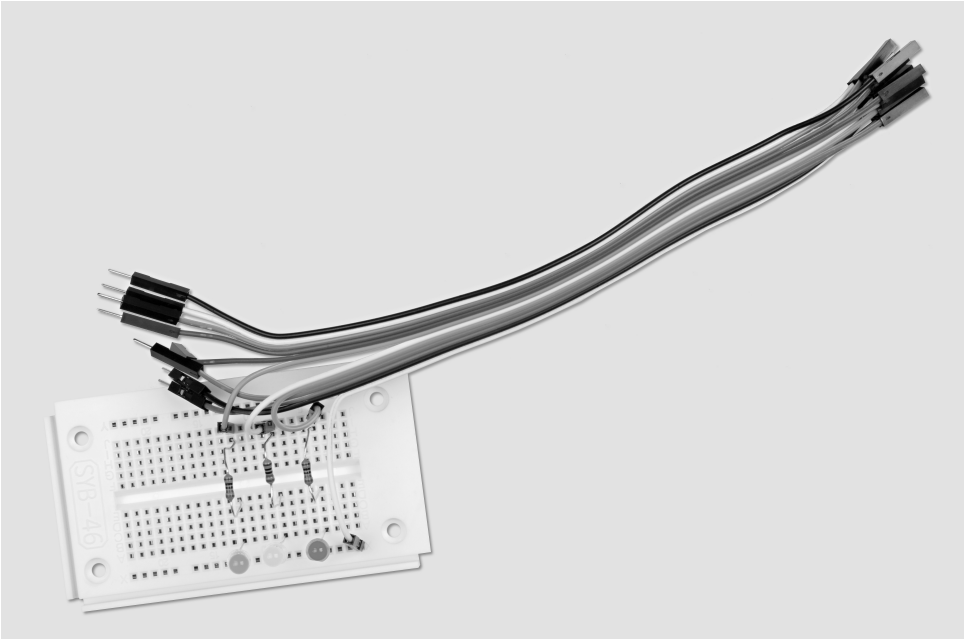

Abb. 3.1: Steckbrettaufbau für die Verkehrsampel.

Benötigte Bauteile:
1x Steckplatine
1x LED rot
1x LED gelb
1x LED grün
3x 220-Ohm-Widerstand
4x Verbindungskabel

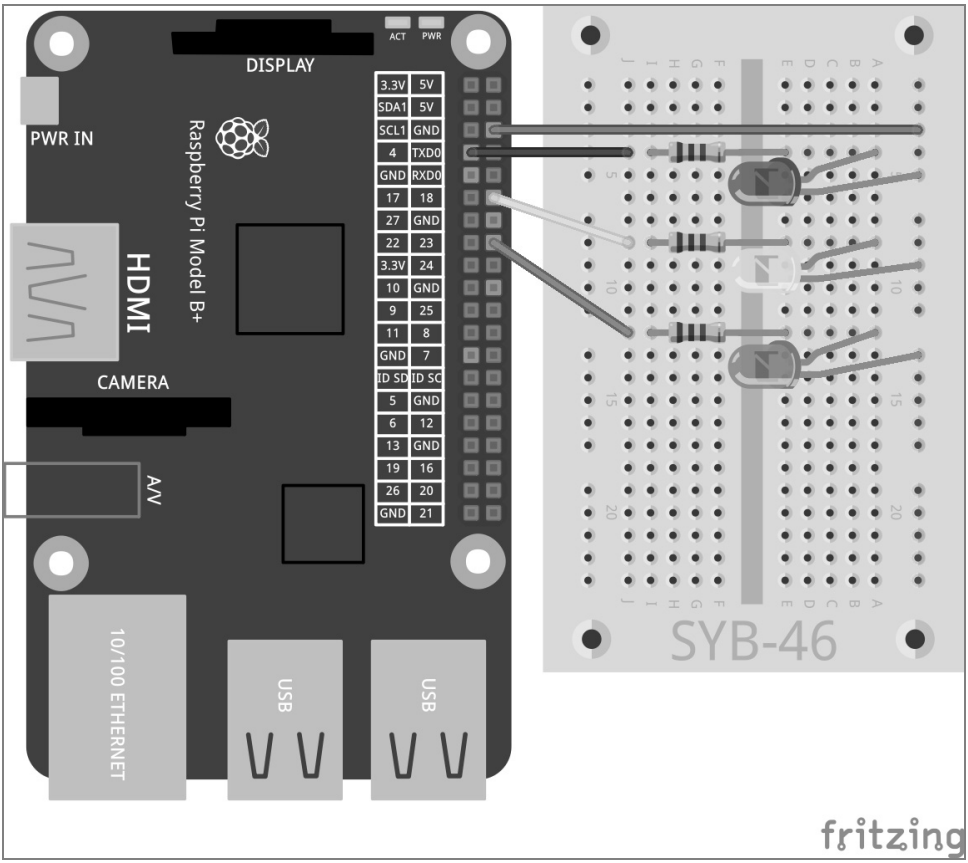

Abb. 3.2: Eine einfache Verkehrsampel.

Das Programm `ampel01.py` steuert die Ampel:

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
rot = 0; gelb = 1; gruen = 2
Ampel=[4,18,23]
GPIO.setup(Ampel[rot], GPIO.OUT, initial=False)
GPIO.setup(Ampel[gelb], GPIO.OUT, initial=False)
GPIO.setup(Ampel[gruen], GPIO.OUT, initial=True)
print ("Strg+C beendet das Programm")
try:
    while True:
        time.sleep(2)
        GPIO.output(Ampel[gruen],False); GPIO.output(Ampel[gelb],True)
```

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

3.1.1 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.

`rot = 0; gelb = 1; gruen = 2` Diese Zeilen definieren die drei Variablen `rot`, `gelb` und `gruen` für die drei LEDs. Damit braucht man sich im Programm keine Nummern oder GPIO-Ports zu merken, sondern kann die LEDs einfach über ihre Farben ansteuern.

`Ampel=[4,18,23]` Zur Ansteuerung der drei LEDs wird eine Liste eingerichtet, die die GPIO-Nummern in der Reihenfolge enthält, in der die LEDs auf der Steckplatine verbaut sind. Da die GPIO-Ports nur an dieser einen Stelle im Programm auftauchen, können Sie das Programm ganz einfach umbauen, wenn Sie andere GPIO-Ports nutzen möchten.

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

Nacheinander werden die drei verwendeten GPIO-Ports als Ausgänge initialisiert. 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. `Ampel[0]` ist also das erste Element, in diesem Fall `4`. Die Variablen `rot`, `gelb` und `gruen` enthalten die Zahlen `0`, `1` und `2`, die als Indizes für die Elemente der Liste benötigt werden. Auf diese Weise lassen sich die verwendeten GPIO-Ports über Farben adressieren:

- `Ampel[rot]` entspricht dem GPIO-Port `4` mit der roten LED.
- `Ampel[gelb]` entspricht dem GPIO-Port `18` mit der gelben LED.
- `Ampel[gruen]` entspricht dem GPIO-Port `23` mit der grünen LED.

Die `GPIO.setup`-Anweisung kann einen optionalen Parameter `initial` enthalten, der dem GPIO-Port bereits beim Initialisieren einen logischen Status zuweist. Damit schalten wir in diesem Programm die grüne LED 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 wäh-

renddessen 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 benötigen 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.

```
        GPIO.output(Ampel[gruen],False); GPIO.output(Ampel[gelb],True)
        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(Ampel[gelb],False); GPIO.output(Ampel[rot],True)
        time.sleep(2)
```

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

```
        GPIO.output(Ampel[gelb],True)
        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.

```
        GPIO.output(Ampel[rot],False)
        GPIO.output(Ampel[gelb],False)
        GPIO.output(Ampel[gruen],True)
```

Am Ende der Schleife springt die Ampel wieder auf Grün. Die rote und die gelbe LED werden ausgeschaltet, die grüne 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.

4 Fußgängerampel

Im nächsten Experiment erweitern wir die Ampelschaltung noch um eine zusätzliche Fußgängerampel, die während der Rotphase der Verkehrsampel ein Blinklicht für Fußgänger anzeigt, wie es in einigen Ländern verwendet wird. Natürlich könnte man auch die in Mitteleuropa übliche Fußgängerampel mit rotem und grünem Licht in das Programm einbauen, nur enthält dieses Lernpaket neben den von der Verkehrsampel verwendeten LEDs nur noch eine weitere.

Bauen Sie für das folgende Experiment eine zusätzliche LED mit Vorwiderstand wie abgebildet in die Schaltung ein. Diese wird am GPIO-Port 24 angeschlossen.

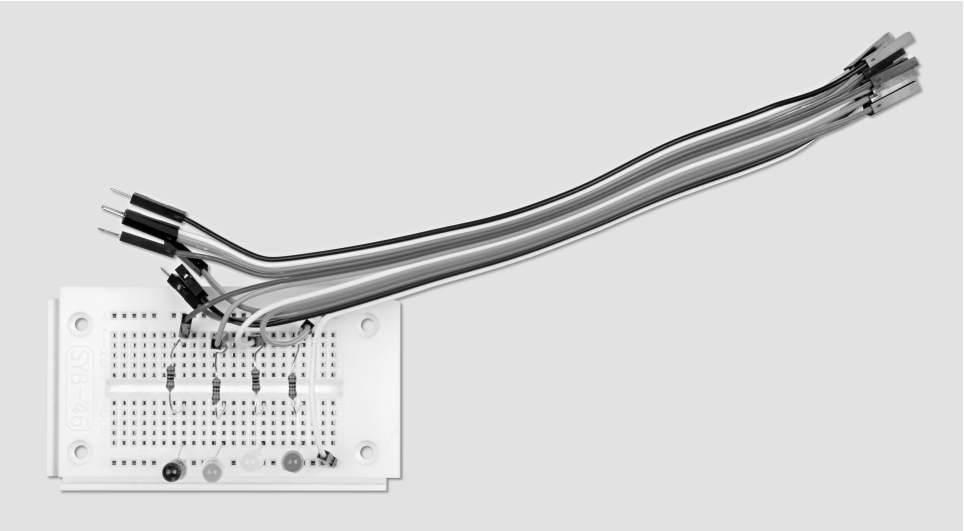

Abb. 4.1: Steckbrettaufbau für Verkehrsampel und Fußgängerblinklicht.

Benötigte Bauteile:
1x Steckplatine
1x LED rot
1x LED gelb
1x LED grün
1x LED blau
4x 220-Ohm-Widerstand
5x Verbindungskabel

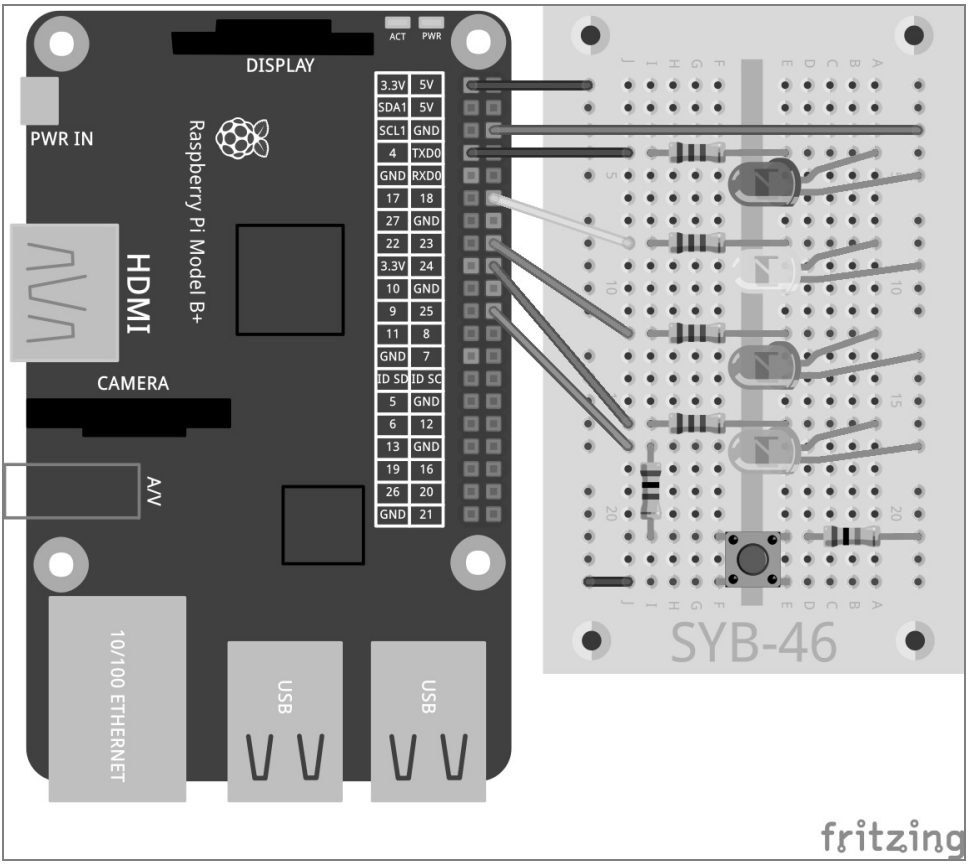

Abb. 4.2: Verkehrsampel mit Fußgängerblinklicht.

Das Programm `ampel02.py` steuert die neue Ampelanlage. Gegenüber der vorherigen Version wurde das Programm geringfügig erweitert.

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
rot    = 0; gelb   = 1; gruen  = 2; blau   = 3
Ampel=[4,18,23,24]
GPIO.setup(Ampel[rot], GPIO.OUT, initial=False)
GPIO.setup(Ampel[gelb], GPIO.OUT, initial=False)
GPIO.setup(Ampel[gruen], GPIO.OUT, initial=True)
GPIO.setup(Ampel[blau], GPIO.OUT, initial=False)
print ("Strg+C beendet das Programm")
try:
    while True:
```

```
        time.sleep(2)
        GPIO.output(Ampel[gruen],False); GPIO.output(Ampel[gelb],True)
        time.sleep(0.6)
        GPIO.output(Ampel[gelb],False); GPIO.output(Ampel[rot],True)
        time.sleep(0.6)
        for i in range(10):
            GPIO.output(Ampel[blau],True); time.sleep(0.05)
            GPIO.output(Ampel[blau],False); time.sleep(0.05)
        time.sleep(0.6)
        GPIO.output(Ampel[gelb],True); time.sleep(0.6)
        GPIO.output(Ampel[rot],False)
        GPIO.output(Ampel[gelb],False)
        GPIO.output(Ampel[gruen],True)
except KeyboardInterrupt:
    GPIO.cleanup()
```

4.1.1 So funktioniert es

Der Programmablauf ist weitgehend bekannt. Während der jetzt etwas längeren Rotphase soll die blaue Fußgängerampel schnell blinken.

`blau = 4` Eine neue Variable definiert die LED für die Fußgängerampel in der Liste.

`Ampel=[4,18,23,24]` Die Liste wird auf vier Elemente vergrößert, um die vier LEDs ansteuern zu können.

`GPIO.setup(Ampel[blau], GPIO.OUT, initial=False)` Die neue LED wird initialisiert und anfangs ausgeschaltet. Dies ist die Grundeinstellung während der Grünphase der Verkehrsampel.

```
        time.sleep(0.6)
        for i in range(10):
            GPIO.output(Ampel[blau],True); time.sleep(0.05)
            GPIO.output(Ampel[blau],False); time.sleep(0.05)
        time.sleep(0.6)
```

Im Ampelzyklus startet 0,6 Sekunden nach Beginn der Rotphase eine Schleife, die die blaue LED blinken lässt. Dazu verwenden wir hier eine `for`-Schleife, die im Gegensatz zu den in den früheren Experimenten verwendeten `while`-Schleifen immer eine bestimmte Anzahl an Schleifendurchläufen verwendet und nicht läuft, bis eine bestimmte Abbruchbedingung erfüllt ist.

`for i in range(10)`: Jede `for`-Schleife benötigt einen Schleifenzähler, eine Variable, die bei jedem Schleifendurchlauf einen neuen Wert annimmt. Für einfache Schleifenzähler hat sich in allen Programmiersprachen der Variablenname `i` eingebürgert. Jeder andere Name ist natürlich auch möglich. Dieser Wert kann wie jede andere Variable innerhalb der Schleife abgefragt werden, was hier aber nicht nötig ist. Der Parameter `range` in der Schleife gibt an, wie oft die Schleife durchläuft, genauer gesagt, welche Werte der Schleifenzähler annehmen kann. In unserem Beispiel läuft die Schleife zehnmal. Der Schleifenzähler `i` bekommt dabei Werte von `0` bis `9`. Innerhalb der Schleife wird die neue blaue LED eingeschaltet und nach 0,05 Sekunden wieder ausgeschaltet. Nach weiteren 0,05 Sekunden ist ein Schleifendurchlauf zu Ende, und der nächste startet wieder mit dem Einschalten der LED. Auf diese Weise blinkt sie zehnmal, was insgesamt 1 Sekunde dauert.

`time.sleep(0.6)` Mit einer Verzögerung von 0,6 Sekunden nach dem letzten Schleifendurchlauf wird der normale Schaltzyklus der Verkehrsampel fortgesetzt, indem die gelbe LED zusätzlich zur bereits leuchtenden roten eingeschaltet wird. So weit nicht viel Neues. Richtig interessant wird die Fußgängerampel, wenn sie nicht automatisch läuft, sondern erst durch einen Tastendruck gestartet wird, wie dies bei vielen Fußgängerampeln der Fall ist. Im nächsten Experiment wird ein an einem GPIO-Port angeschlossener Taster den Druckknopf an einer echten Fußgängerampel simulieren.

4.2 Taster am GPIO-Anschluss

GPIO-Ports 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. Liegt auf einem als Eingang definierten GPIO-Port ein +3,3-V-Signal an, wird dieses als logisch `True` bzw. `1` ausgewertet. Theoretisch könnten Sie also über einen Taster den jeweiligen GPIO-Port mit dem +3,3-V-Anschluss des Raspberry Pi verbinden, was Sie aber auf keinen Fall tun dürfen! Der GPIO-Port wird dadurch überlastet. Schließen Sie immer einen 1-kOhm-Schutzwiderstand zwischen einem GPIO-Eingang und dem +3,3-V-Anschluss an, um zu verhindern, dass zu viel Strom auf den GPIO-Port und damit auf den Prozessor fließt.

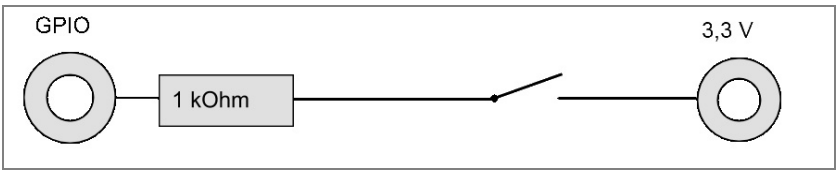

Abb. 4.3: Taster mit Schutzwiderstand an einem GPIO-Eingang.

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

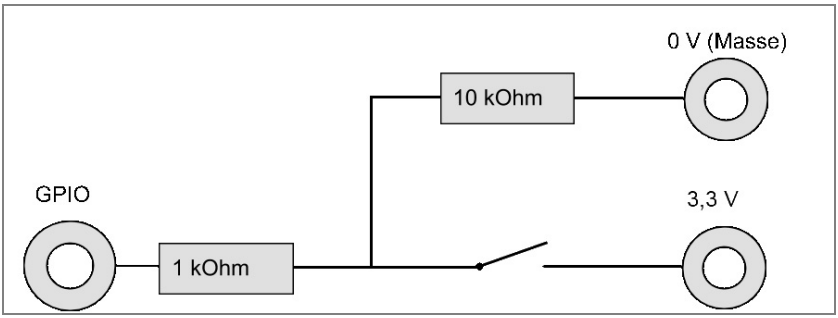

Abb. 4.4: Taster mit Schutzwiderstand und Pull-down-Widerstand an einem GPIO-Eingang.

Bauen Sie gemäß folgender Abbildung einen Taster mit den beiden Widerständen in die Schaltung ein.

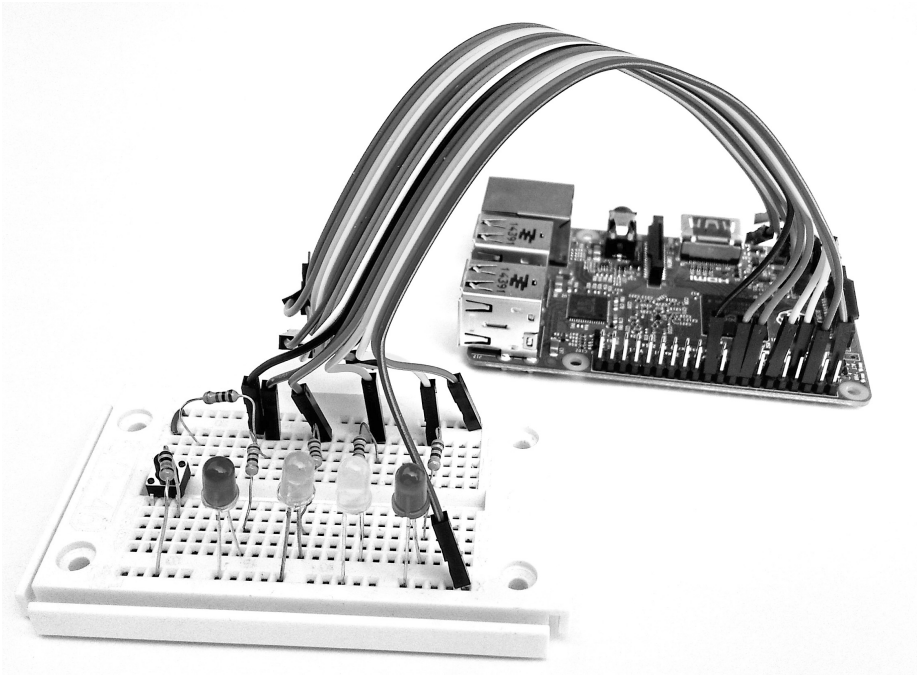

Abb. 4.5: Steckbrettaufbau für die Fußgänger-»Bedarfsampel«.

Benötigte Bauteile:
1x Steckplatine
1x LED rot
1x LED gelb
1x LED grün
1x LED blau
4x 220-Ohm-Widerstand
1x Taster
1x 1-kOhm-Widerstand
1x 10-kOhm-Widerstand
7x Verbindungskabel
1x kurze Drahtbrücke

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

Die in der Abbildung obere Kontaktleiste des Tasters ist über einen 1-kOhm-Schutzwiderstand (Braun-Schwarz-Rot) mit dem GPIO-Port 25 verbunden und über einen 10-kOhm-Pull-down-Widerstand (Braun-Schwarz-Orange) mit der Masseleitung.

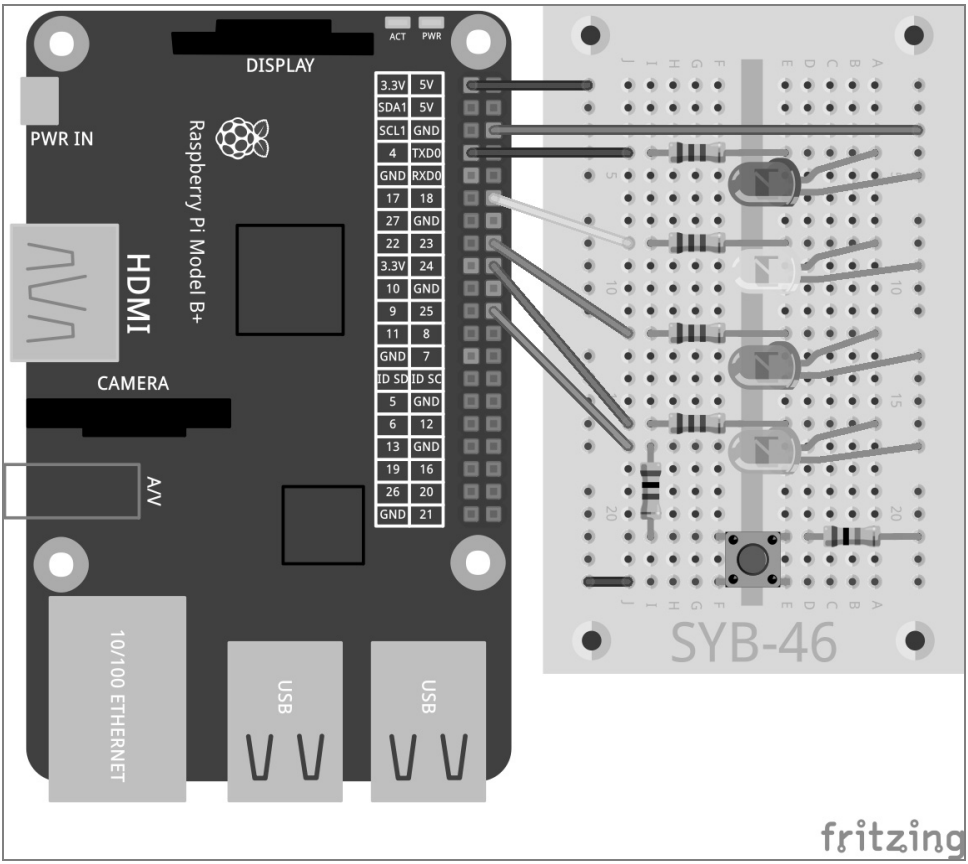

Abb. 4.6: Fußgängerblinklicht mit Taster.

Das Programm `ampel03.py` steuert die neue Ampelanlage mit dem Taster für das Fußgängerblinklicht.

```
# -*- coding: utf-8 -*-
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

rot = 0; gelb = 1; gruen = 2; blau = 3; taster = 4

Ampel=[4,18,23,24,25]
GPIO.setup(Ampel[rot], GPIO.OUT, initial=False)
GPIO.setup(Ampel[gelb], GPIO.OUT, initial=False)
GPIO.setup(Ampel[gruen], GPIO.OUT, initial=True)
GPIO.setup(Ampel[blau], GPIO.OUT, initial=False)
```

```
GPIO.setup(Ampel[taster], GPIO.IN)

print ("Taster drücken, um Fußgängerblinklicht einzuschalten, Strg+C beendet das Programm")

try:
    while True:
        if GPIO.input(Ampel[taster])==True:
            GPIO.output(Ampel[gruen],False)
            GPIO.output(Ampel[gelb],True)
            time.sleep(0.6)
            GPIO.output(Ampel[gelb],False)
            GPIO.output(Ampel[rot],True)
            time.sleep(0.6)
            for i in range(10):
                GPIO.output(Ampel[blau],True); time.sleep(0.05)
                GPIO.output(Ampel[blau],False); time.sleep(0.05)
            time.sleep(0.6)
            GPIO.output(Ampel[gelb],True)
            time.sleep(0.6)
            GPIO.output(Ampel[rot],False); GPIO.output(Ampel[gelb],False)
            GPIO.output(Ampel[gruen],True); time.sleep(2)
except KeyboardInterrupt:
    GPIO.cleanup()
```

4.2.1 So funktioniert es

Das Programm wurde gegenüber der letzten Version noch etwas ergänzt.

`# -*- coding: utf-8 -*-` Damit die deutschen Umlaute von `Fußgängerblinklicht` in der Programmausgabe korrekt angezeigt werden – unabhängig davon, wie die IDLE-Oberfläche beim Benutzer eingestellt ist –, wird am Anfang eine Codierung zur Darstellung der Sonderzeichen definiert. Diese Zeile sollte in allen Programmen enthalten sein, die Texte ausgeben, in denen sich Umlaute oder andere landestypische Sonderzeichen befinden.

ASCII, ANSI und Unicode

Ein normales Alphabet hat 26 Buchstaben plus ein paar Umlaute, alles in Groß- und Kleinschreibung, dazu zehn Ziffern und einige Satzzeichen; das macht zusammen etwa 100 verschiedene Zeichen. Mit einem Byte lassen sich 256 verschiedene Zeichen darstellen. Das sollte also ausreichen – so dachte man am Anfang der Computergeschichte, als die wichtigsten Grundlagen der heutigen Technik definiert wurden.

Ziemlich bald stellte sich heraus, dass die Erfinder des auf 256 Zeichen basierenden ASCII-Zeichensatzes (American Standard Code for Information Interchange) falsch lagen. Es waren Amerikaner, die nicht über den englischen Sprachraum hinaus gedacht hatten. In allen wichtigen Weltsprachen, ohne die ostasiatischen und arabischen Sprachen mit ihren ganz eigenen Schriften, gibt es mehrere Hundert Buchstaben, die dargestellt werden müssen. Von denen passten nur wenige auf die freien Plätze in der 256 Zeichen umfassenden Liste.

Als später parallel zum ASCII-Zeichensatz der ANSI-Zeichensatz eingeführt wurde, der von älteren Windows-Versionen verwendet wird, machte man den gleichen Fehler noch einmal. Um das Sprachengewirr perfekt zu machen, wurden die deutschen Umlaute und andere Buchstaben mit Akzenten an anderen Stellen im Zeichensatz eingeordnet als im ASCII-Standard.

Zur Lösung dieses Problems führte man in den 90er-Jahren den Unicode ein, der alle Sprachen, auch ägyptische Hieroglyphen, Keilschrift und das vedische Sanskrit, die älteste überlieferte Schriftsprache der Welt, darstellen kann. Die am weitesten verbreitete Form, Unicode-Zeichen in reinen Textdateien zu codieren, ist UTF-8, eine Codierung, die plattformübergreifend funktioniert und in den ersten 128 Zeichen mit ASCII deckungsgleich und damit abwärtskompatibel zu fast allen Text darstellenden Systemen ist. Die Codierung wird in einer Kommentarzeile angegeben. Alle Zeilen, die mit einem #-Zeichen beginnen, werden vom Python-Interpreter nicht ausgewertet. Die Codierung, die immer ganz am Anfang eines Programms stehen muss, weist die Python-Shell an, wie Zeichen darzustellen sind, ist aber keine wirkliche Programmanweisung. Auf diese Weise können Sie auch beliebige eigene Kommentare in den Programmcode eintragen.

Kommentare in Programmen
Wenn man ein Programm schreibt, weiß man später oft nicht mehr, was man sich bei bestimmten Programmanweisungen gedacht hat. Programmieren ist eine der kreativsten Tätigkeiten überhaupt, da man einzig und allein, ohne Beschränkungen durch Material und Werkzeug, aus seinen Ideen etwas erschafft. Gerade bei Programmen, die auch eine andere Person verstehen oder sogar weiter bearbeiten soll, sind Kommentare wichtig. In den Beispielprogrammen sind keine Kommentare enthalten, um den Programmcode übersichtlich zu halten. Alle Programmanweisungen sind ausführlich beschrieben.

Bei Programmen, die man selbst veröffentlicht, stellt sich immer die Frage: Kommentare in Deutsch oder Englisch? Bei deutschen Kommentaren beschweren sich die Franzosen über die unverständliche Sprache, englische Kommentare versteht man selbst irgendwann nicht mehr, und die Briten lachen über das schlechte Englisch.

```
taster = 4
Ampel=[4,18,23,24,25]
```

Der Einfachheit halber wird für den Taster ein zusätzliches Element mit der Nummer `4` und dem GPIO-Port 25 in die Liste eingefügt. Auf diese Weise kann man für den Taster auch leicht einen anderen GPIO-Port wählen, da dessen Nummer wie die GPIO-Ports der LEDs nur an dieser einen Stelle im Programm eingetragen ist.

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

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

Beim Start zeigt das Programm eine erweiterte Meldung an, die mitteilt, dass der Benutzer den Taster drücken soll.

```
    while True:
        if GPIO.input(Ampel[taster])==True:
```

Innerhalb der Endlosschleife ist jetzt eine Abfrage eingebaut. Die folgenden Anweisungen werden erst ausgeführt, wenn der GPIO-Port 25 den Wert `True` annimmt, der Benutzer also den Taster drückt. Bis zu diesem Zeitpunkt bleibt die Verkehrsampel in ihrer Grünphase stehen. Der weitere Ablauf der Schleife entspricht im Wesentlichen dem des letzten Programms. Die Verkehrsampel schaltet über Gelb auf Rot, das Blinklicht blinkt zehnmal. Danach schaltet die Ampel wieder von Rot über Gelb auf Grün.

`time.sleep(2)` Einen kleinen Unterschied gibt es in diesem Programm. Die 2 Sekunden dauernde Grünphase ist jetzt am Ende der Schleife und nicht mehr am Anfang eingebaut. Sie wird trotzdem einmal je Schleifendurchlauf angewandt, mit dem Unterschied, dass der Ampelzyklus sofort und ohne Verzögerung beginnt, wenn der Taster gedrückt wird. Um zu verhindern, dass die Grünphase fast ausfällt, wenn der Taster unmittelbar nach der Gelbphase wieder gedrückt wird, ist diese Verzögerung jetzt am Ende der Schleife eingebaut.

5 Bunte LED-Muster und Lauflichter

Lauflichter sind immer wieder beliebte Effekte, um Aufmerksamkeit zu erregen, sei es im Partykeller oder in professioneller Leuchtwerbung. Mit dem Raspberry Pi und ein paar LEDs lässt sich so etwas leicht realisieren.

Bauen Sie für das folgende Experiment vier LEDs mit Vorwiderständen wie abgebildet auf. Diese Schaltung entspricht der Fußgängerampel ohne den Taster aus dem vorherigen Experiment.

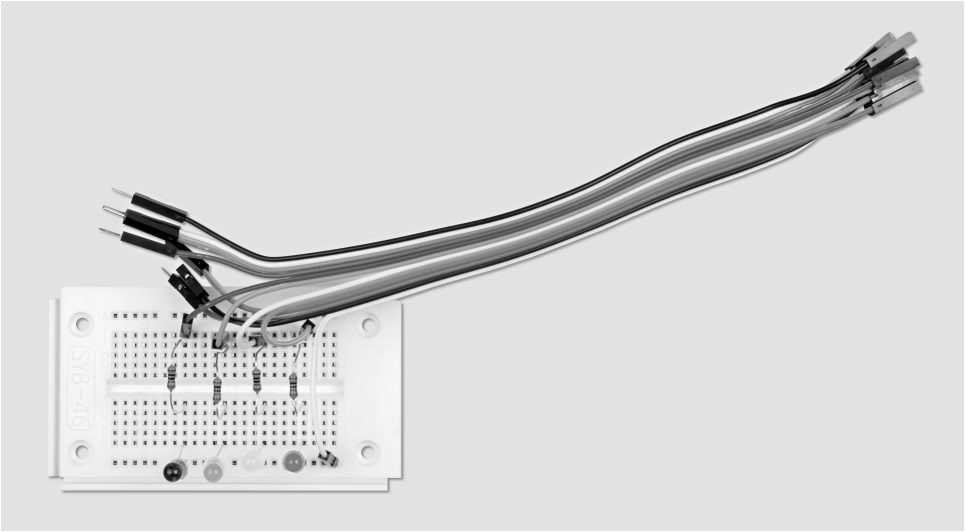

Abb. 5.1: Steckbrettaufbau für die Muster und Lauflichter.

Benötigte Bauteile:
1x Steckplatine
1x LED rot
1x LED gelb
1x LED grün
1x LED blau
4x 220-Ohm-Widerstand
5x Verbindungskabel

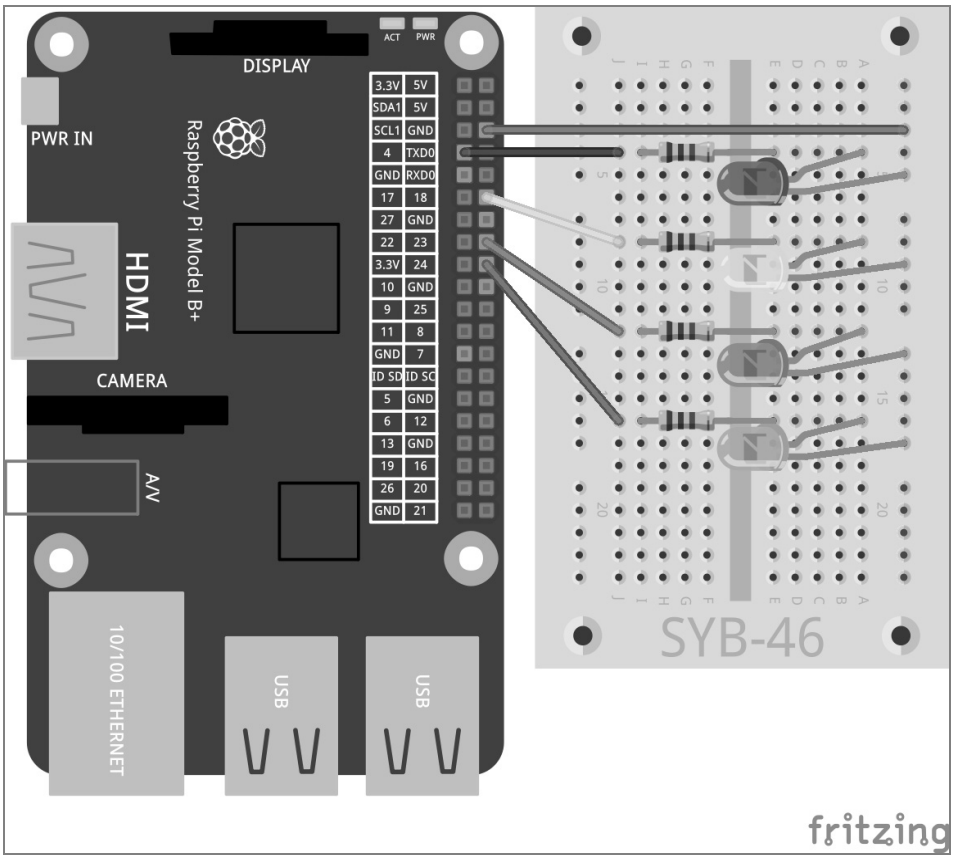

Abb. 5.2: Vier LEDs mit Vorwiderständen.

Anhand verschiedener LED-Blinkmuster erklären wir weitere Schleifen und Programmiermethoden in Python. Das nächste Programm bietet verschiedene LED-Muster an, die vom Benutzer per Tastatureingabe ausgewählt werden können.

Das Programm `ledmuster.py` lässt die LEDs in unterschiedlichen Mustern blinken.

```
# -*- coding: utf-8 -*-
import RPi.GPIO as GPIO
import time
import random

GPIO.setmode(GPIO.BCM)

LED = [4,18,23,24]
for i in LED:
    GPIO.setup(i, GPIO.OUT, initial=0)

z = len(LED); w = 5; t = 0.2

print ("Lichteffekte zur Auswahl"); print ("1 - Lauflicht zyklisch")
print ("2 - Lauflicht hin und zurück"); print ("3 - auf- und absteigend")
print ("4 - alle blinken gleichzeitig"); print ("5 - alle blinken zufällig")
print ("Strg+C beendet das Programm")

try:
    while True:
        e = raw_input ("Bitte Muster auswählen: ")
        if e == "1":
            for i in range(w):
                for j in range(z):
                    GPIO.output(LED[j], True); time.sleep(t)
                    GPIO.output(LED[j], False)
        elif e == "2":
            for i in range(w):
                for j in range(z):
                    GPIO.output(LED[j], True); time.sleep(t)
                    GPIO.output(LED[j], False)
                for j in range(z-1, -1, -1):
                    GPIO.output(LED[j], True); time.sleep(t)
                    GPIO.output(LED[j], False)
        elif e == "3":
            for i in range(w):
                for j in range(z):
                    GPIO.output(LED[j], True); time.sleep(t)
                time.sleep(2*t)
                for j in range(z-1, -1, -1):
                    GPIO.output(LED[j], False)
                    time.sleep(t)
                time.sleep(2*t)
        elif e == "4":
            for i in range(w):
                for j in range(z):
                    GPIO.output(LED[j], True)
                time.sleep(2*t)
```

```
                for j in range(z):
                    GPIO.output(LED[j], False)
                time.sleep(t)
        elif e == "5":
            for i in range(w*z):
                j = random.randint(0,z-1)
                GPIO.output(LED[j], True); time.sleep(t)
                GPIO.output(LED[j], False)
        else:
             print ("Ungültige Eingabe")

except KeyboardInterrupt:
    GPIO.cleanup()
```

5.1.1 So funktioniert es

Die ersten Zeilen des Programms mit der Definition der UTF-8-Codierung und dem Import der notwendigen Bibliotheken sind bereits aus früheren Experimenten bekannt. Hier wird zusätzlich die Bibliothek `random` importiert, um ein zufälliges Blinkmuster zu erzeugen.

Bei der Entwicklung dieses Programms wurde darauf geachtet, dass es möglichst vielseitig nutzbar ist, sich also problemlos auf mehr als vier LEDs erweitern lässt. Zu einem guten Programmierstil gehört heute eine solche Flexibilität. Am Beispiel des Raspberry Pi lassen sich derart programmierte Programme nicht nur um neue GPIO-Ports erweitern, sondern auch leicht auf andere GPIO-Ports umschreiben, wenn dies hardwaretechnisch notwendig ist.

`LED=[4,18,23,24]` Für die LEDs wird wieder eine Liste mit GPIO-Portnummern definiert, damit man diese Ports nur an einer Stelle im Programm fest eintragen muss.

```
for i in LED:
    GPIO.setup(i, GPIO.OUT, initial=0)
```

Anstatt wie in früheren Programmen die GPIO-Ports der LEDs einzeln zu initialisieren, läuft diesmal eine `for`-Schleife über die Liste `LED`. Der Schleifenzähler `i` nimmt nacheinander die einzelnen Werte aus der Liste an, im Beispiel die GPIO-Portnummern der LEDs, und wird nicht einfach hochgezählt, wie in den bisher verwendeten `for`-Schleifen. Auf diese Weise können beliebig lange Listen abgearbeitet werden. Die Länge der Liste muss bei der Entwicklung des Programms nicht einmal bekannt sein.

Die vier GPIO-Ports für die LEDs werden als Ausgänge definiert und auf `0` gesetzt, um eventuell aus früheren Experimenten leuchtende LEDs abzuschalten.

```
z = len(LED); w = 5; t = 0.2
```

Um das Programm allgemeingültig und leicht veränderbar zu halten, werden jetzt noch drei Variablen definiert:

`z`	Zahl der LEDs	Die Anzahl der LEDs wird mithilfe der Funktion `len()` automatisch aus der Liste `LED` übernommen.
`w`	Wiederholungen	Jedes Muster wird, damit man es besser erkennt, standardmäßig fünfmal wiederholt. Diese Anzahl kann beliebig geändert werden und gilt dann für alle Muster.
`t`	Zeit	Diese Variable gibt an, wie lange eine LED beim Blinken eingeschaltet ist. Die Pause danach dauert genauso lange. Der Name `t` hat sich für Variablen, die Zeiträume in Programmen speichern, in fast allen Programmiersprachen eingebürgert.

Die als Variablen definierten Werte sind nur an dieser Stelle fest im Programm eingetragen und können so leicht verändert werden. Nach diesen Definitionen startet das eigentliche Programm.

```
print ("Lichteffekte zur Auswahl"); print ("1 - Lauflicht zyklisch")
print ("2 - Lauflicht hin und zurück"); print ("3 - auf- und absteigend")
print ("4 - alle blinken gleichzeitig"); print ("5 - alle blinken zufällig")
print ("Strg+C beendet das Programm")
```

Diese Zeilen geben für den Benutzer eine Anleitung auf dem Bildschirm dazu aus, mit welcher Zifferntaste welches Muster dargestellt wird.

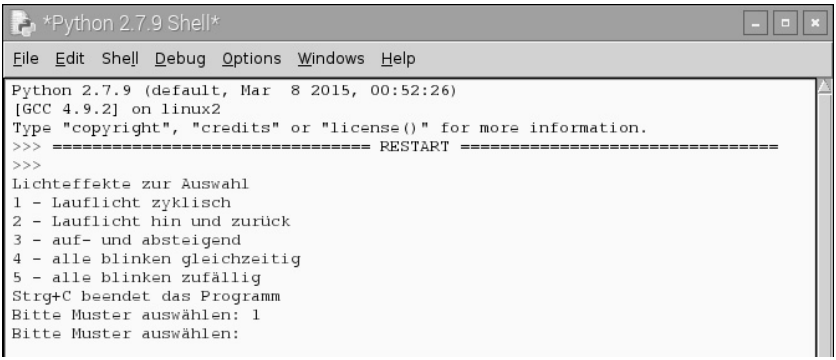

Abb. 5.3: Das Programm auf dem Bildschirm.

Nachdem die Auswahl angezeigt wurde, startet die Hauptschleife des Programms. Dazu verwenden wir auch hier eine `while True:`-Endlosschleife, die in einer `try...except`-Anweisung eingebettet ist.

`e = raw_input ("Bitte Muster auswählen: ")` Gleich am Beginn der Schleife wartet das Programm auf eine Benutzereingabe, die in der Variablen `e` gespeichert wird. Die Funktion `raw_input()` übernimmt die Eingabe im Klartext, ohne sie auszuwerten. Im Gegensatz dazu werden mit `input()` eingegebene mathe-

matische Operationen oder Variablennamen direkt ausgewertet. In den meisten Fällen ist also `raw_input()` die bessere Wahl, da man sich um viele Eventualitäten möglicher Eingaben nicht zu kümmern braucht.

Das Programm wartet, bis der Benutzer einen Buchstaben eingibt und die Enter-Taste drückt. Abhängig davon, welche Zahl der Benutzer eingegeben hat, soll jetzt ein bestimmtes Muster mit den LEDs angezeigt werden. Um dies abzufragen, verwenden wir ein `if...elif...else`-Konstrukt.

Muster 1
War die Eingabe eine `1`, wird der hinter dieser Zeile eingerückte Programmteil ausgeführt.

`if e == "1"`: Beachten Sie, dass Einrückungen in Python nicht nur der Optik dienen. Wie bei Schleifen werden auch solche Abfragen mit einer Einrückung eingeleitet.

Gleich ist nicht gleich Gleich
Python verwendet zwei Arten von Gleichheitszeichen. Das einfache = dient dazu, einer Variablen einen bestimmten Wert zuzuweisen. Das doppelte Gleichheitszeichen == wird in Abfragen verwendet und prüft, ob zwei Werte wirklich gleich sind.

Falls der Benutzer also eine `1` über die Tastatur eingegeben hat, startet eine Schleife, die ein zyklisches Lauflicht erzeugt. Diese Schleifen sind für alle verwendeten LED-Muster prinzipiell gleich aufgebaut.

`for i in range(w)`: Die äußere Schleife wiederholt das Muster so oft, wie in der eingangs definierten Variablen `w` angegeben. In dieser Schleife liegt eine weitere, die das jeweilige Muster erzeugt. Diese unterscheidet sich bei jedem Muster.

```
                for j in range(z):
                    GPIO.output(LED[j], True); time.sleep(t)
                    GPIO.output(LED[j], False)
```

Im Fall des einfachen zyklischen Lauflichts läuft diese Schleife nacheinander für jede LED der Liste einmal durch. Wie viele LEDs das sind, wurde am Anfang des Programms in der Variablen `z` gespeichert. Die LED mit der Nummer des aktuellen Stands des Schleifenzählers wird eingeschaltet. Danach wartet das Programm die eingangs in der Variablen `t` gespeicherte Zeit und schaltet die LED danach wieder aus. Anschließend beginnt der nächste Schleifendurchlauf mit der nächsten LED. Die äußere Schleife wiederholt die gesamte innere Schleife fünfmal.

Muster 2
Hat der Benutzer eine `2` eingegeben, startet eine ähnliche Schleife. Hier werden die LEDs aber nicht nur in einer Richtung durchgezählt, sondern am Ende des Lauflichts wieder in umgekehrter Reihenfolge. Das Licht läuft abwechselnd vor und zurück.

`elif e == "2"`: Die weiteren Abfragen nach der ersten verwenden die Abfrage `elif`, was bedeutet, dass sie nur dann ausgeführt werden, wenn die vorhergehende Abfrage als Ergebnis `False` zurückgab.

```
            for i in range(w):
                for j in range(z):
                    GPIO.output(LED[j], True); time.sleep(t)
                    GPIO.output(LED[j], False)
                for j in range(z-1, -1, -1):
                    GPIO.output(LED[j], True); time.sleep(t)
                    GPIO.output(LED[j], False)
```

Auch hier werden ineinandergeschachtelte Schleifen verwendet. Nach der ersten inneren Schleife, die dem zuvor beschriebenen Programmteil entspricht, also nachdem die LED mit der Nummer 3 leuchtet, beginnt eine weitere Schleife für das Lauflicht in entgegengesetzter Richtung. Listenelemente sind immer mit 0 beginnend nummeriert. Die vierte LED hat demnach die Nummer 3.

Um eine Schleife rückwärts laufen zu lassen, verwenden wir die erweiterte Syntax von `for...range()`. Anstatt nur einen Endwert anzugeben, können auch drei Parameter angegeben werden: Startwert, Schrittweite und Endwert. In unserem Beispiel sind das:

Startwert	z-1	Die Variable `z` enthält die Anzahl der LEDs. Da die Nummerierung der Listenelemente mit `0` beginnt, hat die letzte LED die Nummer `z-1`.
Schrittweite	-1	Bei einer Schrittweite von `-1` zählt jeder Schleifendurchlauf eine Zahl rückwärts.
Endwert	-1	Der Endwert in einer Schleife ist immer der erste Wert, der nicht erreicht wird. In der ersten vorwärts zählenden Schleife beginnt der Schleifenzähler bei `0` und erreicht in unserem Beispiel die Werte `0`, `1`, `2`, `3`, um die LEDs zu adressieren. Die `4` wird bei viermaligem Schleifendurchlauf nicht erreicht. Die rückwärts zählende Schleife soll mit der `0` enden und so den Wert `-1` als ersten nicht erreichen.

Die beschriebene zweite Schleife lässt nacheinander die vier LEDs in umgekehrter Richtung blinken. Danach startet die äußere Schleife den gesamten Zyklus neu, der hier, da jede LED zweimal blinkt, doppelt so lange dauert wie im ersten Programmteil.

Muster 3

Hat der Benutzer eine `3` eingegeben, startet eine ähnliche Schleife. Hier werden die LEDs auch in beiden Richtungen durchgezählt, aber nicht gleich nach dem Einschalten wieder ausgeschaltet.

```
        elif e == "3":
            for i in range(w):
                for j in range(z):
                    GPIO.output(LED[j], True); time.sleep(t)
                time.sleep(2*t)
                for j in range(z-1, -1, -1):
                    GPIO.output(LED[j], False); time.sleep(t)
                time.sleep(2*t)
```

Die erste innere Schleife schaltet die LEDs nacheinander mit einer Zeitverzögerung ein. Am Ende der Schleife, das an der Ausrückung der Zeile `time.sleep(2*t)` zu erkennen ist, wird die doppelte Verzögerungszeit gewartet. So lange leuchten alle LEDs. Danach beginnt eine weitere Schleife, die rückwärts zählt und eine LED nach der anderen wieder ausschaltet. Auch hier wird am Ende, wenn alle LEDs aus sind, die doppelte Verzögerungszeit gewartet, bevor die äußere Schleife den gesamten Zyklus noch einmal startet.

Muster 4
Hat der Benutzer eine `4` eingegeben, startet ein anderes Blinkmuster, bei dem alle LEDs gleichzeitig blinken und nicht nacheinander aufgerufen werden.

```
        elif e == "4":
            for i in range(w):
                for j in range(z):
                    GPIO.output(LED[j], True)
                time.sleep(2*t)
                for j in range(z):
                    GPIO.output(LED[j], False)
                time.sleep(t)
```

Da nicht mehrere GPIO-Ports mit einer einzigen Anweisung auf einmal ein- oder ausgeschaltet werden können, werden auch hier Schleifen verwendet, allerdings ohne Zeitverzögerung innerhalb der Schleife. Die vier LEDs werden unmittelbar nacheinander eingeschaltet. Für das menschliche Auge erscheint es gleichzeitig. Am Ende der ersten inneren Schleife wartet das Programm die doppelte Verzögerungszeit, bevor alle LEDs wieder ausgeschaltet werden.

Durch unterschiedliche Leucht- und Dunkelzeiten lassen sich verschiedene Effekte bei Blinklichtern erzeugen. Blinken wird eher wahrgenommen, wenn die Leuchtzeit länger als die Dunkelzeit ist. Sehr kurze Leuchtzeiten bei vergleichsweise langen Dunkelzeiten erzeugen einen Blitzlichteffekt.

Muster 5
Hat der Benutzer eine `5` eingegeben, blinken die LEDs völlig zufällig.

```
        elif e == "5":
            for i in range(w*z):
                j = random.randint(0,z-1)
                GPIO.output(LED[j], True); time.sleep(t)
                GPIO.output(LED[j], False)
```

Da hier keine verschachtelten Schleifen verwendet werden, lassen wir die Schleife öfter durchlaufen. Durch Multiplikation der Variablen `w` und `z` blinkt jede LED durchschnittlich so oft wie im ersten Muster.

Die Funktion `random.randint()` schreibt eine Zufallszahl in die Variable `j`. Diese Zufallszahl ist größer oder gleich dem ersten Parameter und kleiner oder gleich dem zweiten Parameter, kann also in unserem Fall die Werte `0`, `1`, `2`, `3` annehmen.

Die per Zufall ausgewählte LED wird ein- und nach der Verzögerungszeit wieder ausgeschaltet. Danach startet die Schleife neu, und eine neue LED wird per Zufall ausgewählt.

Ungültige Eingabe

Bei allen Programmen, die Benutzereingaben erfordern, muss man Fehleingaben abfangen. Gibt der Benutzer etwas nicht Vorgesehenes ein, muss das Programm darauf reagieren.

```
        else:
            print ("Ungültige Eingabe")
```

Hat der Benutzer irgendetwas anderes eingegeben, wird die unter `else` angegebene Anweisung ausgeführt. Dieser Abschnitt einer Abfrage trifft immer dann zu, wenn keine der anderen Abfragen ein wahres Ergebnis geliefert hat. In unserem Fall gibt das Programm eine Meldung auf dem Bildschirm aus.

Wie in den vorangegangenen Experimenten wird das Programm über einen `KeyboardInterrupt` beendet, indem der Benutzer die Tastenkombination Strg + C drückt. Die letzte Zeile schließt die verwendeten GPIO-Ports und schaltet damit alle LEDs aus.

6 LED per Pulsweitenmodulation dimmen

LEDs sind typische Bauteile zur Ausgabe von Signalen in der Digitalelektronik. Sie können zwei verschiedene Zustände annehmen, ein und aus, 0 und 1 oder `True` und `False`. Das Gleiche gilt für die als digitale Ausgänge definierten GPIO-Ports. Demnach wäre es theoretisch nicht möglich, eine LED zu dimmen.

Mit einem Trick ist es dennoch möglich, die Helligkeit einer LED an einem digitalen GPIO-Port zu regeln. Lässt man eine LED schnell genug blinken, nimmt das menschliche Auge das nicht mehr als Blinken wahr. Die als Pulsweitenmodulation bezeichnete Technik erzeugt ein pulsierendes Signal, das sich in sehr kurzen Abständen ein- und ausschaltet. Die Spannung des Signals bleibt immer gleich, nur das Verhältnis zwischen Level `False` (0 V) und Level `True` (+3,3 V) wird verändert. Das Tastverhältnis gibt das Verhältnis der Länge des eingeschalteten Zustands zur Gesamtdauer eines Schaltzyklus an.

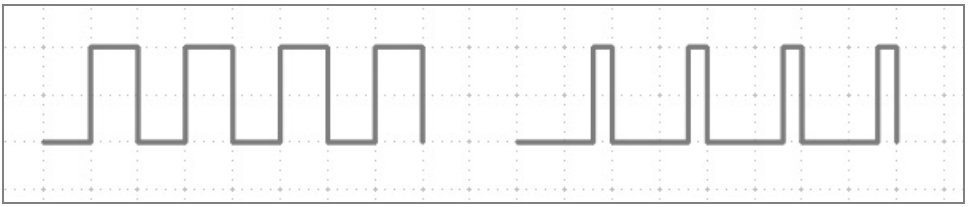

Abb. 6.1: Links: Tastverhältnis 50 % – rechts: Tastverhältnis 20 %.

Je kleiner das Tastverhältnis, desto kürzer ist die Leuchtzeit der LED innerhalb eines Schaltzyklus. Dadurch wirkt die LED dunkler als eine permanent eingeschaltete LED.

Schließen Sie für das nächste Experiment eine LED über einen Vorwiderstand an den GPIO-Port 18 an.

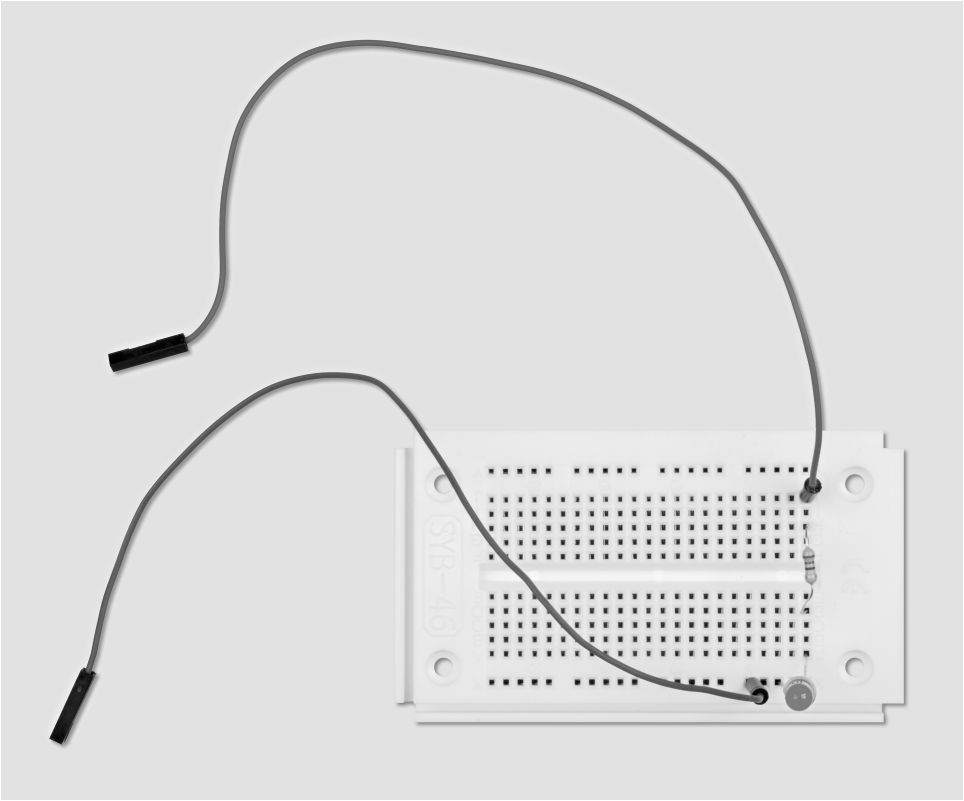

Abb. 6.2: Steckbrettaufbau mit einer LED.

Benötigte Bauteile:
1x Steckplatine
1x LED gelb
1x 220-Ohm-Widerstand
2x Verbindungskabel

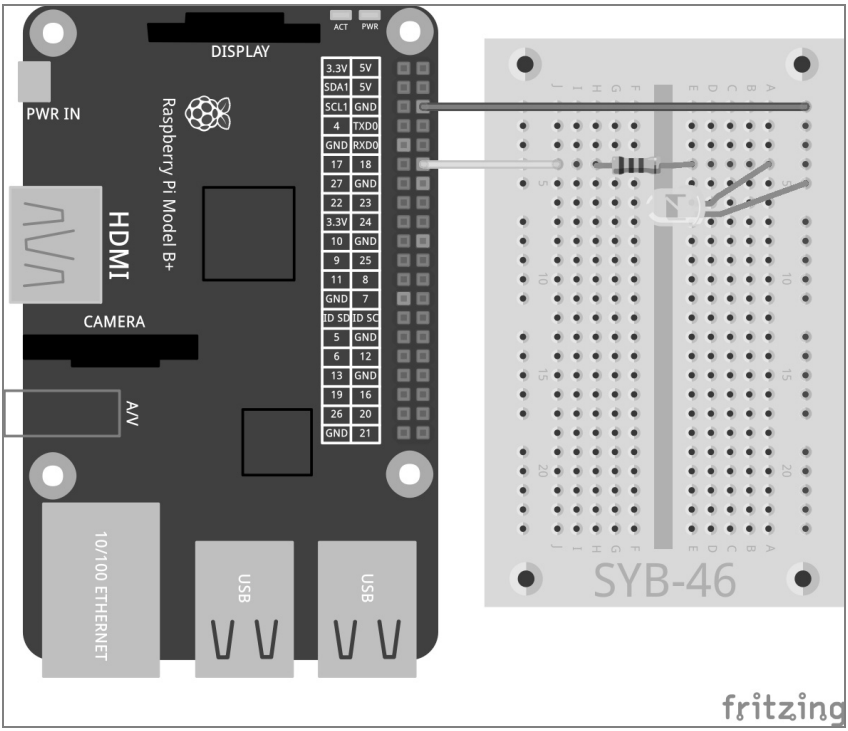

Abb. 6.3: Eine LED am GPIO-Port 18.

Das Programm `leddimmen01.py` dimmt die LED zyklisch heller und dunkler und verwendet dazu eine eigene PWM-Funktionalität der GPIO-Bibliothek. Das PWM-Signal wird als eigener Thread generiert. Auf diese Weise kann eine gedimmte LED (fast) wie eine normal leuchtende in einem Programm eingesetzt werden.

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM); LED = 18
GPIO.setup(LED, GPIO.OUT)
print ("Strg+C beendet das Programm")
p = GPIO.PWM(LED, 50); p.start(0)
try:
    while True:
        for c in range(0, 101, 2):
            p.ChangeDutyCycle(c); time.sleep(0.1)
        for c in range(100, -1, -2):
            p.ChangeDutyCycle(c); time.sleep(0.1)
except KeyboardInterrupt:
    p.stop(); GPIO.cleanup()
```

6.1.1 So funktioniert es

Ein Teil dieses Programms wird Ihnen bekannt vorkommen, einige Elemente aber auch gar nicht, da wir an dieser Stelle einen Exkurs in die objektorientierte Programmierung machen. Am Anfang werden, wie bekannt, Bibliotheken importiert. Diesmal wird nur eine einzige Variable, `LED`, für den GPIO-Port 18 festgelegt, dieser wird als Ausgang initialisiert.

`print ("Strg+C beendet das Programm")` Da auch dieses Programm mit einem `try...except`-Konstrukt läuft und vom Benutzer angehalten werden muss, wird eine entsprechende Information für den Benutzer angezeigt.

`p = GPIO.PWM(LED, 50)` Die Funktion `GPIO.PWM()` aus der GPIO-Bibliothek ist entscheidend für die Ausgabe von PWM-Signalen. Diese Funktion benötigt zwei Parameter, den GPIO-Port und die Frequenz des PWM-Signals. In unserem Fall wird der GPIO-Port über die Variable `LED` festgelegt, die Frequenz ist 50 Hertz (Schwingungen pro Sekunde).

Warum 50 Hertz die ideale Frequenz für PWM ist
Das menschliche Auge nimmt Lichtwechsel schneller als 20 Hertz nicht mehr wahr. Da das Wechselstromnetz in Europa eine Frequenz von 50 Hertz nutzt, blinken viele Beleuchtungskörper mit dieser Frequenz, die vom Auge nicht wahrgenommen wird. Würde eine LED mit mehr als 20 Hertz, aber weniger als 50 Hertz blinken, könnte es zu Interferenzen mit anderen Lichtquellen kommen, wodurch der Dimmeffekt nicht mehr gleichmäßig erschiene.

`GPIO.PWM()` erzeugt ein sogenanntes Objekt, das in der Variablen `p` gespeichert wird. Solche Objekte sind weit mehr als nur einfache Variablen. Objekte können verschiedene Eigenschaften haben und über sogenannte Methoden beeinflusst werden. Methoden werden, durch einen Punkt getrennt, direkt hinter dem Objektnamen angegeben.

`p.start(0)` Die Methode `start()` startet die Generierung des PWM-Signals. Dazu muss noch ein Tastverhältnis angegeben werden. In unserem Fall ist das Tastverhältnis `0`, die LED ist also immer ausgeschaltet. Jetzt startet die Endlosschleife, in der direkt nacheinander zwei Schleifen eingebettet sind, die abwechselnd die LED heller und dunkler werden lassen.

`for c in range(0, 101, 2)`: Die Schleife zählt in Schritten von `2` von `0` bis `100`. Als Ende einer `for`-Schleife wird immer der Wert angegeben, der gerade nicht erreicht wird, in unserem Fall `101`.

`p.ChangeDutyCycle(c)` In jedem Schleifendurchlauf setzt die Methode `ChangeDutyCycle()` das Tastverhältnis des PWM-Objekts auf den Wert des Schleifenzählers, also jedes Mal um 2 % höher, bis es beim letzten Durchlauf auf 100 % steht und die LED mit voller Helligkeit leuchtet.

`time.sleep(0.1)` In jedem Schleifendurchlauf werden 0,1 Sekunden gewartet, bevor der nächste Durchlauf das Tastverhältnis wieder um 2 % erhöht.

```
        for c in range(100, -1, -2):
            p.ChangeDutyCycle(c); time.sleep(0.1)
```

Nachdem die LED volle Helligkeit erreicht hat, regelt eine zweite Schleife sie nach dem gleichen Schema wieder herunter. Diese Schleife zählt von `100` an in Schritten von `-2` nach unten. Dieser Zyklus wiederholt sich, bis ein Benutzer die Tastenkombination Strg + C drückt.

```
except KeyboardInterrupt:
    p.stop(); GPIO.cleanup()
```

Der `KeyboardInterrupt` löst jetzt zusätzlich die Methode `stop()` des PWM-Objekts aus. Diese Methode beendet die Erzeugung eines PWM-Signals. Danach werden, wie aus den letzten Programmen bekannt, die GPIO-Ports zurückgesetzt.

6.1.2 Zwei LEDs unabhängig dimmen

Da für die Programmierung des PWM-Signals keine Programmzeit im Python-Skript benötigt wird, lassen sich auch mehrere LEDs unabhängig voneinander dimmen, wie das nächste Experiment zeigt. Schließen Sie dazu eine weitere LED über einen Vorwiderstand an den GPIO-Port 25 an.

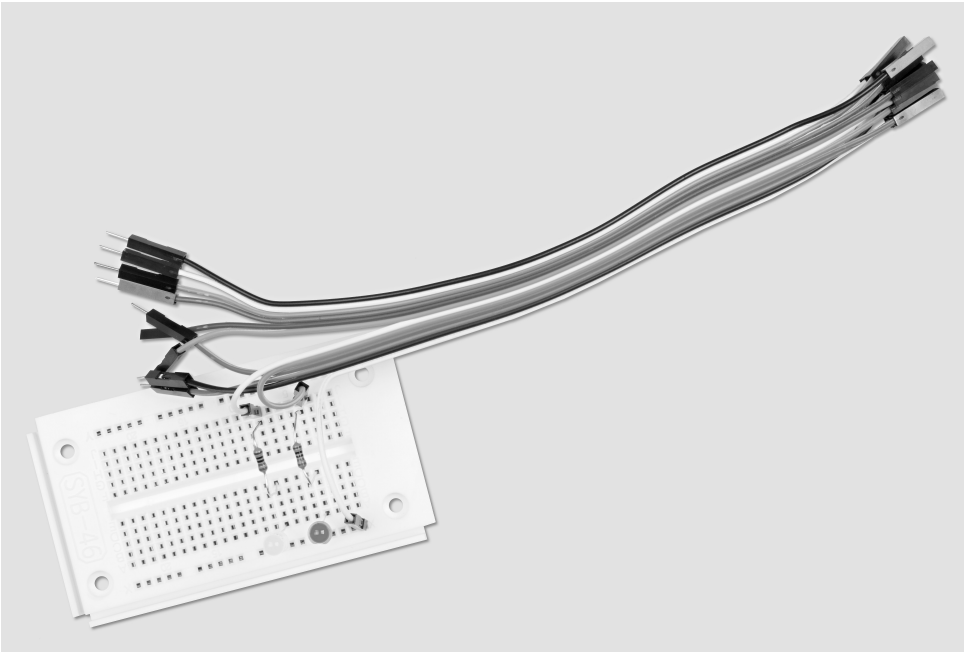

Abb. 6.4: Steckbrettaufbau, um zwei LEDs zu dimmen.

Benötigte Bauteile:
1x Steckplatine
1x LED gelb
1x LED rot
2x 220-Ohm-Widerstand
3x Verbindungskabel

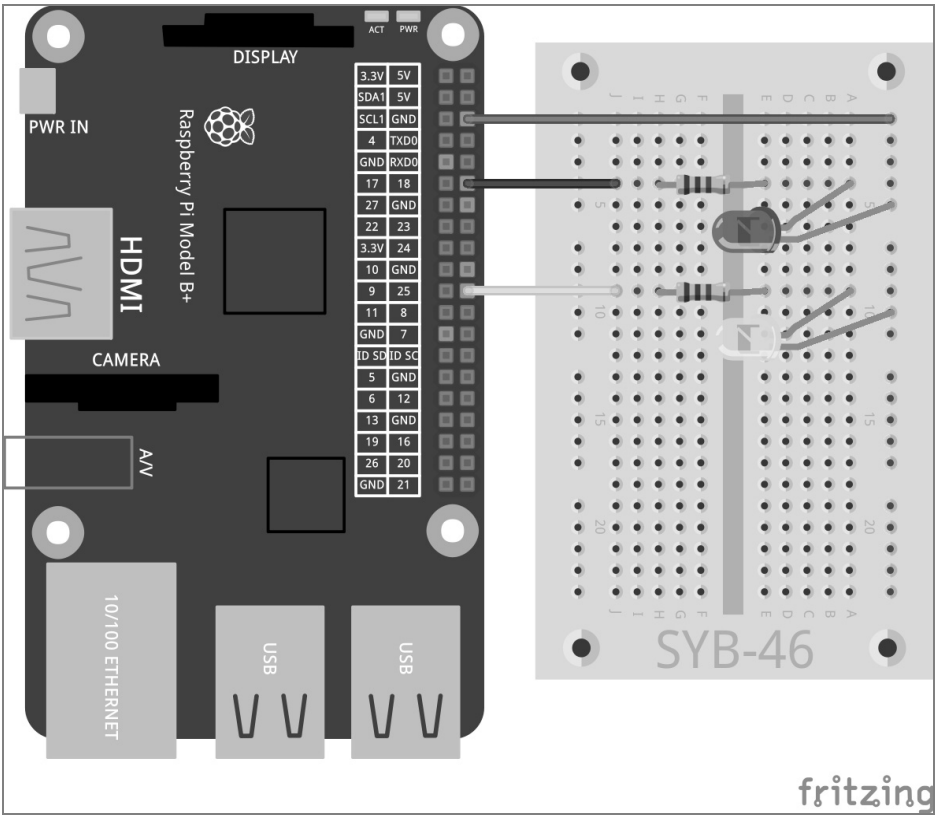

Abb. 6.5: Eine zweite LED am GPIO-Port 25.

Das Programm `leddimmen02.py` dimmt die eine LED zyklisch heller und dunkler, während die andere LED zwar zusammen mit der ersten LED heller, im anderen Zyklus aber nicht dunkler wird, sondern wieder von 0 an heller wird und dabei schnell flackert.

```
import RPi.GPIO as GPIO
import time
```

```
GPIO.setmode(GPIO.BCM); LED = [18,25]
GPIO.setup(LED[0], GPIO.OUT); GPIO.setup(LED[1], GPIO.OUT)

print ("Strg+C beendet das Programm")

p = GPIO.PWM(LED[0], 50); q = GPIO.PWM(LED[1], 50)
p.start(0)
q.start(0)

try:
    while True:
        for c in range(0, 101, 2):
            p.ChangeDutyCycle(c); q.ChangeDutyCycle(c)
            time.sleep(0.1)
        q.ChangeFrequency(10)
        for c in range(0, 101, 2):
            p.ChangeDutyCycle(100-c); q.ChangeDutyCycle(c)
            time.sleep(0.1)
        q.ChangeFrequency(50)

except KeyboardInterrupt:
    p.stop(); GPIO.cleanup()
```

6.1.3 So funktioniert es

Die Grundstruktur des Programms entspricht der des vorherigen Experiments mit ein paar kleinen Erweiterungen.

```
LED = [18,25]; GPIO.setup(LED[0], GPIO.OUT); GPIO.setup(LED[1], GPIO.OUT)
```

Statt einer Variablen für den GPIO-Port wird jetzt eine Liste mit zwei Variablen definiert, und damit werden zwei GPIO-Ports, `18` und `25`, als Ausgänge für die LEDs initialisiert.

```
p = GPIO.PWM(LED[0], 50); q = GPIO.PWM(LED[1], 50); p.start(0); q.start(0)
```

Anschließend werden auch die zwei Objekte `p` und `q` angelegt, die die PWM-Signale für die beiden LEDs mit jeweils 50 Hertz erzeugen.

```
        for c in range(0, 101, 2):
            p.ChangeDutyCycle(c); q.ChangeDutyCycle(c)
            time.sleep(0.1)
```

In der ersten Schleife werden die Tastverhältnisse beider PWM-Objekte gleichzeitig Schritt für Schritt erhöht. Die beiden LEDs verhalten sich in dieser Phase gleich.

`q.ChangeFrequency(10)` Am Ende dieser Schleife, wenn beide LEDs die volle Helligkeit erreicht haben, wird die Frequenz des PWM-Signals der zweiten LED über die Methode `ChangeFrequency()` auf 10 Hertz herabgesetzt. Diese Frequenz nimmt das Auge noch als Blinken wahr.

```
        for c in range(0, 101, 2):
            p.ChangeDutyCycle(100-c); q.ChangeDutyCycle(c)

            time.sleep(0.1)
```

Jetzt startet die zweite Schleife, der Übersichtlichkeit halber diesmal auch mit aufsteigender Zählung. Für die erste LED aus dem PWM-Objekt `p`, die in diesem Zyklus Schritt für Schritt abgedimmt werden soll, werden die entsprechenden Werte für das Tastverhältnis in jedem Durchlauf ausgerechnet. Bei der zweiten LED aus dem PWM-Objekt `q` wird das Tastverhältnis einfach wieder hochgezählt. Der Blinkeffekt entsteht durch die veränderte Frequenz.

`q.ChangeFrequency(50)` Am Ende der zweiten Schleife wird die Frequenz dieser LED wieder auf 50 Hertz zurückgesetzt, damit sie im nächsten Zyklus wieder genau wie die erste LED langsam heller wird.

7 Speicherkartenfüllstandsanzeige mit LEDs

Speicherkarten sind wie Festplatten immer viel zu schnell voll. Da wünscht man sich eine einfache optische Füllstandsanzeige, um immer auf einen Blick erkennen zu können, wann der Speicherplatz zur Neige geht. Mit drei LEDs lässt sich das auf dem Raspberry Pi sehr komfortabel realisieren. Dazu werden Funktionen des Betriebssystems genutzt, die über Python abgefragt werden.

Abb. 7.1: Natürlich lässt sich der freie Speicherplatz auch direkt im Dateimanager auf dem Raspberry Pi anzeigen.

Für die Anzeige des freien Speicherplatzes verwenden wir die drei LEDs aus der Ampelschaltung, die in unterschiedlichen Farbkombinationen leuchten.

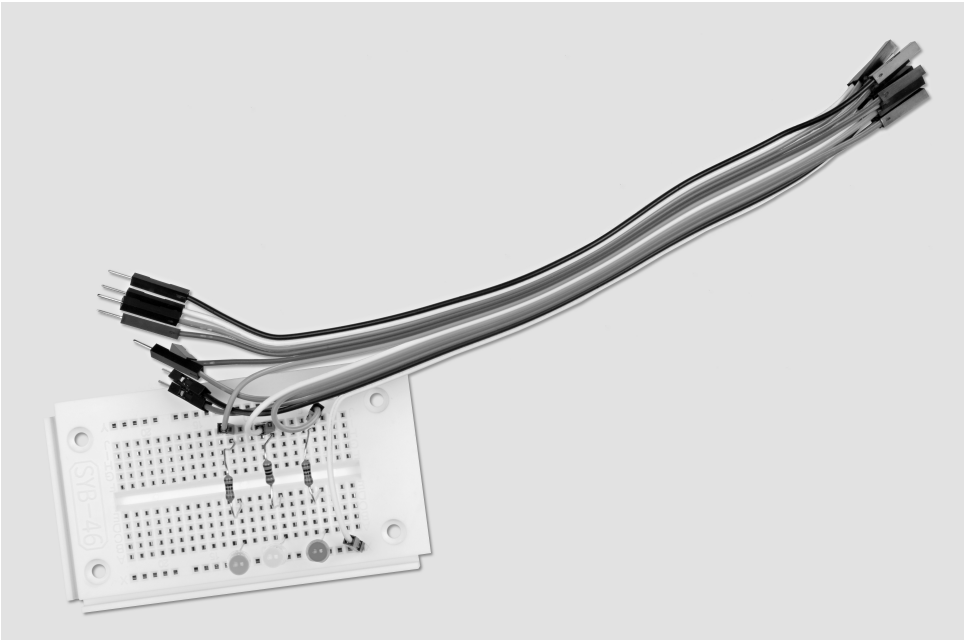

Abb. 7.2: Steckbrettaufbau für die Speicherkartenfüllstandsanzeige.

Benötigte Bauteile:
1x Steckplatine
1x LED rot
1x LED gelb
1x LED grün
3x 220-Ohm-Widerstand
4x Verbindungskabel

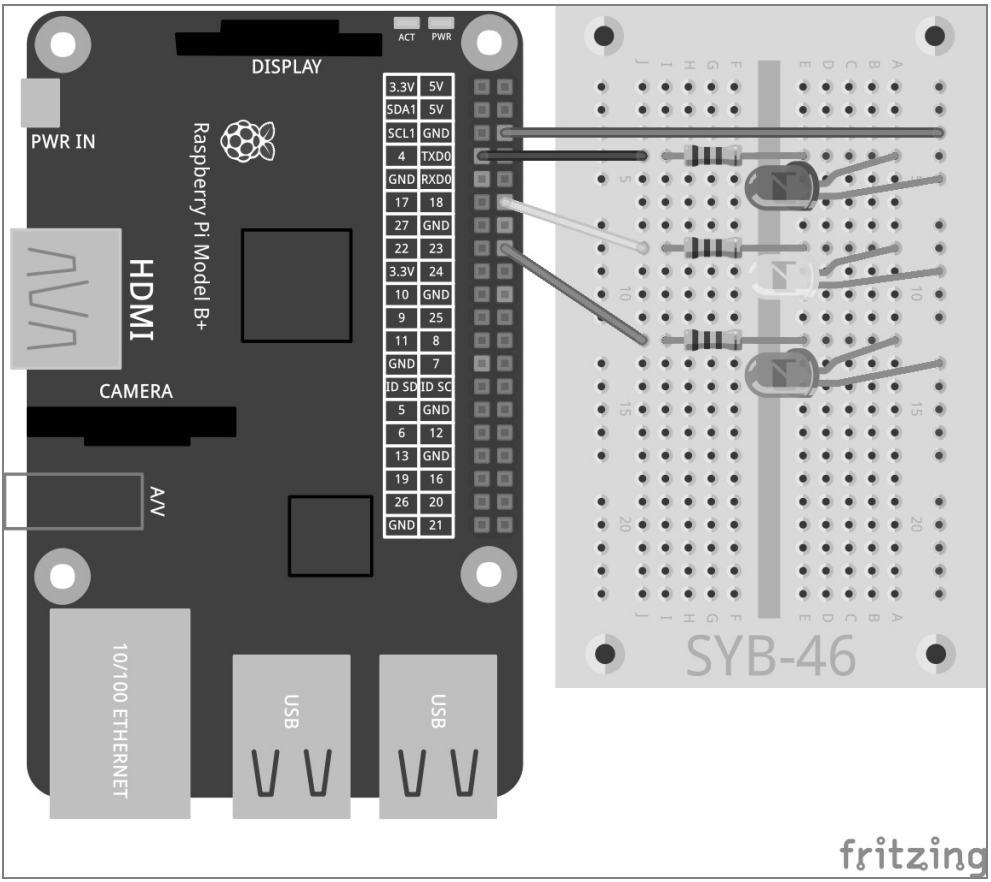

Abb. 7.3: Drei LEDs zeigen den freien Speicherplatz auf der Speicherkarte an.

Das Programm `speicheranzeige.py` liefert abhängig vom freien Speicherplatz auf der Speicherkarte unterschiedliche LED-Anzeigen:

Freier Speicherplatz	LED-Anzeige
< 1 MB	Rot
1 MB bis 10 MB	Rot-Gelb
10 MB bis 100 MB	Gelb
100 MB bis 500 MB	Gelb-Grün
> 500 MB	Grün

Tab. 7.1: So wird der Füllstand der Speicherkarte angezeigt.

```
import RPi.GPIO as GPIO
import time
import os

g1 = 1; g2 = 10; g3 = 100; g4 = 500

GPIO.setmode(GPIO.BCM)
LED = [4,18,23]
for i in range(3):
    GPIO.setup(LED[i], GPIO.OUT, initial=False)

print ("Strg+C beendet das Programm")

try:
  while True :
    s = os.statvfs('/')
    f = s.f_bsize * s.f_bavail / 1000000

    if f < g1:
      x = "100"
    elif f < g2:
      x = "110"
    elif f < g3:
      x = "010"
    elif f < g4:
      x = "011"
    else:
      x = "001"

    for i in range(3):
      GPIO.output(LED[i], int(x[i]))
    time.sleep(1.0)

except KeyboardInterrupt:
  GPIO.cleanup()
```

Lassen Sie das Programm laufen, zeigen die LEDs ständig den freien Speicherplatz auf der Speicherkarte an. Probieren Sie es aus, indem Sie große Dateien über das Netzwerk auf die Speicherkarte kopieren und wieder löschen. Die Anzeige aktualisiert sich automatisch.

7.1.1 So funktioniert es

Das Programm verwendet das Python-Modul `os` zur Berechnung des freien Speicherplatzes, das grundlegende Betriebssystemfunktionen zur Verfügung stellt.

`import os` Das Modul `os` muss, wie andere Module auch, am Anfang des Programms importiert werden.

`g1 = 1; g2 = 10; g3 = 100; g4 = 500` Diese Zeilen definieren die Grenzen der Bereiche für freien Speicherplatz, an denen die Anzeige umschalten soll. Der Einfachheit halber verwendet das Programm Megabyte und nicht Byte, da man sich diese Zahlen besser vorstellen kann. Sie können die Grenzen jederzeit anders festlegen, die vier Werte müssen nur in aufsteigender Größe angeordnet sein.

```
GPIO.setmode(GPIO.BCM)
LED = [4,18,23]
for i in range(3):
    GPIO.setup(LED[i], GPIO.OUT, initial=False)
```

Eine Liste definiert die GPIO-Portnummern der drei LEDs. Danach initialisiert eine Schleife die drei GPIO-Ports als Ausgänge und setzt alle LEDs auf ausgeschaltet.

Auch in diesem Experiment verwenden wir eine `try...except`-Konstruktion und eine Endlosschleife, um das Programm automatisch immer wieder laufen zu lassen, bis es der Benutzer mit Strg + C abbricht. Danach folgen die eigentlich interessanten Funktionen, die auf das Betriebssystem zugreifen und den freien Speicherplatz abfragen.

`s = os.statvfs('/')` Das Statistikmodul `os.statvfs()` aus der `os`-Bibliothek liefert diverse statistische Informationen zum Dateisystem, die hier innerhalb der Endlosschleife bei jedem Schleifendurchlauf aufs Neue als Objekt in die Variable `s` geschrieben werden.

`f = s.f_bsize * s.f_bavail / 1048576` Jetzt liefert die Methode `s.f_bsize` die Größe eines Speicherblocks in Byte. `s.f_bavail` gibt die Anzahl freier Blöcke an. Das Produkt aus beiden Werten gibt demnach die Anzahl freier Bytes an, die hier durch 1.048.576 geteilt wird, um die Anzahl freier Megabytes zu erhalten. Das Ergebnis wird in der Variablen `f` gespeichert.

```
    if f < g1:
      x = "100"
```

Ist der freie Speicherplatz kleiner als der erste Grenzwert (1 MB), wird die Zeichenfolge `x`, die das Muster der eingeschalteten LEDs angibt, auf `"100"` gesetzt. Die erste, rote LED soll leuchten. Das Muster ist eine einfache Zeichenkette aus den Ziffern 0 und 1.

```
    elif f < g2:
      x = "110"
    elif f < g3:
      x = "010"
    elif f < g4:
      x = "011"
```

Mithilfe von `elif`-Abfragen werden die weiteren Grenzwerte abgefragt und die LED-Muster entsprechend gesetzt, wenn die erste Frage nicht zutrifft, also mehr als 1 MB freier Speicherplatz vorhanden ist.

```
    else:

      x = "001"
```

Sollte keine der Abfragen zutreffen, also mehr freier Speicherplatz vorhanden sein, als der höchste Grenzwert angibt, wird das LED-Muster auf `"001"` gesetzt. Die letzte, grüne LED soll leuchten.

```
    for i in range(3):

      GPIO.output(LED[i], int(x[i]))
```

Eine Schleife legt die GPIO-Ausgabewerte für die drei LEDs fest. Nacheinander bekommen alle LEDs den Zahlenwert der jeweiligen Ziffer aus der Zeichenfolge, `0` oder `1`, zugewiesen. Die Werte `0` und `1` können genauso wie `False` und `True` verwendet werden, um GPIO-Ausgänge aus- oder einzuschalten. Die Funktion `int()` errechnet aus einem Zeichen dessen Zahlenwert. Das Zeichen wird über den Schleifenzähler `i` aus einer bestimmten Position der Musterzeichenkette ausgelesen.

`time.sleep(1.0)` Das Programm wartet 1 Sekunde bis zum nächsten Schleifendurchlauf. Um Performance zu sparen, können Sie auch längere Wartezeiten festlegen, bis die Berechnung des freien Speicherplatzes wiederholt werden soll.

An dieser Stelle beginnt die `while...True`-Schleife von Neuem. Sollte der Benutzer zwischenzeitlich die Tastenkombination Strg + C gedrückt haben, wird ein `KeyboardInterrupt` ausgelöst und die Schleife verlassen. Danach werden die GPIO-Ports geschlossen und damit die LEDs ausgeschaltet.

8 Grafischer Spielwürfel

Ein cooles Spiel braucht Grafik und nicht nur eine Textausgabe wie in Zeiten der allerersten DOS-Computer. Die Bibliothek PyGame liefert vordefinierte Funktionen und Objekte zur Grafikdarstellung und Spieleprogrammierung. Damit braucht man nicht mehr alles von Grund auf neu zu erfinden.

Für viele Spiele braucht man einen Würfel, aber oft ist gerade keiner griffbereit. Das nächste Programmbeispiel zeigt, wie einfach es ist, den Raspberry Pi mithilfe von Python und PyGame als Würfel zu benutzen:

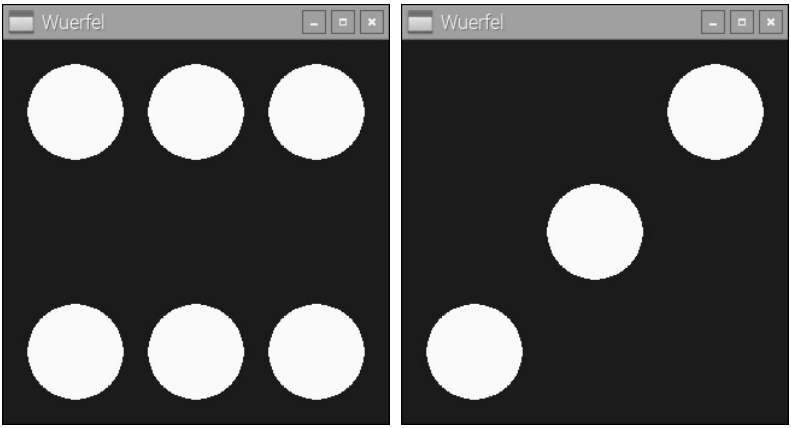

Abb. 8.1: Der Raspberry Pi als Würfel.

Der Würfel soll möglichst einfach und mit nur einer Taste zu bedienen sein, und das zufällig gewürfelte Ergebnis soll grafisch wie ein »echter« Würfel angezeigt werden. Das folgende Programm `wuerfel.py` simuliert einen solchen Würfel auf dem Bildschirm.

```
# -*- coding: utf-8 -*-
import pygame, sys, random
from pygame.locals import *
pygame.init()

FELD = pygame.display.set_mode((320, 320))
pygame.display.set_caption("Wuerfel")

BLAU = (0, 0, 255); WEISS = (255, 255, 255)
P1 = ((160, 160)); P2 = ((60, 60)); P3 = ((160, 60));
P4 = ((260, 60))
P5 = ((60, 260)); P6 = ((160, 260)); P7 = ((260, 260))
mainloop = True

print "Beliebige Taste drücken, um zu würfeln, [Esc] beendet das Spiel"

while mainloop:
    for event in pygame.event.get():
        if event.type == QUIT or (event.type == KEYUP and event.key == K_ESCAPE):
            mainloop = False
        if event.type == KEYDOWN:
            FELD.fill(BLAU)
            ZAHL = random.randrange (1, 7); print ZAHL
            if ZAHL == 1:
                pygame.draw.circle(FELD, WEISS, P1, 40)
            if ZAHL == 2:
                pygame.draw.circle(FELD, WEISS, P2, 40)
                pygame.draw.circle(FELD, WEISS, P7, 40)
            if ZAHL == 3:
                pygame.draw.circle(FELD, WEISS, P1, 40)
                pygame.draw.circle(FELD, WEISS, P4, 40)
                pygame.draw.circle(FELD, WEISS, P5, 40)
            if ZAHL == 4:
                pygame.draw.circle(FELD, WEISS, P2, 40)
                pygame.draw.circle(FELD, WEISS, P4, 40)
                pygame.draw.circle(FELD, WEISS, P5, 40)
                pygame.draw.circle(FELD, WEISS, P7, 40)
            if ZAHL == 5:
                pygame.draw.circle(FELD, WEISS, P1, 40)
                pygame.draw.circle(FELD, WEISS, P2, 40)
                pygame.draw.circle(FELD, WEISS, P4, 40)
                pygame.draw.circle(FELD, WEISS, P5, 40)
                pygame.draw.circle(FELD, WEISS, P7, 40)
            if ZAHL == 6:
                pygame.draw.circle(FELD, WEISS, P2, 40)
                pygame.draw.circle(FELD, WEISS, P3, 40)
                pygame.draw.circle(FELD, WEISS, P4, 40)
                pygame.draw.circle(FELD, WEISS, P5, 40)
                pygame.draw.circle(FELD, WEISS, P6, 40)
                pygame.draw.circle(FELD, WEISS, P7, 40)
    pygame.display.update()
pygame.quit()
```

8.1.1 So funktioniert es

Dieses Programm zeigt zahlreiche neue Funktionen, besonders zur Grafikausgabe mit der PyGame-Bibliothek, die natürlich nicht nur für Spiele, sondern auch für jegliche andere Grafik auf dem Bildschirm verwendet werden kann.

```
import pygame, sys, random
from pygame.locals import *
pygame.init()
```

Diese drei Programmzeilen stehen am Anfang fast jedes Programms, das PyGame verwendet. Neben dem bereits erwähnten Modul `random` zur Erzeugung von Zufallszahlen werden das Modul `pygame` selbst sowie das Modul `sys` geladen, da es wichtige, von PyGame benötigte Systemfunktionen enthält, wie z. B. das Öffnen und Schließen von Fenstern. Alle Funktionen aus der PyGame-Bibliothek werden importiert, und danach wird das eigentliche PyGame-Modul initialisiert.

```
FELD = pygame.display.set_mode((320, 320))
```

Diese wichtige Funktion in jedem Programm, das eine grafische Ausgabe nutzt, definiert eine Zeichenfläche, ein sogenanntes Surface, die in unserem Beispiel die Größe von 320 x 320 Pixeln hat und den Namen `FELD` bekommt. Beachten Sie die Schreibweise in doppelten Klammern, die grundsätzlich für grafische Bildschirmkoordinaten verwendet wird. Ein solches Surface wird in einem neuen Fenster auf dem Bildschirm dargestellt.

```
pygame.display.set_caption("Wuerfel")
```

Diese Zeile trägt den Fensternamen ein.

```
BLAU = (0, 0, 255); WEISS = (255, 255, 255)
```

Diese Zeilen definieren die beiden verwendeten Farben Blau und Weiß. Man könnte auch jedes Mal im Programm die Farbwerte direkt angeben, was aber nicht gerade zur Übersicht beiträgt.

Darstellung von Farben auf dem Bildschirm

Farben werden in Python, wie auch in den meisten anderen Programmiersprachen, durch drei Zahlen zwischen 0 und 255 definiert, die die drei Farbanteile Rot, Grün und Blau festlegen. Bildschirme verwenden eine additive Farbmischung, bei der alle drei Farbanteile in voller Sättigung zusammen Weiß ergeben.

```
P1 = ((160, 160)); P2 = ((60, 60)); P3 = ((160, 60)); P4 = ((260, 60)); P5 = ((60, ➲
260)); P6 = ((160, 260)); P7 = ((260, 260))
```

Diese Zeilen legen die Mittelpunkte der Würfelaugen fest. Auf dem 320 x 320 Pixel großen Zeichenfeld liegen die drei Achsen der Würfelaugen jeweils auf den Koordinaten `60`, `160` und `260`.

Das Koordinatensystem für Computergrafik
Jeder Punkt in einem Fenster bzw. auf einem Surface-Objekt wird durch eine x- und eine y-Koordinate bezeichnet. Der Nullpunkt des Koordinatensystems ist nicht, wie man in der Schule lernt, links unten, sondern links oben. Genau so, wie man einen Text von links oben nach rechts unten liest, erstreckt sich die x-Achse von links nach rechts, die y-Achse von oben nach unten.

Die sieben Punkte `P1` bis `P7` bezeichnen die in der Grafik angegebenen Mittelpunkte der Würfelaugen. Jedes Würfelauge hat einen Radius von 40 Pixeln. Bei 80 Pixeln Achsabstand bleiben demnach 20 Pixel zwischen den Würfelaugen und 20 Pixel zu den Fensterrändern.

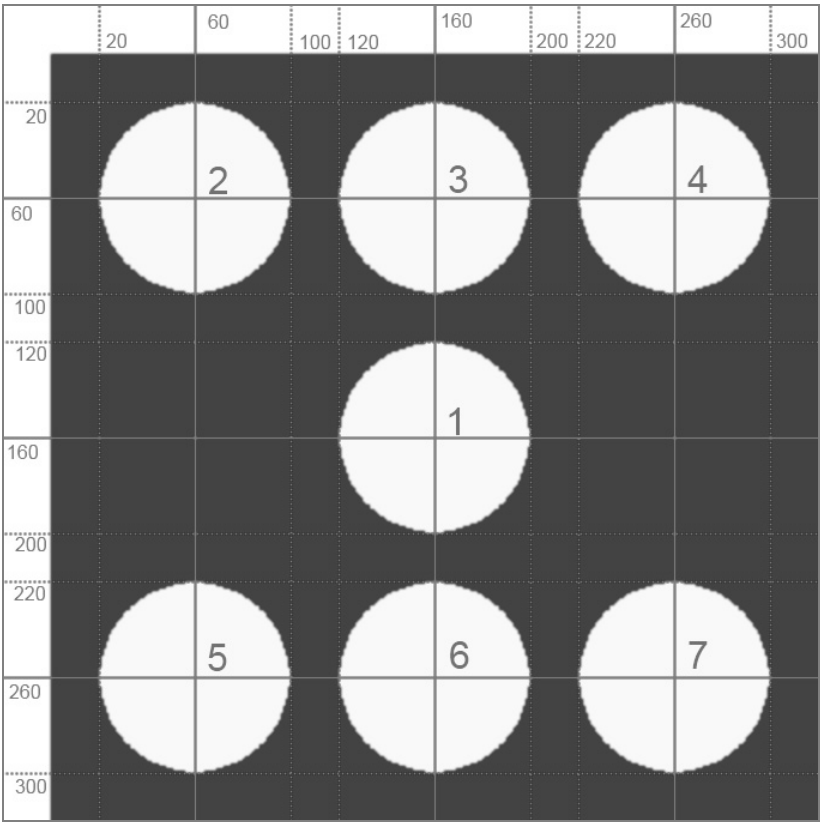

Abb. 8.2: Die Würfelaugen und ihre Koordinaten.

An dieser Stelle wird mit den anderen Variablen auch noch eine Variable namens `mainloop` auf `True` gesetzt, die später für die Hauptschleife des Spiels benötigt wird.

`mainloop = True` Damit sind die Grundlagen geschaffen, und das eigentliche Spiel kann beginnen.

```
print "Beliebige Taste drücken, um zu würfeln, [Esc] beendet das Spiel"
```

Diese Zeile erklärt dem Benutzer kurz, was zu tun ist. Bei jedem Druck auf eine beliebige Taste der Tastatur wird neu gewürfelt. `print` schreibt immer in das Python-Shell-Fenster, nicht in das neue grafische Fenster.

`while mainloop`: Jetzt beginnt die Hauptschleife des Spiels. In vielen Spielen wird eine Endlosschleife verwendet, die sich immer wiederholt und ständig irgendwelche Benutzeraktivitäten abfragt. Irgendwo in der Schleife muss eine Abbruchbedingung definiert sein, die dafür sorgt, dass das Spiel beendet werden kann.

Dafür wird hier die Variable `mainloop` verwendet, die nur die beiden booleschen Werte `True` und `False` (Wahr und Falsch, Ein und Aus) annimmt. Sie steht am Anfang auf `True` und wird bei jedem Schleifendurchlauf abgefragt. Hat sie während der Schleife den Wert `False` angenommen, wird die Schleife vor dem nächsten Durchlauf beendet.

`for event in pygame.event.get()`: Diese Zeile liest die letzte Benutzeraktivität und speichert sie als `event`. Im Spiel gibt es nur zwei Arten spielrelevanter Benutzeraktivitäten: Der Benutzer drückt eine Taste und würfelt damit, oder der Benutzer möchte das Spiel beenden.

```
        if event.type == QUIT or (event.type == KEYUP and event.key == K_ESCAPE):
            mainloop = False
```

Es gibt zwei Möglichkeiten, das Spiel zu beenden: Man kann auf das x-Symbol in der oberen rechten Fensterecke klicken oder die Taste Esc drücken. Wenn man auf das x-Symbol klickt, ist der vom Betriebssystem gelieferte `event.type == QUIT`. Wenn man eine Taste drückt und wieder loslässt, ist der `event.type == KEYUP`. Zusätzlich wird in diesem Fall die gedrückte Taste in `event.key` gespeichert.

Die beschriebene `if`-Abfrage prüft, ob der Benutzer das Fenster schließen will oder (`or`) eine Taste gedrückt und losgelassen hat und (`and`) dies die Taste mit der internen Bezeichnung `K_ESCAPE` ist. Ist das der Fall, wird die Variable `mainloop` auf `False` gesetzt, was die Hauptschleife des Spiels vor dem nächsten Durchlauf beendet.

`if event.type == KEYDOWN`: Die zweite Art von Benutzeraktivität, die während des Spiels immer wieder und nicht nur einmal vorkommt, ist, dass der Benutzer eine Taste drückt. Dabei spielt es keine Rolle, welche außer der Esc-Taste das ist. Sowie eine Taste gedrückt wurde (`KEYDOWN`), wird ein wichtiger Programmteil in Gang gesetzt, der das Würfelergebnis erzeugt und auch darstellt.

`FELD.fill(BLAU)` Als Erstes wird das als `FELD` bezeichnete Surface-Objekt, das eigentliche Programmfenster, mit der am Anfang als `BLAU` definierten Farbe gefüllt, um das vorherige Würfelergebnis zu übermalen.

```
            ZAHL = random.randrange (1, 7)
```

Jetzt generiert die Zufallsfunktion `random` eine Zufallszahl zwischen `1` und `6` und speichert sie in der Variablen `ZAHL`.

`print ZAHL` Diese Zeile schreibt nur zur Kontrolle das Würfelergebnis in das Python-Shell-Fenster. Sie können diese Zeile auch weglassen, wenn Sie auf die textbasierte Ausgabe verzichten wollen.

```
            if ZAHL == 1:
                pygame.draw.circle(FELD, WEISS, P1, 40)
```

Jetzt folgen, alle nach dem gleichen Schema, sechs Abfragen. Wenn die zufällig gewürfelte Zahl einen bestimmten Wert hat, werden dementsprechend ein bis sechs Würfelaugen gezeichnet. Die dazu verwendete Funktion `pygame.draw.circle()` benötigt vier oder fünf Parameter:

- *Surface* gibt die Zeichenfläche an, auf der gezeichnet wird, im Beispiel das `FELD`.
- *Farbe* gibt die Farbe des Kreises an, im Beispiel die zuvor definierte Farbe `WEISS`.
- *Mittelpunkt* gibt den Mittelpunkt des Kreises an.
- *Radius* gibt den Radius des Kreises an.
- *Dicke* gibt die Linienstärke der Kreislinie an. Wird dieser Parameter weggelassen oder auf `0` gesetzt, wird der Kreis gefüllt.

Ist eine der `if`-Bedingungen erfüllt, sind die Würfelaugen zunächst nur auf einer virtuellen Zeichenfläche gespeichert.

`pygame.display.update()` Erst diese Zeile am Schleifenende aktualisiert die Grafik auf dem Bildschirm. Nun sind die Würfelaugen wirklich zu sehen.

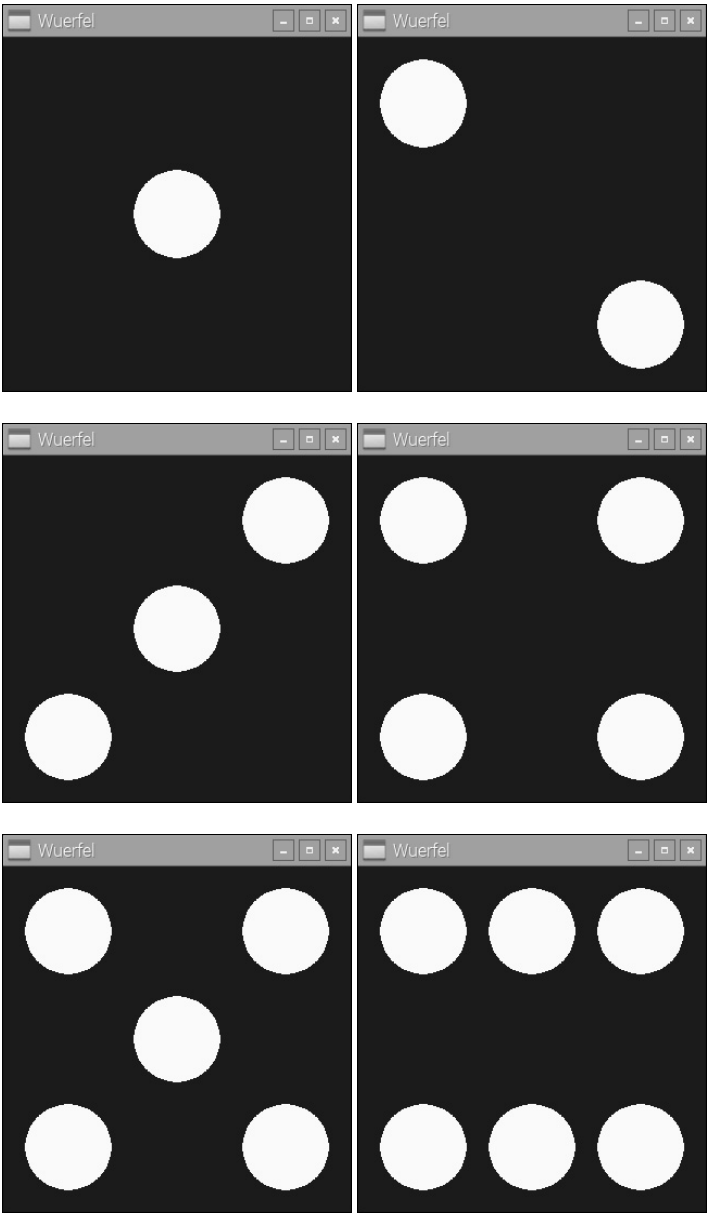

Abb. 8.3: Die sechs möglichen Würfelergebnisse.

Die Schleife startet sofort von Neuem und wartet wieder auf einen Tastendruck des Benutzers. Falls während der Schleife `mainloop` auf `False` gesetzt wurde, weil der Benutzer das Spiel beenden will, wird die Schleife kein weiteres Mal durchlaufen, stattdessen wird die folgende Zeile ausgeführt:

`pygame.quit()` Diese Zeile beendet das PyGame-Modul, was auch das grafische Fenster schließt und danach das ganze Programm.

9 Analoguhr auf dem Bildschirm

Die digitale Zeitanzeige, wie wir sie heute von Computern gewohnt sind, ist erst in den 70er-Jahren in Mode gekommen. Davor hat man jahrhundertelang die Uhrzeit analog mit Zeigern auf einem Zifferblatt angezeigt. Der Digitaluhrboom ist in den letzten Jahren schon wieder etwas zurückgegangen, da man erkannt hat, dass Analoguhren schneller und bei schlechten Wetterbedingungen oder auf große Entfernungen, wie zum Beispiel auf Bahnhöfen, auch klarer abzulesen sind. Das menschliche Auge erfasst eine Grafik schneller als Ziffern oder Buchstaben. Das Bild einer Analoguhr prägt sich ins Kurzzeitgedächtnis ein, sodass man es, auch wenn man es nur unvollständig oder verschwommen gesehen hat, dennoch richtig umsetzen kann. Sieht man dagegen eine Digitaluhr nur ungenau, kann man daraus keine zuverlässigen Rückschlüsse auf die angezeigte Zeit ziehen.

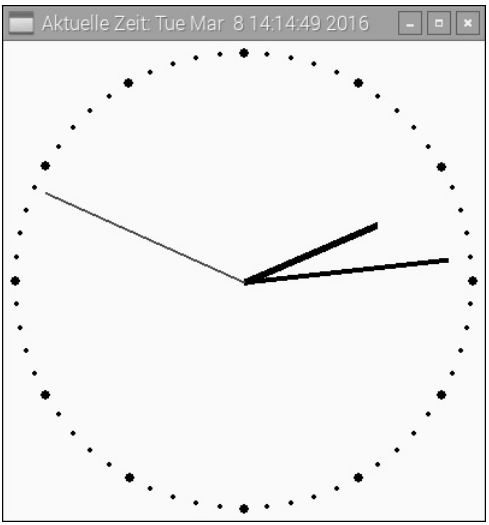

Abb. 9.1: Analoguhr, mit PyGame programmiert.

Dieses Programm soll nicht nur zeigen, wie man eine Uhr programmiert, sondern auch grundsätzliche Prinzipien zur Darstellung analoger Anzeigen verdeutlichen, wie sie nicht nur für Uhren, sondern auch zur Darstellung verschiedenster Messwerte oder statistischer Daten verwendet werden können.

Um den Mittelpunkt des runden Zifferblatts laufen drei Uhrzeiger, die die Stunde, die Minute und die Sekunde anzeigen. Oben im Fenstertitel läuft außerdem noch eine digitale Zeitanzeige mit.

Das Programm `uhr01.py` stellt die abgebildete Analoguhr auf dem Bildschirm dar:

```
import pygame, time
from pygame.locals import *
from math import sin, cos, radians
pygame.init()
ROT = (255, 0, 0); WEISS = (255, 255, 255); SCHWARZ = (0, 0, 0)
FELD = pygame.display.set_mode((400, 400))
FELD.fill(WEISS)
MX = 200; MY = 200; MP = ((MX, MY))
def punkt(A, W):
    w1 = radians(W * 6 - 90); x1 = int(MX + A * cos(w1))
    y1 = int(MY + A * sin(w1)); return((x1, y1))
for i in range(60):
    pygame.draw.circle(FELD, SCHWARZ, punkt(190, i), 2)
for i in range(12):
    pygame.draw.circle(FELD, SCHWARZ, punkt(190, i * 5), 4)
mainloop = True; s1 = 0
while mainloop:
    zeit = time.localtime()
    s = zeit.tm_sec; m = zeit.tm_min; h = zeit.tm_hour
    if h > 12:
        h = h - 12
    hm = (h + m / 60.0) * 5
    if s1 <> s:
        pygame.draw.circle(FELD, WEISS, MP, 182)
        pygame.draw.line(FELD, SCHWARZ, MP, punkt(120, hm), 6)
        pygame.draw.line(FELD, SCHWARZ, MP, punkt(170, m), 4)
        pygame.draw.line(FELD, ROT, MP, punkt(180, s), 2)
        s1 = s
        pygame.display.set_caption("Aktuelle Zeit: " + time.asctime())
        pygame.display.update()
        for event in pygame.event.get():
            if event.type == QUIT or (event.type == KEYUP and event.key == K_ESCAPE):
                mainloop = False
pygame.quit()
```

9.1.1 So funktioniert es

Dieses Programm zeigt weitere Funktionen der PyGame-Bibliothek und der `time`-Bibliothek sowie einfache trigonometrische Winkelfunktionen, die zur Darstellung analoger Anzeigen verwendet werden.

```
import pygame, time
from pygame.locals import *
from math import sin, cos, radians
pygame.init()
```

Am Anfang wird wie im letzten Programm die PyGame-Bibliothek initialisiert. Zusätzlich werden die `time`-Bibliothek zur Zeitermittlung sowie drei Funktionen aus der sehr umfangreichen `math`-Bibliothek importiert.

```
ROT = (255, 0, 0); WEISS = (255, 255, 255); SCHWARZ = (0, 0, 0)
```

Die drei in der Grafik verwendeten Farben werden in drei Variablen gespeichert.

```
FELD = pygame.display.set_mode((400, 400)); FELD.fill(WEISS)
```

Ein 400 x 400 Pixel großes Fenster wird geöffnet und komplett mit Weiß gefüllt.

```
MX = 200; MY = 200; MP = ((MX, MY))
```

Drei Variablen legen die Koordinaten des Mittelpunkts fest, auf den bezogen alle weiteren grafischen Elemente, das Zifferblatt und der Uhrzeiger ausgerichtet werden. Die Variablen `MX` und `MY` enthalten die x- und y-Koordinaten des Mittelpunkts, die Variable `MP` den Mittelpunkt als Punkt, wie er für grafische Funktionen verwendet wird.

Als Nächstes folgt die Definition einer wichtigen Funktion, die anhand des Abstands zum Mittelpunkt und eines Winkels Punkte im Koordinatensystem berechnet. Diese Funktion wird für die Darstellung sowohl des Zifferblatts als auch der Uhrzeiger mehrfach im Programm aufgerufen.

```
def punkt(A, W):
    w1 = radians(W * 6 - 90); x1 = int(MX + A * cos(w1))
    y1 = int(MY + A * sin(w1)); return((x1, y1))
```

Die Funktion verwendet zwei Parameter: `A` ist der Abstand des gewünschten Punkts vom Mittelpunkt, `W` der Winkel bezogen auf den Mittelpunkt. Um die Darstellung im Fall der Uhr zu vereinfachen, nehmen wir den Winkel im Uhrzeigersinn bezogen auf die senkrechte 12-Uhr-Richtung an. Der Winkel wird auch nicht in Grad, sondern in Minuten, 1/60 eines Vollkreises, an die Funktion übergeben. Solche Annahmen sparen viele Zwischenberechnungen.

Python rechnet wie die meisten Programmiersprachen Winkeleinheiten in Bogenmaß und nicht in Grad. Die Funktion `radian()` aus der `math`-Bibliothek rechnet Grad in Bogenmaß um. Dazu wird die im Programm verwendete Winkelangabe in Minuten mit 6 multipliziert, um auf Grad zu kommen, und anschließend wird 90 Grad subtrahiert, damit die 0-Richtung senkrecht nach oben zeigt, wie die Minute 0 jeder Stunde. Dieser Winkel umgerechnet in Bogenmaß wird für weitere Berechnungen innerhalb der Funktion in der Variablen `w1` gespeichert.

Die Anzeige einer Analoguhr basiert auf den Winkelfunktionen Sinus und Kosinus. Damit werden aus dem Winkel eines Punkts im Bogenmaß gegenüber dem Mittelpunkt (`w1`) dessen Koordinaten im rechtwinkligen Koordinatensystem (`x1` und `y1`) ermittelt. Die Koordinaten des Mittelpunkts werden aus den Variablen `MX` und `MY` übernommen, die außerhalb der Funktion definiert wurden und global gelten. Der Abstand des Punkts vom Mittelpunkt wird über den Parameter `A` der Funktion übergeben. Die Funktion `int()` ermittelt aus dem Ergebnis den Integerwert (Ganzzahl), da Pixelkoordinaten nur als Integer angegeben werden können.

Der Rückgabewert der Funktion ist ein geometrischer Punkt mit den errechneten Koordinaten `x1` und `y1`, der wie alle Punkte in doppelte Klammern gesetzt wird.

Nach der Definition dieser Funktion wird das Zifferblatt gezeichnet.

```
for i in range(60):
    pygame.draw.circle(FELD, SCHWARZ, punkt(190, i), 2)
```

Eine Schleife zeichnet nacheinander die `60` Minutenpunkte auf einem Kreis. Alle Punkte werden mit der Funktion `punkt()` ermittelt. Sie haben den gleichen Abstand vom Mittelpunkt, der mit `190` Pixeln in den vier Quadranten noch genau 10 Pixel vom Fensterrand entfernt ist. Die Punkte haben einen Radius von `2` Pixeln.

```
for i in range(12):
    pygame.draw.circle(FELD, SCHWARZ, punkt(190, i * 5), 4)
```

Eine zweite Schleife zeichnet `12` größere Kreise, die die Stunden auf dem Zifferblatt markieren. Diese haben einen Radius von `4` Pixeln, werden einfach über die vorhandenen Kreise gezeichnet und überdecken diese vollständig. Die Stundensymbole folgen im Winkelabstand von fünf Minuteneinheiten aufeinander, was man durch die Multiplikation mit `5` in der Winkelangabe, die an die Funktion übergeben wird, erreicht.

`mainloop = True; s1 = 0` Bevor die Hauptschleife des Programms startet, werden noch zwei Hilfsvariablen definiert, die im folgenden Verlauf benötigt werden. `mainloop` gibt wie im letzten Programmbeispiel an, ob die Schleife weiterlaufen soll oder der Benutzer das Programm beenden möchte. `s1` speichert die zuletzt angezeigte Sekunde.

```
while mainloop:
    zeit = time.localtime()
```

Jetzt startet die Hauptschleife des Programms, die in jedem Durchlauf, unabhängig davon, wie lange er dauert, die aktuelle Zeit in das Objekt `zeit` schreibt. Dazu wird die Funktion `time.localtime()` aus der `time`-Bibliothek verwendet. Das Ergebnis ist eine Datenstruktur, die aus verschiedenen Einzelwerten besteht.

```
    s = zeit.tm_sec; m = zeit.tm_min; h = zeit.tm_hour
```

Die drei für die Uhrzeiger relevanten Werte, Sekunden, Minuten und Stunden, werden aus der Struktur in drei Variablen `s`, `m` und `h` geschrieben.

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

Analoguhren zeigen nur zwölf Stunden an. Die Funktion `time.localtime()` liefert alle Zeitangaben im 24-Stunden-Format. Bei Zeitangaben am Nachmittag werden also einfach `12` Stunden subtrahiert.

Zeitdarstellung bei Analoguhren

Je nach verwendeter Mechanik gibt es zwei unterschiedliche Anzeigen bei Analoguhren. Bei echten analog laufenden Uhren führt der Minutenzeiger eine gleichförmige Kreisbewegung aus, bei digital gesteuerten Uhren, wie zum Beispiel Bahnhofsuhren, springt er zur vollen Minute um eine ganze Minute weiter. Letzteres Verfahren hat den Vorteil, dass die Uhrzeit besser auf einen Blick minutengenau abgelesen werden kann. Bruchteile von Minuten sind im Alltag normalerweise nicht wichtig. Wir verwenden für unser Uhrenprogramm ebenfalls dieses Verfahren. Der Stundenzeiger muss aber eine gleichförmige Kreisbewegung ausführen. Hier wäre es sehr befremdlich und unübersichtlich, würde er jede volle Stunde um eine ganze Stunde weiterspringen.

`hm = (h + m / 60.0) * 5` Die Variable `hm` speichert den Winkel des Stundenzeigers in Minuteneinheiten, wie sie im gesamten Programm verwendet werden. Dazu wird zur aktuellen Stunde 1/60 des Minutenwerts addiert. In jeder Minute bewegt sich der Stundenzeiger um 1/60 einer Stunde weiter. Der errechnete Wert wird mit `5` multipliziert, da der Stundenzeiger in einer Stunde fünf Minuteneinheiten auf dem Zifferblatt vorrückt.

`if s1 <> s:` Die Dauer eines Schleifendurchlaufs im Programm ist nicht bekannt. Für die Analoguhr bedeutet dies, dass die Grafik nicht bei jedem Schleifendurchlauf aktualisiert werden muss, sondern nur, wenn die aktuelle Sekunde eine andere ist als die zuletzt gezeichnete. Dazu wird später im Programm die gezeichnete Sekunde in der Variablen `s1` gespeichert, die aktuelle Sekunde steht immer in der Variablen `s`.

Wenn sich die Sekunde gegenüber der zuletzt gezeichneten verändert hat, wird mit den nun folgenden Anweisungen die Grafik der Uhr aktualisiert. Hat sie sich nicht verändert, braucht die Grafik nicht aktualisiert zu werden, und die Schleife startet von Neuem mit einer weiteren Abfrage der aktuellen Systemzeit.

```
        pygame.draw.circle(FELD, WEISS, MP, 182)
```

Als Erstes wird eine weiße Kreisfläche gezeichnet, die die Uhrzeiger restlos überdeckt. Der Radius ist mit 182 Pixeln etwas größer als der längste Zeiger, damit keine Reste davon mehr stehen bleiben. Einen vollflächigen Kreis zu zeichnen, ist deutlich einfacher, als den zuletzt gezeichneten Zeiger wieder pixelgenau zu übermalen.

```
      pygame.draw.line(FELD, SCHWARZ, MP, punkt(120, hm), 6)
```

Diese Zeile zeichnet den Stundenzeiger als Linie mit einer Breite von `6` Pixeln vom Mittelpunkt aus `120` Pixel lang im Winkel, der durch die Variable `hm` angegeben wird. Die Funktion `pygame.draw.line()` wurde bisher nicht verwendet. Sie benötigt fünf Parameter:

- *Surface* gibt die Zeichenfläche an, auf der gezeichnet wird, im Beispiel das `FELD`.
- *Farbe* gibt die Farbe des Kreises an, im Beispiel die zuvor definierte Farbe `SCHWARZ`.
- *Anfangspunkt* gibt den Anfangspunkt der Linie an, im Beispiel den Mittelpunkt der Uhr.
- *Endpunkt* gibt den Endpunkt der Linie an, im Beispiel wird dieser mit der Funktion `punkt()` aus dem Winkel des Stundenzeigers errechnet.
- *Dicke* gibt die Linienstärke an.

Die gleiche Funktion zeichnet auch die anderen beiden Zeiger der Uhr.

```
        pygame.draw.line(FELD, SCHWARZ, MP, punkt(170, m), 4)
```

Diese Zeile zeichnet den Minutenzeiger als Linie mit einer Breite von `4` Pixeln vom Mittelpunkt aus `170` Pixel lang im Winkel, der durch den Minutenwert angegeben wird.

```
        pygame.draw.line(FELD, ROT, MP, punkt(180, s), 2)
```

Diese Zeile zeichnet den Sekundenzeiger als rote Linie mit einer Breite von `2` Pixeln vom Mittelpunkt aus `180` Pixel lang im Winkel, der durch den Sekundenwert angegeben wird.

`s1 = s` Jetzt wird die gerade dargestellte Sekunde in der Variablen `s1` gespeichert, um diesen Wert in den nächsten Schleifendurchläufen mit der aktuellen Sekunde zu vergleichen.

```
        pygame.display.set_caption("Aktuelle Zeit: " + time.asctime())
```

Diese Zeile schreibt die aktuelle Uhrzeit in digitaler Form in den Fenstertitel. Dazu wird die Funktion `time.asctime()` aus der `time`-Bibliothek verwendet, die die Zeitangabe als fertig formatierte Zeichenkette liefert.

`pygame.display.update()` Bisher wurden alle Grafikelemente nur virtuell gezeichnet. Erst diese Zeile baut die Grafikanzeige wirklich neu auf. Die Aktualisierung erfolgt gleichzeitig. Deshalb ist auch keinerlei Flackern beim Zeichnen der einzelnen Zeiger nacheinander zu sehen.

```
        for event in pygame.event.get():
            if event.type == QUIT or (event.type == KEYUP and event.key == K_ESCAPE):
                mainloop = False
```

Immer noch innerhalb der `if`-Abfrage, also auch nur einmal in der Sekunde, erfolgt die relativ leistungshungrige Abfrage eventueller Systemereignisse, mit der hier festgestellt wird, ob der Benutzer innerhalb der letzten Sekunde das Uhrenfenster schließen wollte oder die Esc-Taste gedrückt hat. Ist das der Fall, wird die Variable `mainloop` auf `False` gesetzt, und damit wird die Schleife kein weiteres Mal gestartet.

`pygame.quit()` Die letzte Zeile beendet zunächst das PyGame-Modul, was auch das grafische Fenster schließt und danach das ganze Programm.

10 Grafische Dialogfelder zur Programmsteuerung

Kein modernes Programm, das irgendeine Interaktion mit dem Benutzer erfordert, läuft im reinen Textmodus. Überall gibt es grafische Oberflächen, auf denen man Buttons anklicken kann, anstatt Eingaben über die Tastatur vornehmen zu müssen.

Python selbst bietet keine grafischen Oberflächen für Programme, es gibt aber mehrere externe Module, ähnlich dem schon beschriebenen PyGame, die speziell dafür da sind, grafische Oberflächen zu erstellen. Eines der bekanntesten derartigen Module ist *Tkinter*, das die grafische Oberfläche *Tk*, die auch für diverse andere Programmiersprachen genutzt werden kann, für Python verfügbar macht.

Die Strukturen des grafischen Toolkits Tk unterscheiden sich etwas von Python und mögen auf den ersten Blick ungewöhnlich erscheinen. Deshalb fangen wir mit einem ganz einfachen Beispiel an: Eine LED soll über Buttons in einem Dialogfeld ein- und ausgeschaltet werden.

Benötigte Bauteile:
1x Steckplatine
1x LED rot
1x 220-Ohm-Widerstand
2x Verbindungskabel

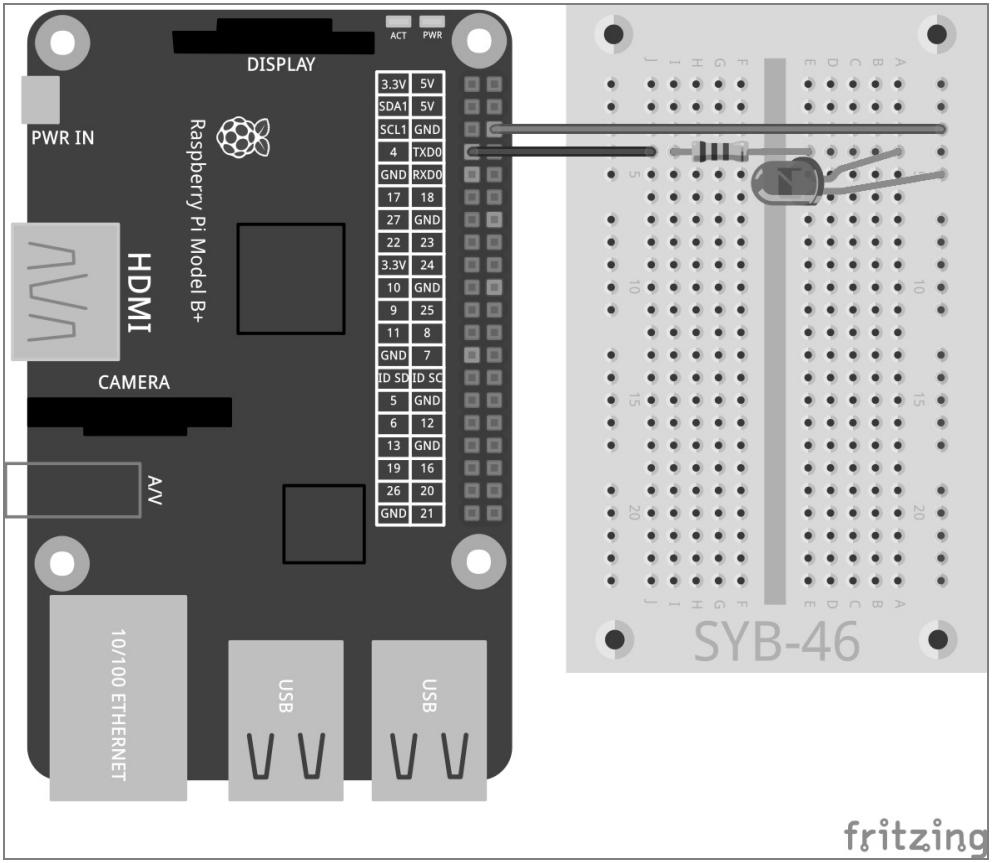

Abb. 10.1: Eine einzelne LED am GPIO-Port 4.

Schließen Sie eine LED über einen Vorwiderstand am GPIO-Port 4 an. Das Programm `ledtk01.py` wird diese zum Leuchten bringen.

```
import RPi.GPIO as GPIO
from Tkinter import *
LED = 4; GPIO.setmode(GPIO.BCM); GPIO.setup(LED,GPIO.OUT)
def LedEin():
    GPIO.output(LED,True)

def LedAus():
    GPIO.output(LED,False)

root = Tk(); root.title("LED")
Label(root, text="Bitte Button klicken, um die LED ein- und auszuschalten").pack()
Button(root, text="Ein", command=LedEin).pack(side=LEFT)
Button(root, text="Aus", command=LedAus).pack(side=LEFT)
root.mainloop()
GPIO.cleanup()
```

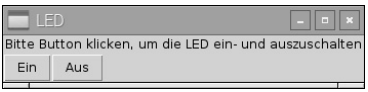

Abb. 10.2: So wird das fertige Dialogfeld aussehen.

10.1.1 So funktioniert es

Dieses Programm zeigt die Grundfunktionen der Tkinter-Bibliothek zum Aufbau grafischer Dialogfelder. Im Unterschied zur Grafikbibliothek PyGame, mit der Grafiken pixelgenau aufgebaut werden, ergibt sich die Größe der Dialogfelder und Steuerelemente in Tkinter aus der jeweils erforderlichen Größe automatisch, kann aber bei Bedarf auch nachträglich manuell beeinflusst werden.

```
import RPi.GPIO as GPIO
from Tkinter import *
```

Nach dem Import der GPIO-Bibliothek werden zusätzlich noch die Elemente der Tkinter-Bibliothek importiert.

```
LED = 4
GPIO.setmode(GPIO.BCM)
GPIO.setup(LED,GPIO.OUT)
```

Diese Zeilen zeigen nichts Neues. Der GPIO-Port `4` wird als Ausgabeport für eine LED definiert und mit der Variablen `LED` bezeichnet.

```
def LedEin():
    GPIO.output(LED,True)
```

Jetzt wird eine Funktion `LedEin()` definiert, die die LED einschaltet.

```
def LedAus():
    GPIO.output(LED,False)
```

Eine ähnliche Funktion, `LedAus()`, schaltet die LED wieder aus. Diese beiden Funktionen werden später über die beiden Buttons im Dialogfeld aufgerufen.

Bis hierhin war alles reines Python, jetzt geht es mit Tk und seinen Eigenheiten los.

`root = Tk()` Tkinter arbeitet mit sogenannten Widgets. Dabei handelt es sich um eigenständige Bildschirmelemente, in den meisten Fällen um Dialogfelder, die ihrerseits verschiedene Elemente enthalten. Jedes Programm braucht ein `root`-Widget, von dem aus alle weiteren Objekte aufgerufen werden. Dieses `root`-Widget heißt immer `Tk()`, generiert automatisch ein Fenster und initialisiert auch die Tkinter-Bibliothek.

`root.title("LED")` Objekte in Tkinter stellen verschiedene Methoden für unterschiedliche Zwecke zur Verfügung. Die Methode `title()` in einem Widget setzt den Fenstertitel, schreibt also in diesem Fall das Wort `LED` in die Titelzeile des neuen Fensters.

Jedes Widget kann mehrere Objekte enthalten, die einzeln definiert werden. Tkinter kennt dazu verschiedene Objekttypen, von denen jeder unterschiedliche Parameter ermöglicht, die die Eigenschaften des Objekts beschreiben. Die Parameter werden, durch Kommata getrennt, in einer Klammer hinter dem Objekttyp angegeben. Da diese Liste sehr lang werden kann, schreibt man üblicherweise jeden Parameter in eine eigene Zeile, sodass alle Parameter untereinander ausgerichtet sind. Im Gegensatz zu den Einrückungen bei Schleifen und Abfragen in Python sind diese Einrückungen der Tkinter-Objekte jedoch nicht obligatorisch.

```
Label(root, text="Bitte Button klicken, um die LED ein- und auszuschalten").pack()
```

Objekte vom Typ `Label` sind reine Texte in einem Widget. Diese können vom Programm verändert werden, bieten aber keine Interaktion mit dem Benutzer. Der erste Parameter in jedem Tkinter-Objekt ist der Name des übergeordneten Widgets, meistens des Fensters, in dem sich das jeweilige Objekt befindet. In unserem Fall ist das das einzige Fenster im Programm, das `root`-Widget.

Der Parameter `text` enthält den Text, der auf dem Label angezeigt werden soll. Am Ende der Objektdefinition wird der sogenannte Packer als Methode `pack()` angehängt. Dieser Packer baut das Objekt in die Dialogbox ein und generiert die Geometrie des Widgets.

```
Button(root, text="Ein", command=LedEin).pack(side=LEFT)
```

Objekte vom Typ `Button` sind Schaltflächen, die der Benutzer anklickt, um eine bestimmte Aktion auszulösen. Auch hier enthält der Parameter `text` den Text, der auf dem Button angezeigt werden soll.

Der Parameter `command` enthält eine Funktion, die der Button beim Anklicken aufruft. Dabei können keine Parameter übergeben werden, und der Funktionsname muss ohne Klammern angegeben werden. Dieser Button ruft die Funktion `LedEin()` auf, die die LED einschaltet.

Die Methode `.pack()` kann auch noch Parameter enthalten, die festlegen, wie ein Objekt innerhalb des Dialogfelds angeordnet werden soll. `side=LEFT` bedeutet, dass der Button linksbündig und nicht zentriert angeordnet wird.

```
Button(root, text="Aus", command=LedAus).pack(side=LEFT)
```

Nach dem gleichen Schema wird noch ein zweiter Button angelegt, der die LED über die Funktion `LedAus()` wieder ausschaltet.

Nun sind alle Funktionen und Objekte definiert, und das eigentliche Programm kann starten.

`root.mainloop()` Das Hauptprogramm besteht nur aus einer einzigen Zeile. Es startet die Hauptschleife `mainloop()`, eine Methode des `root`-Widgets. Diese Programmschleife wartet darauf, dass der Benutzer eines der Widgets betätigt und damit eine Aktion auslöst.

Das x-Symbol oben rechts zum Schließen des Fensters braucht bei Tkinter nicht eigens definiert zu werden. Schließt der Benutzer das Hauptfenster `root`, wird automatisch die Hauptschleife `mainloop()` beendet.

`GPIO.cleanup()` Das Programm läuft weiter zur letzten Zeile und schließt die geöffneten GPIO-Ports.

Nach dem Start des Programms erscheint ein Dialogfeld auf dem Bildschirm. Klicken Sie auf den Button *Ein*, um die LED einzuschalten, danach auf *Aus*, um sie wieder auszuschalten.

10.2 Lauflicht mit grafischer Oberfläche steuern

Die Python-Bibliothek Tkinter bietet noch weitaus mehr Steuerelemente als nur einfache Buttons. Über Radiobuttons lassen sich Auswahlmenüs bauen, in denen der Benutzer eine von mehreren angebotenen Optionen auswählen kann.

Was sind Radiobuttons?
Der Name »Radiobutton« stammt tatsächlich von alten Radios, auf denen es Stationstasten für vorprogrammierte Sender gab. Immer wenn man eine dieser Tasten gedrückt hat, sprang die zuletzt gedrückte durch eine raffinierte Mechanik automatisch wieder heraus. Radiobuttons verhalten sich genauso. Wählt der Benutzer eine Option, werden die anderen automatisch ausgeschaltet.

Das nächste Experiment zeigt verschiedene LED-Blinkmuster, die denen aus dem Experiment »Bunte LED-Muster und Lauflichter« ähneln. Im Unterschied zu dort braucht der Benutzer keine Zahlen auf dem Textbildschirm einzugeben, sondern kann komfortabel aus einer einfachen Liste das gewünschte Muster auswählen.

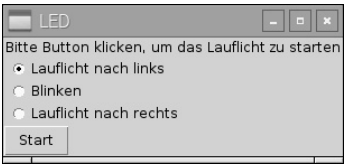

Abb. 10.3: Das Dialogfeld bietet drei LED-Muster zur Auswahl.

Der Aufbau der Schaltung ist der gleiche wie der im Experiment »Bunte LED-Muster und Lauflichter«.

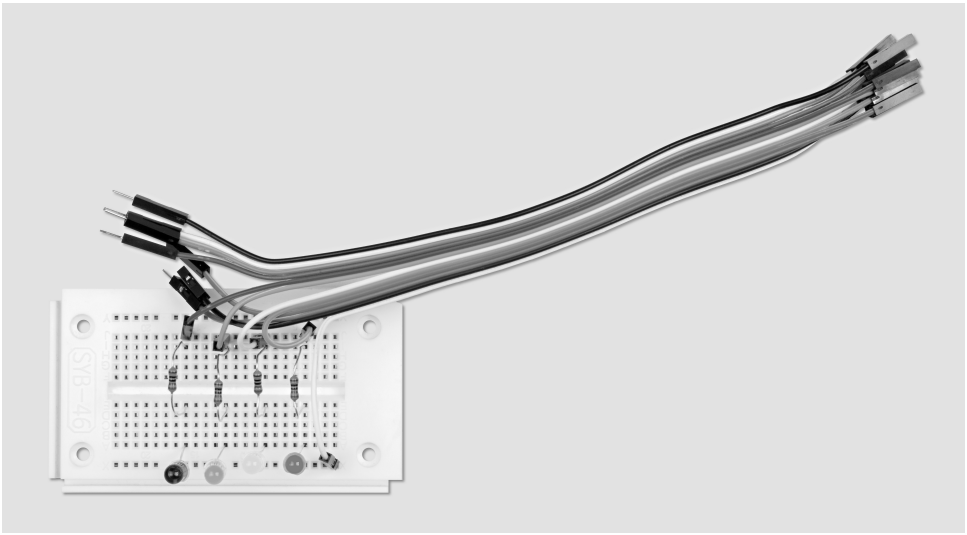

Abb. 10.4: Steckbrettaufbau zu Experiment 10.2.

Benötigte Bauteile:
1x Steckplatine
1x LED rot
1x LED gelb
1x LED grün
1x LED blau
4x 220-Ohm-Widerstand
5x Verbindungskabel

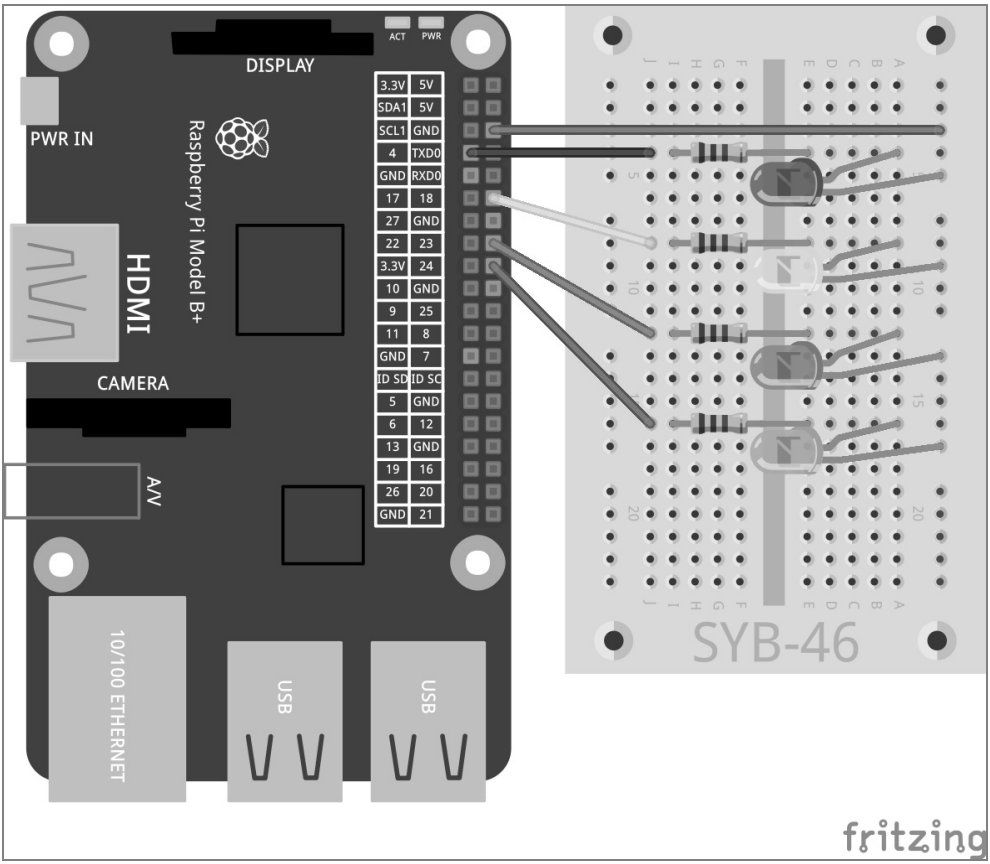

Abb. 10.5: Vier LEDs blinken in unterschiedlichen Mustern.

Das Programm `ledtk02.py` basiert auf dem vorherigen Programm, wurde aber um die Radiobuttons sowie die Funktionen für die LED-Lauflichter und Blinkmuster erweitert.

```
import RPi.GPIO as GPIO
import time
from Tkinter import *
GPIO.setmode(GPIO.BCM)
LED = [4,18,23,24]
for i in LED:
    GPIO.setup(i, GPIO.OUT, initial=0)
w = 5; t = 0.2
muster = [
    ("Lauflicht nach links",1),
    ("Blinken",2),
    ("Lauflicht nach rechts",3)
```

```
    ]
root = Tk(); root.title("LED"); v = IntVar(); v.set(1)
def LedEin():
    e = v.get()
    if e == 1:
        for i in range(w):
            for j in range(4):
                GPIO.output(LED[j], True)
                time.sleep(t)
                GPIO.output(LED[j], False)
    elif e == 2:
        for i in range(w):
            for j in range(4):
                GPIO.output(LED[j], True)
            time.sleep(t)
            for j in range(4):
                GPIO.output(LED[j], False)
            time.sleep(t)
    else:
        for i in range(w):
            for j in range(4):
                GPIO.output(LED[3-j], True); time.sleep(t)
                GPIO.output(LED[3-j], False)
Label(root,
      text="Bitte Button klicken, um das Lauflicht zu
starten").pack()
for txt, m in muster:
    Radiobutton(root, text = txt,
                variable = v, value = m).pack(anchor=W)
Button(root, text="Start", command=LedEin).pack(side=LEFT)
root.mainloop()
GPIO.cleanup()
```

10.2.1 So funktioniert es

Am Anfang werden wieder die notwendigen Bibliotheken importiert. Zusätzlich zum letzten Programm ist auch die `time`-Bibliothek dabei, die für die Wartezeiten bei den LED-Blinkeffekten benötigt wird.

```
GPIO.setmode(GPIO.BCM); LED = [4,18,23,24]
for i in LED:
    GPIO.setup(i, GPIO.OUT, initial=0)
```

Anschließend wird eine Liste für die vier LEDs definiert. Die entsprechenden GPIO-Ports werden als Ausgänge definiert und auf 0 gesetzt, damit alle LEDs am Anfang ausgeschaltet sind.

```
w = 5; t = 0.2
```

Zwei Variablen legen zwei Werte im Programm fest: die Anzahl der Wiederholungen `w` eines Musters sowie die Blinkzeit `t`. Beide Werte könnten auch bei jedem Vorkommen im Programm fest eingetragen werden. Auf diese Weise lassen sie sich aber leichter anpassen, da sie nur an einer Stelle definiert sind.

```
muster = [
    ("Lauflicht nach links",1), ("Blinken",2), ("Lauflicht nach rechts",3)
    ]
```

Die Texte der drei Muster, die zur Auswahl stehen, werden in einer besonderen Listenform definiert. Jedes der drei Listenelemente besteht aus einem Wertepaar, jeweils bestehend aus dem angezeigten Text und einem Zahlenwert, der später bei der Auswahl des jeweiligen Radiobuttons zurückgegeben werden soll.

```
root = Tk(); root.title("LED")
```

Die Initialisierung des `root`-Widgets entspricht wieder dem vorherigen Programm, nur die Inhalte des Dialogfelds unterscheiden sich.

```
v = IntVar(); v.set(1)
```

Variablen, die in Tk-Dialogfeldern genutzt werden, müssen im Gegensatz zu normalen Python-Variablen vor der ersten Verwendung deklariert werden. Diese beiden Zeilen deklarieren eine Variable `v` als Integer und setzen sie am Anfang auf den Wert `1`.

```
def LedEin():
    e = v.get()
```

Jetzt wird wieder eine Funktion definiert, die wie im letzten Beispiel `LedEin()` heißt, diesmal aber nicht nur eine LED einschaltet, sondern ein LED-Muster startet. Die zweite Funktion `LedAus()` aus dem letzten Beispiel wird hier nicht benötigt. Die erste Zeile der neuen Funktion liest die Benutzereingabe aus der Tk-Variablen `v` und schreibt den Wert in die Python-Variable `e`. Wie der Wert genau in die Variable `v` gelangt, erfahren Sie weiter unten bei der Erklärung der Radiobuttons.

Abhängig von der Benutzerauswahl werden drei verschiedene Programmschleifen gestartet:

```
    if e == 1:
        for i in range(w):
            for j in range(4):
                GPIO.output(LED[j], True); time.sleep(t)
                GPIO.output(LED[j], False)
```

Im ersten Fall läuft eine Schleife fünfmal durch, die nacheinander jede der vier LEDs einschaltet, 0,2 Sekunden leuchten lässt und wieder ausschaltet. Die fünf Wiederholungen und die 0,2 Sekunden Blinkzeit sind über die Variablen `w` und `t` am Anfang des Programms definiert.

```
    elif e == 2:
        for i in range(w):
            for j in range(4):
                GPIO.output(LED[j], True)
            time.sleep(t)
            for j in range(4):
                GPIO.output(LED[j], False)
            time.sleep(t)
```

Im zweiten Fall werden fünfmal hintereinander alle vier LEDs gleichzeitig eingeschaltet und, nachdem sie für 0,2 Sekunden geleuchtet haben, auch gleichzeitig wieder ausgeschaltet.

```
    else:
        for i in range(w):
            for j in range(4):
                GPIO.output(LED[3-j], True); time.sleep(t)
                GPIO.output(LED[3-j], False)
```

Der dritte Fall entspricht dem ersten, mit dem Unterschied, dass die LEDs rückwärts gezählt werden und dadurch das Lauflicht in umgekehrter Richtung läuft.

Nachdem die Funktion definiert ist, werden die Elemente der grafischen Oberfläche angelegt.

```
Label(root, text="Bitte Button klicken, um das Lauflicht zu starten").pack()
```

Der Text des Dialogfelds wird wieder als `Label`-Objekt definiert. Neu ist die Definition der drei Radiobuttons.

```
for txt, m in muster:
    Radiobutton(root, text = txt, variable = v, value = m).pack(anchor=W)
```

Die Radiobuttons werden über eine besondere Form der `for`-Schleife definiert. Anstelle eines Schleifenzählers sind hier zwei Variablen angegeben, die parallel gezählt werden. Die beiden Zähler durchlaufen nacheinander die Elemente der Liste `muster`. Dabei übernimmt die erste Zählvariable `txt` den ersten Wert des Wertepaars: den neben dem Radiobutton anzuzeigenden Text. Die zweite Zählvariable `m` übernimmt die Nummer des jeweiligen Musters aus dem zweiten Wert jedes Wertepaars.

Die Schleife legt auf diese Weise drei Radiobuttons an, deren erster Parameter immer `root` ist, das Widget, in dem die Radiobuttons liegen. Der Parameter `text` eines Radiobuttons gibt den anzuzeigenden Text an, der in diesem Fall aus der Variablen `txt` gelesen wird. Der Parameter `variable` legt eine zuvor deklarierte Tk-Variable fest, in die nach der Auswahl durch den Benutzer der Wert des ausgewählten Radiobuttons eingetragen wird.

Der Parameter `value` legt für jeden Radiobutton einen Zahlenwert fest, der in diesem Fall aus der Variablen `m` gelesen wird. Klickt ein Benutzer auf diesen Radiobutton, wird der Wert des Parameters `value` in die unter `variable` eingetragene Variable geschrieben. Jeder der drei Radiobuttons wird nach seiner Definition gleich mit der Methode `pack()` in das Dialogfeld eingebaut. Der Parameter `anchor=W` sorgt dafür, dass die Radiobuttons linksbündig untereinander ausgerichtet werden.

```
Button(root, text="Start", command=LedEin).pack(side=LEFT)
```

Der Button wird wie im letzten Beispiel definiert.

```
root.mainloop(); GPIO.cleanup()
```

Auch die Hauptschleife und das Programmende entsprechen dem letzten Beispiel.

Starten Sie das Programm und wählen Sie über einen der Radiobuttons ein Blinkmuster. Über die Variable `v` ist die erste Auswahl vorausgewählt. Wenn Sie Radiobuttons in einem Dialogfeld verwenden, sollten Sie immer eine sinnvolle Vorauswahl festlegen, damit es nie zu einem undefinierten Ergebnis kommt, falls der Benutzer selbst keine Auswahl trifft. Ein Klick auf *Start* startet anschließend das gewählte Muster und lässt es fünfmal laufen. Danach können Sie ein anderes Muster auswählen.

10.3 Blinkgeschwindigkeit einstellen

Im dritten Schritt wird das Dialogfeld nochmals erweitert. Der Benutzer kann jetzt über einen Schieberegler die Blinkgeschwindigkeit einstellen.

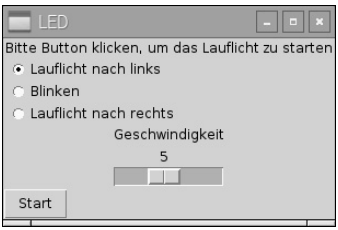

Abb. 10.6: Drei LED-Muster zur Auswahl und die einstellbare Blinkgeschwindigkeit.

Verwendung von Schiebereglern

Schieberegler bieten eine sehr intuitive Methode zur Eingabe von Zahlenwerten innerhalb eines bestimmten Bereichs. Auf diese Weise spart man sich eine Plausibilitätsabfrage, die ermittelt, ob der Benutzer einen Wert eingegeben hat, den das Programm auch sinnvoll umsetzen kann, da Werte außerhalb des durch den Schieberegler vorgegebenen Bereichs nicht möglich sind. Richten Sie den Schieberegler immer so ein, dass die Werte für den Benutzer vorstellbar sind. Es ergibt keinen Sinn, Werte im Millionenbereich einstellen zu lassen. Spielt der absolute Zahlenwert selbst keine wirkliche Rolle, geben Sie dem Benutzer einfach eine Skala von 1 bis 10 oder 100 vor und rechnen den Wert im Programm entsprechend um. Die Werte sollten von links nach rechts ansteigen, umgekehrt wirkt es für die meisten Benutzer befremdlich. Geben Sie außerdem immer einen sinnvollen Wert vor, der übernommen wird, wenn der Benutzer den Schieberegler nicht verändert.

Das Programm `ledtk03.py` entspricht weitgehend dem vorherigen Beispiel, nur die Regelung der Geschwindigkeit wird ergänzt.

```
import RPi.GPIO as GPIO
import time
from Tkinter import *

GPIO.setmode(GPIO.BCM); LED = [4,18,23,24]
for i in LED:
    GPIO.setup(i, GPIO.OUT, initial=0)

w = 5
muster = [
    ("Lauflicht nach links",1), ("Blinken",2), ("Lauflicht nach rechts",3)
    ]

root = Tk(); root.title("LED"); v = IntVar(); v.set(1); g = IntVar(); g.set(5)

def LedEin():
    e = v.get()
    t = 1.0/g.get()
    if e == 1:
        for i in range(w):
            for j in range(4):
                GPIO.output(LED[j], True); time.sleep(t)
                GPIO.output(LED[j], False)
    elif e == 2:
        for i in range(w):
            for j in range(4):
                GPIO.output(LED[j], True)
            time.sleep(t)
            for j in range(4):
                GPIO.output(LED[j], False)
            time.sleep(t)
    else:
        for i in range(w):
            for j in range(4):
                GPIO.output(LED[3-j], True): time.sleep(t)
                GPIO.output(LED[3-j], False)

Label(root,text="Bitte Button klicken, um das Lauflicht zu starten").pack()

for txt, m in muster:
    Radiobutton(root, text = txt, variable = v, value = m).pack(anchor=W)

Label(root, text="Geschwindigkeit").pack()

Scale(root, orient=HORIZONTAL, from_ = 1, to = 10, variable = g).pack()

Button(root, text="Start", command=LedEin).pack(side=LEFT)

root.mainloop()
GPIO.cleanup()
```

10.3.1 So funktioniert es

Die Initialisierung der Bibliotheken und GPIO-Ports sowie die Definition der Liste für die drei Blinkmuster entsprechen dem vorangegangenen Programm. Die Festlegung der Variablen `t` für die Blinkzeit fällt weg, da diese später aus dem Schieberegler ausgelesen wird.

`g = IntVar(); g.set(5)` Zusätzlich zur Tk-Variablen `v`, in der das ausgewählte Blinkmuster gespeichert wird, wird eine weitere Integervariable `g` für die Geschwindigkeit deklariert. Diese erhält einen Startwert von `5`, der dem Mittelwert des Schiebereglers entspricht.

```
def LedEin():
    e = v.get(); t = 1.0/g.get()
```

Die Funktion, die die LEDs blinken lässt, entspricht ebenfalls dem vorherigen Beispiel, jedoch mit einem Unterschied. Die Variable `t` für die Blinkdauer wird aus dem Wert des Schiebereglers `g` ermittelt.

Da ein Benutzer intuitiv ein schnelleres Blinken mit einer höheren Geschwindigkeit verbindet, wird der Schieberegler nach rechts größere Werte zurückliefern. Im Programm wird aber für eine höhere Geschwindigkeit eine kürzere Wartezeit, also ein kleinerer Wert benötigt. Dies wird durch eine Kehrwertberechnung erreicht, die aus den Werten `1` bis `10` des Schiebereglers die Werte `1.0` bis `0.1` für die Variable `t` ermittelt. In der Formel muss `1.0` und nicht `1` stehen, damit das Ergebnis eine Fließkommazahl und keine Ganzzahl wird.

Ganzzahlen in Fließkommawerte umrechnen
Das Ergebnis einer Berechnung wird automatisch als Fließkommazahl gespeichert, wenn mindestens einer der Werte in der Formel eine Fließkommazahl ist. Sind alle Werte in der Formel Ganzzahlen (Integer), wird das Ergebnis ebenfalls auf eine Ganzzahl gekürzt.

Die Definition des Labels und der Radiobuttons im Dialogfeld werden aus dem vorhergehenden Beispiel übernommen.

```
Label(root,
      text="Geschwindigkeit").pack()
```

Zur Erklärung des Schiebereglers wird ein weiteres Label in das Dialogfeld geschrieben. Da dieses keine Parameter in der `pack()` Methode enthält, wird es horizontal zentriert unterhalb der Radiobuttons eingebaut.

```
Scale(root, orient=HORIZONTAL, from_ = 1, to = 10, variable = g).pack()
```

Der Schieberegler ist ein Objekt vom Typ `Scale`, das wie alle Objekte in diesem Dialogfeld als ersten Parameter `root` enthält. Der Parameter `orient=HORIZONTAL` gibt an, dass der Schieberegler waagerecht liegt. Ohne diesen Parameter würde er senkrecht stehen. Die Parameter `from_` und `to` geben die Anfangs- und Endwerte des Schiebereglers an. Beachten Sie dabei die Schreibweise `from_`, da `from` ohne Unterstrich ein reserviertes Wort in Python für den Import von Bibliotheken ist. Der Parameter `variable` legt eine zuvor deklarierte Tk-Variable fest, in die der aktuell eingestellte Wert des Schiebereglers eingetragen wird. Der Anfangswert wird aus dem bei der Variablendeklaration festgelegten Wert, in diesem Fall `5`, übernommen.

Der Schieberegler wird mit der `pack()`-Methode auch wieder horizontal zentriert im Dialogfeld eingebaut.

Die weiteren Programmteile – der *Start*-Button, die Hauptschleife und das Programmende – werden unverändert aus dem vorherigen Beispiel übernommen.

Starten Sie das Programm, wählen Sie ein Blinkmuster und legen Sie die Geschwindigkeit fest. Höhere Werte lassen die Muster schneller blinken. Beim Klick auf den *Start*-Button liest die Funktion `LedEin()` das ausgewählte Blinkmuster aus den Radiobuttons sowie die Geschwindigkeit aus der Position des Schiebereglers aus.

11 PiDance mit LEDs

In den späten 70er-Jahren, noch vor der Zeit echter Computerspiele, gab es ein elektronisches Spiel mit vier farbigen Lampen, das es im Jahr 1979 sogar auf die allererste Auswahlliste zum Spiel des Jahres brachte. Das Spiel war in Deutschland unter dem Namen *Senso* auf dem Markt. Atari brachte einen Nachbau unter dem Namen *Touch Me* in der Größe eines Taschenrechners heraus. Ein weiterer Nachbau erschien als *Einstein*, im englischen Sprachraum wurde Senso als *Simon* vermarktet.

Raspbian liefert eine grafische Version dieses Spiels bei den *Python Games* unter dem Namen *Simulate* mit.

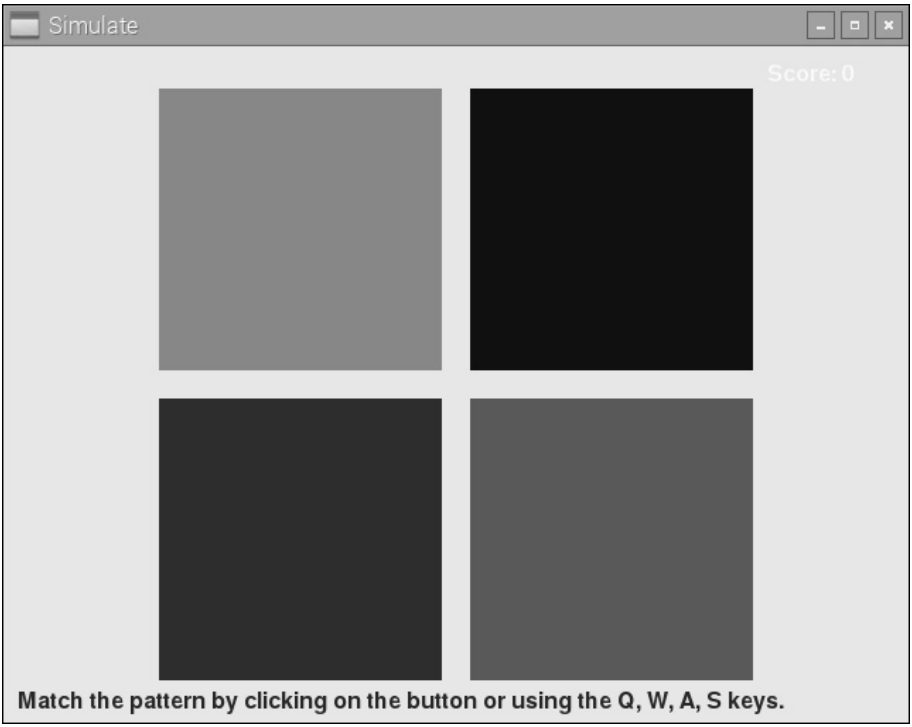

Abb. 11.1: Das Spiel Simulate aus den Python Games.

Unser Spiel PiDance basiert ebenfalls auf diesem Spielprinzip. LEDs blinken in einer zufälligen Reihenfolge. Der Benutzer muss die gleiche Reihenfolge anschließend mit den Tastern drücken. Mit jeder Runde leuchtet eine weitere LED, sodass es immer schwieriger wird, sich die Reihenfolge zu merken. Sobald man einen Fehler macht, ist das Spiel zu Ende.

Das Spiel wird auf zwei Steckplatinen aufgebaut, damit die Taster am Rand liegen und sich gut bedienen lassen, ohne dabei versehentlich Kabel aus den Steckplatinen zu ziehen. Zur besseren Stabilität lassen sich die Steckplatinen an den Längsseiten aneinanderstecken.

Zusätzlich zu den bereits bekannten Verbindungskabeln werden noch vier kurze Drahtbrücken benötigt. Schneiden Sie dazu mit einer scharfen Zange oder einer Drahtschere vom mitgelieferten Schaltdraht etwa 2,5 Zentimeter lange Stücke ab und entfernen an beiden Enden mit einem scharfen Messer jeweils etwa 7 Millimeter der Isolierung. Biegen Sie diese Drahtstücke in U-Form. Dann lassen sich damit jeweils zwei Reihen auf einer Steckplatine verbinden.

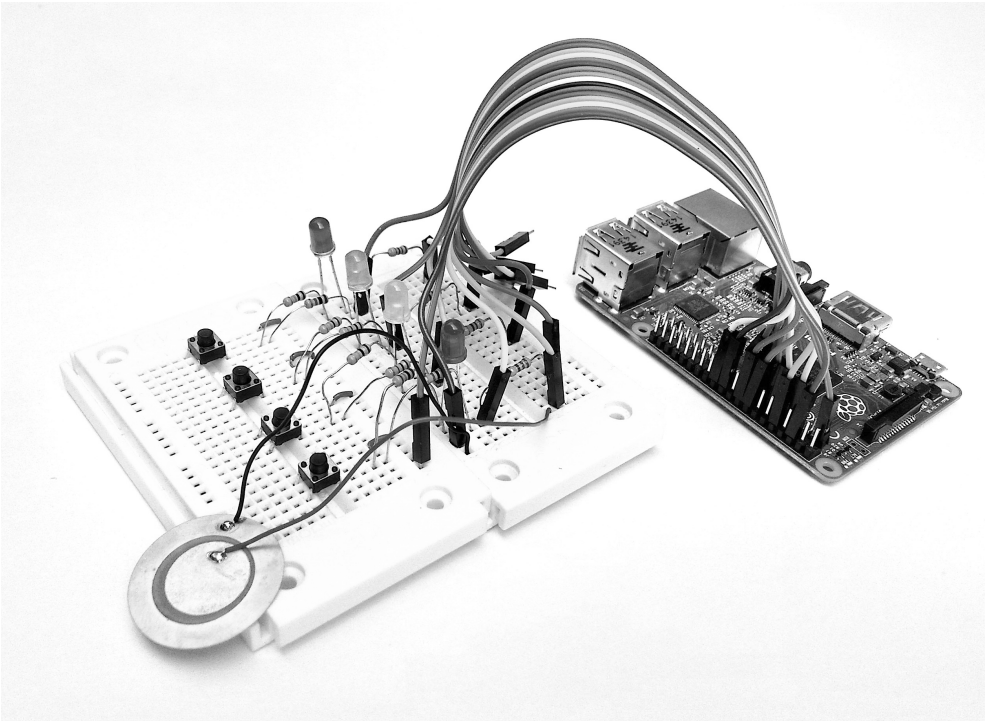

Abb. 11.2: Steckbrettaufbau zu Experiment 11.

Benötigte Bauteile:
2x Steckplatine
1x LED rot
1x LED gelb
1x LED grün
1x LED blau
4x 220-Ohm-Widerstand
4x 1-kOhm-Widerstand
4x 10-kOhm-Widerstand
4x Taster
10x Verbindungskabel
4x kurze Drahtbrücke

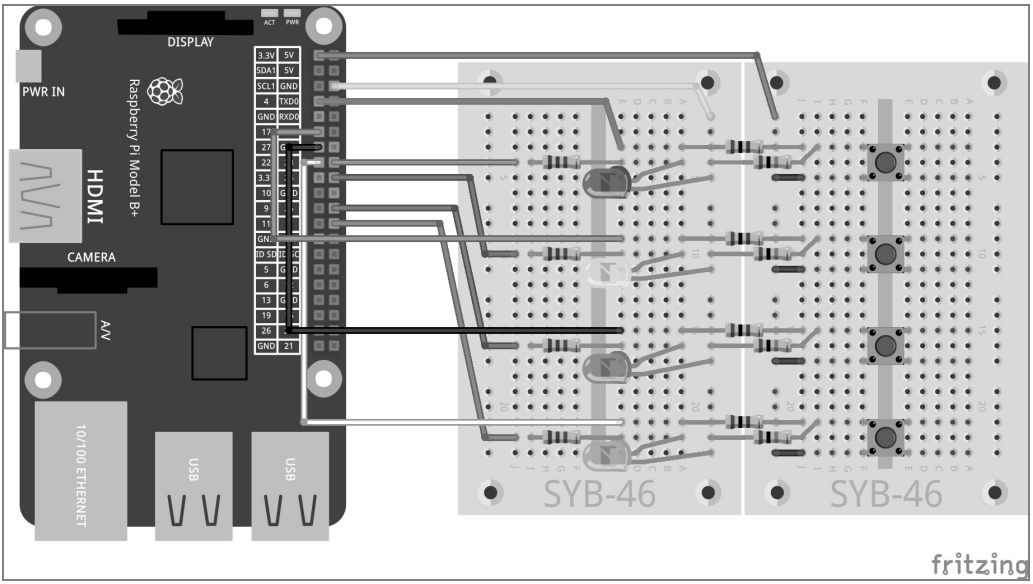

Abb. 11.3: PiDance mit LEDs und Tastern auf zwei Steckplatinen.

Die Taster sind gegenüber von den zugehörigen LEDs aufgebaut. Die mittleren beiden Längsreihen der Steckplatinen auf beiden Seiten der Verbindungsstelle dienen als 0-V- und +3,3-V-Leitung für die Schaltung.

Das Programm `pidance01.py` enthält das fertige Spiel.

```
# -*- coding: utf-8 -*-
import time, random
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
rzahl = 10; farbe = []
for i in range(rzahl):
    farbe.append(random.randrange(4))
LED = [23,24,25,8]
for i in LED:
    GPIO.setup(i, GPIO.OUT, initial=False)
TAST = [4,17,21,22]
for i in TAST:
    GPIO.setup(i, GPIO.IN)
def LEDein(n, z):
    GPIO.output(LED[n], True); time.sleep(z)
    GPIO.output(LED[n], False); time.sleep(0.15)
def Druecken():
```

```
    while True:
        if(GPIO.input(TAST[0])):
            return 0
        if(GPIO.input(TAST[1])):
            return 1
        if(GPIO.input(TAST[2])):
            return 2
        if(GPIO.input(TAST[3])):
            return 3
ok = True
for runde in range(1, rzahl +1):
    print "Runde", runde
    for i in range(runde):
        LEDein(farbe[i], 1)
    for i in range(runde):
        taste = Druecken()
        LEDein(taste, 0.2)
        if(taste != farbe[i]):
            print "Verloren!"
            print "Du hast es bis Runde ", runde - 1, " geschafft"
            for j in range(4):
                GPIO.output(LED[j], True)
            for j in range(4):
                time.sleep(0.5)
                GPIO.output(LED[j], False)
            ok = False
            break
    if(ok == False):
        break
    time.sleep(0.5)
if(ok == True):
    print "Super gemacht!"
    for i in range(5):
        for j in range(4):
            GPIO.output(LED[j], True)
        time.sleep(0.05)
        for j in range(4):
            GPIO.output(LED[j], False)
        time.sleep(0.05)
GPIO.cleanup()
```

11.1.1 So funktioniert es

Das Programm bietet viel Neues, die Grundlagen der GPIO-Steuerung sind aber bekannt.

`rzahl = 10` Nach dem Import der Module `time`, `random` und `RPi.GPIO` wird eine Variable `rzahl` angelegt, die die Anzahl zu spielender Runden festlegt. Natürlich können Sie auch mehr als zehn Runden spielen – je mehr Runden, desto schwieriger wird es, sich die Blinkfolge zu merken.

```
farbe = []
for i in range(rzahl):
    farbe.append(random.randrange(4))
```

Die Liste `farbe` wird über eine Schleife mit so vielen Zufallszahlen zwischen 0 und 3 gefüllt, wie Runden gespielt werden. Dazu wird die Methode `append()` verwendet, die in jeder Liste zur Verfügung steht. Diese hängt das als Parameter übergebene Element an die Liste an.

```
LED = [23,24,25,8]
for i in LED:
    GPIO.setup(i, GPIO.OUT, initial=False)
```

Die GPIO-Ports für die LEDs werden nach dem bekannten Schema in einer Liste `LED` als Ausgänge eingerichtet und alle ausgeschaltet.

```
TAST = [4,17,21,22]
for i in TAST:
    GPIO.setup(i, GPIO.IN)
```

Nach dem gleichen Prinzip werden die GPIO-Ports für die vier Taster in einer Liste `TAST` als Eingänge eingerichtet.

Damit sind die Grundlagen geschaffen, und es werden noch zwei Funktionen definiert, die im Programm mehrfach benötigt werden.

```
def LEDein(n, z):
    GPIO.output(LED[n], True); time.sleep(z)
    GPIO.output(LED[n], False); time.sleep(0.15)
```

Die Funktion `LEDein()` schaltet eine LED ein und lässt sie eine bestimmte Zeit leuchten. Die Funktion verwendet zwei Parameter. Der erste Parameter, `n`, gibt die Nummer der LED zwischen 0 und 3 an, der zweite Parameter, `z`, die Zeit, die die LED leuchten soll. Nachdem die LED wieder ausgeschaltet wurde, wartet die Funktion noch 0,15 Sekunden, bis sie beendet wird, um beim mehrfachen Aufrufen kurze Pausen zwischen dem Aufleuchten der LEDs zu sehen. Das ist besonders wichtig, wenn eine LED mehrfach hintereinander leuchtet. Dies wäre sonst nicht zu erkennen.

```
def Druecken():
    while True:
        if(GPIO.input(TAST[0])):
            return 0
        if(GPIO.input(TAST[1])):
            return 1
        if(GPIO.input(TAST[2])):
            return 2
        if(GPIO.input(TAST[3])):
            return 3
```

Die Funktion `Druecken()` besteht aus einer Endlosschleife, die darauf wartet, dass der Benutzer einen der Taster drückt. Danach wird die Nummer des Tasters an das Hauptprogramm zurückgegeben.

`ok = True` Nach der Definition der Funktionen startet das eigentliche Hauptprogramm und setzt als Erstes eine Variable `ok` auf `True`. Sobald der Spieler einen Fehler macht, wird `ok` auf `False` gesetzt. Ist die Variable nach der vorgegebenen Rundenzahl immer noch `True`, hat der Spieler gewonnen.

```
for runde in range(1, rzahl +1):
```

Das Spiel läuft über die in der Variablen `rzahl` festgelegte Anzahl von Runden. Der Rundenzähler ist dabei um 1 nach oben verschoben, damit das Spiel in Runde 1 beginnt und nicht in Runde 0.

```
    print "Runde", runde
```

Die aktuelle Runde wird im Python-Shell-Fenster angezeigt.

```
    for i in range(runde):
        LEDein(farbe[i], 1)
```

Jetzt spielt das Programm das Muster vor, das sich der Spieler merken muss. Je nach aktueller Rundenzahl leuchten nacheinander entsprechend viele LEDs gemäß der am Programmanfang festgelegten Liste `farbe` mit den zufällig gewählten Farben. Da der Zähler `runde` mit 1 beginnt, leuchtet bereits in der ersten Runde auch genau eine LED. Um die LED leuchten zu lassen, wird die Funktion `LEDein()` verwendet, deren erster Parameter die Farbe aus der entsprechenden Listenposition ist, der zweite Parameter lässt jede LED eine Sekunde lang leuchten.

`for i in range(runde)`: Nachdem das Farbmuster abgespielt wurde, startet eine weitere Schleife, in der der Spieler das gleiche Muster über die Taster aus dem Gedächtnis wieder eingeben muss.

`taste = Druecken()` Dazu wird die Funktion `Druecken()` aufgerufen, die wartet, bis der Spieler einen der Taster gedrückt hat. Die Nummer des gedrückten Tasters wird in der Variablen `taste` gespeichert.

`LEDein(taste, 0.2)` Nach dem Drücken einer Taste leuchtet die entsprechende LED kurz für 0,2 Sekunden auf.

`if(taste != farbe[i])`: Wenn die zuletzt gedrückte Taste nicht der Farbe an der entsprechenden Position in der Liste entspricht, hat der Spieler verloren. Der Operator `!=` steht für ungleich. Hier kann auch `<>` verwendet werden.

```
            print "Verloren!"
            print "Du hast es bis Runde ", runde - 1, " geschafft"
```

Das Programm zeigt auf dem Bildschirm an, dass der Spieler verloren hat und wie viele Runden er geschafft hat. Die Zahl der bestandenen Runden ist um eins kleiner als der aktuelle Rundenzähler.

```
            for j in range(4):
                GPIO.output(LED[j], True)
```

Als optisch sichtbares Zeichen werden alle LEDs eingeschaltet ...

```
            for j in range(4):
                time.sleep(0.5); GPIO.output(LED[j], False)
```

... dann wird eine nach der anderen im Abstand von 0,5 Sekunden wieder ausgeschaltet. So ergibt sich ein deutlicher Abbaueffekt.

`ok = False` Die Variable `ok`, die kennzeichnet, ob der Spieler noch im Spiel ist, wird auf `False` gesetzt ...

`break` ... und die Schleife abgebrochen. Der Spieler kann keine weiteren Tasten mehr drücken. Beim ersten Fehler ist sofort Schluss.

```
    if(ok == False):
        break
```

Wenn `ok` auf `False` steht, wird auch die äußere Schleife abgebrochen, es folgen keine weiteren Runden.

`time.sleep(0.5)` War die Eingabe der Sequenz richtig, wartet das Programm 0,5 Sekunden, bis die nächste Runde startet.

`if(ok == True)`: An dieser Stelle kommt das Programm an, wenn entweder die Schleife komplett durchgelaufen ist, der Spieler also alle Sequenzen richtig eingegeben hat, oder die vorherige Schleife durch einen Spielfehler abgebrochen wurde. Sollte `ok` noch auf `True` stehen, folgt die Siegerehrung. Andernfalls wird dieser Block übersprungen, und das Spiel führt nur noch die letzte Programmzeile aus.

```
    print "Super gemacht!"
    for i in range(5):
        for j in range(4):
            GPIO.output(LED[j], True)
        time.sleep(0.05)
        for j in range(4):
            GPIO.output(LED[j], False)
        time.sleep(0.05)
```

Im Gewinnfall erscheint eine Anzeige im Python-Shell-Fenster. Danach blinken alle LEDs fünfmal kurz hintereinander.

`GPIO.cleanup()` Die letzte Zeile wird auf jeden Fall ausgeführt. Hier werden die verwendeten GPIO-Ports geschlossen.

Impressum
© 2016 Franzis Verlag GmbH, Richard-Reitzner-Allee 2, 85540 Haar bei München
www.elo-web.de
Autor: Christian Immler

ISBN 978-3-645-10145-5

Elektrische und elektronische Geräte dürfen nicht über den Hausmüll entsorgt werden!
Entsorgen Sie das Produkt am Ende seiner Lebensdauer gemäß den geltenden gesetzlichen Vorschriften. Zur Rückgabe sind Sammelstellen eingerichtet worden, an denen Sie Elektrogeräte kostenlos abgeben können. Ihre Kommune informiert Sie, wo sich solche Sammelstellen befinden.

CE

Dieses Produkt ist konform zu den einschlägigen CE-Richtlinien, soweit Sie es gemäß der beiliegenden Anleitung verwenden. Die Beschreibung gehört zum Produkt und muss mitgegeben werden, wenn Sie es weitergeben.

Achtung! 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.