

Hands-On Computer Science

Nicolas Meseth

13. August 2025

Inhaltsverzeichnis

Vorwort	5
Was macht dieses Buch besonders?	5
Tipps für die Lektüre	6
Experimente	8
Hands-On Programmieren lernen	8
Spannende Experimente	9
Frust ist dein Freund – zumindest ein bisschen	10
1 Farben	12
Setup	12
1.1 Erste Schritte mit dem Brick Viewer	12
1.2 Unser erstes Programm	17
1.2.1 Programme	17
1.2.2 Boilerplate Code	18
1.2.3 Bibliotheken	18
1.2.4 Klassen und Objekte	18
1.2.5 Schlüsselwörter	19
1.2.6 Objekte erzeugen	19
1.2.7 Methoden	19
1.2.8 Ein Objekt für den Button	20
1.2.9 Zusammenfassung unseres ersten Programms	20
1.2.10 Und jetzt?	21
1.3 Farben und Licht	21
1.4 Farben im Computer	23
1.5 Farben mischen	23
1.5.1 Additive Farbmischung	23
1.5.2 Subtraktive Farbmischung	24
1.6 Der RGB-Code	25
1.7 Schleifen	25
1.7.1 Abzählbare Wiederholungen	27
1.7.2 Bedingte Wiederholung	28
1.8 Das fertige Programm	28

2 Eingaben	31
Setup	31
2.1 Das fertige Programm	31
3 Texte	33
Setup	33
3.1 Texte – ganz ohne Tastatur?	33
3.2 Klein anfangen: Ein einfacher Button mit Handgesten	34
4 Bilder	36
5 Codes	37
6 Umwandlung	38
Setup	38
6.1 Der Weg in den Computer hinein	38
6.2 Der Weg aus dem Computer heraus	38
7 Information	39
Setup	39
8 Sensoren	40
Setup	40
Aufgaben	40
9 Signale	41
Setup	41
9.1 Pulsmesser: Dein Finger als Signalquelle	41
9.2 Vom Diagramm zur Zahl	42
10 Protokolle	45
Setup	45
11 Verschlüsselung	46
Setup	46
12 Algorithmen	47
Setup	47
13 Kompression	48
Setup	48
14 Computer	49
Setup	49
14.1 Logik und Arithmetik	49

14.2 Die von-Neumann-Architektur	49
14.3 Der Arbeitsspeicher oder das Kurzzeitgedächtnis des Computers	49
15 Probleme	51
Setup	51
Literaturverzeichnis	52

Vorwort

Glückwunsch – du bist angekommen! Wie auch immer dein Weg hierher aussah, du hast es geschafft, dieses Buch zu öffnen. Vielleicht bist du Student oder Studentin an der Hochschule Osnabrück und wurdest (zu deinem Glück) gezwungen, oder du bist ganz bewusst hier gelandet und freust dich darauf, etwas Neues zu lernen – genau wie ich.

Dieses Buch entstand ursprünglich, um meinen Veranstaltungen an der Hochschule Osnabrück eine verständliche und praxisnahe Grundlage zu geben. Es dient als Hauptlektüre für meine Vorlesungen, aber auch als Nachschlagewerk für alle, die vielleicht mal eine Sitzung verpasst haben oder Themen eigenständig vertiefen wollen. Besonders willkommen sind dabei Quereinsteiger, Wiederholer oder einfach neugierige Menschen, die bisher noch gar keinen Kontakt mit der [Hochschule Osnabrück](#) hatten.

Hier bekommst du keine trockene Theorie präsentiert, sondern eine spannende, praxisnahe Einführung in die Grundlagen moderner Computer und unserer digitalen Welt. Das Fach, das sich dahinter verbirgt, heißt auf Deutsch Informatik, international auch bekannt als Computer Science. Der Titel Hands-On Computer Science verrät bereits: Hier wird es praktisch – und zwar von Anfang an.

Was macht dieses Buch besonders?

Lehrbücher zur Informatik gibt es reichlich. Viele davon sind großartig, aber kaum eines passt perfekt zu dem, was ich mit meinen Studierenden vorhave. Woran liegt das?

Viele klassische Informatikbücher versuchen, das gesamte Fachgebiet möglichst umfassend abzubilden. Das ist sinnvoll für angehende Informatiker, aber meine Zielgruppe bist du: Studierende in Studiengängen wie [Management nachhaltiger Ernährungssysteme](#), [Lebensmittelproduktion](#) oder [kAgrarsystemtechnologien](#) – oder vielleicht bist du nicht mal Student oder Studentin, sondern einfach interessiert daran, endlich Zugang zur digitalen Welt zu finden.

Kurz gesagt: Dieses Buch ist für jeden gedacht, der Lust hat, in die digitale Welt einzutauchen, ohne sich gleich mit komplizierten Details zu überfordern. Dafür brauchst du kein allumfassendes Nachschlagewerk, sondern einen klaren roten Faden, der dich Schritt für Schritt an die grundlegenden Konzepte heranführt.

Viele Bücher versprechen Praxisnähe, doch oft endet diese in nüchternen Übungsaufgaben am Kapitelende. Genau hier setzt *Hands-On Computer Science* an und macht zwei Dinge anders:

1. Du lernst informatische Konzepte direkt anhand spannender Projekte mit Hardware wie Microcontrollern, Sensoren, Buttons, LEDs und Displays kennen.
2. Du arbeitest kontinuierlich am LiFi-Projekt, das dich durch alle Kapitel begleitet und dabei immer weiter wächst.
3. Theorie und Praxis sind nicht getrennt, sondern eng miteinander verbunden – Programmieren und informatische Grundlagen lernst du gleichzeitig.

Schon ab Kapitel 1 beginnst du zu programmieren und zwar nicht abstrakt, sondern konkret mit Bauteilen wie Buttons. Im Laufe des Buches lernst du Schritt für Schritt neue Hardware-Komponenten kennen, die immer direkt mit relevanten informatischen Konzepten verknüpft sind. So schließt du am Ende nicht nur das LiFi-Projekt erfolgreich ab, sondern verfügst fast nebenbei über ein solides Fundament in der Informatik. Wenn alles gut läuft, merkst du kaum, wie schnell du gelernt hast.

Tipps für die Lektüre

Weil es in diesem Buch viel ums Programmieren geht, findest du natürlich viele Codeblöcke. Als Einstiegssprache verwenden wir Python. Warum ausgerechnet Python? Das erfährst du später genauer.

Codeblöcke sind deutlich sichtbar vom übrigen Text abgehoben, meist grau hinterlegt und in einer Schreibmaschinenschrift dargestellt, etwa *Courier New* oder *Consolas*. Hier ein kleines Beispiel:

```
led.set_rgb_value(0, 0, 0)                                     ①
led.set_rgb_value(255, 255, 255)                                ②

print("Diese Zeile hat keine Annotation")

# Lasse die LED blau aufleuchten                               ③
led.set_rgb_value(0, 0, 255)
```

- ① Schaltet die LED aus, weil der RGB-Code (0,0,0) schwarz erzeugt.
- ② Schaltet die LED auf weißes Licht, weil drei Mal die 255 die Farbe Weiß ergibt.
- ③ Auch Kommentare sind für kurze Erläuterungen nützlich.

Kommentare sind mit einer kleinen Zahl versehen. Wenn du die Online-Version nutzt und mit der Maus über diese Zahl fährst, erscheint ein Tooltip, der die Codezeile erklärt. Das funktioniert nur online, nicht in PDF oder Druckversion.

Noch ein kleiner Tipp: Wenn du mit der Maus über den Codeblock fährst, siehst du rechts oben ein Clipboard-Symbol. Ein Klick darauf kopiert den Code direkt in deine Zwischenablage, und du kannst ihn problemlos in dein geöffnetes Visual Studio Code oder eine andere IDE einfügen und ausprobieren.

Alle Codebeispiele findest du außerdem im [GitHub-Repository](#), das zu diesem Buch gehört.

Experimente

Hands-On Programmieren lernen

Hast du dich schon einmal gefragt, wie man Informationen über Licht übertragen kann? Oder wie man mit Licht den Puls messen kann? Oder wie man mit zwei einfachen Kabeln einen Wasserstandssensor baut? Das alles klingt vielleicht weit hergeholt, ist aber tatsächlich machbar – und wie genau, das wollen wir in diesem Buch herausfinden! Dabei werden wir nicht nur die digitale Welt der Computer und Programmierung kennenlernen, sondern auch mit spannenden Geräten in der analogen Welt arbeiten. In jedem Kapitel arbeiten wir mit anderen Geräten, die dir unterschiedliche Facetten der digitalen Welt näherbringen und gleichzeitig ermöglichen, das Programmieren spielerisch zu erlernen.

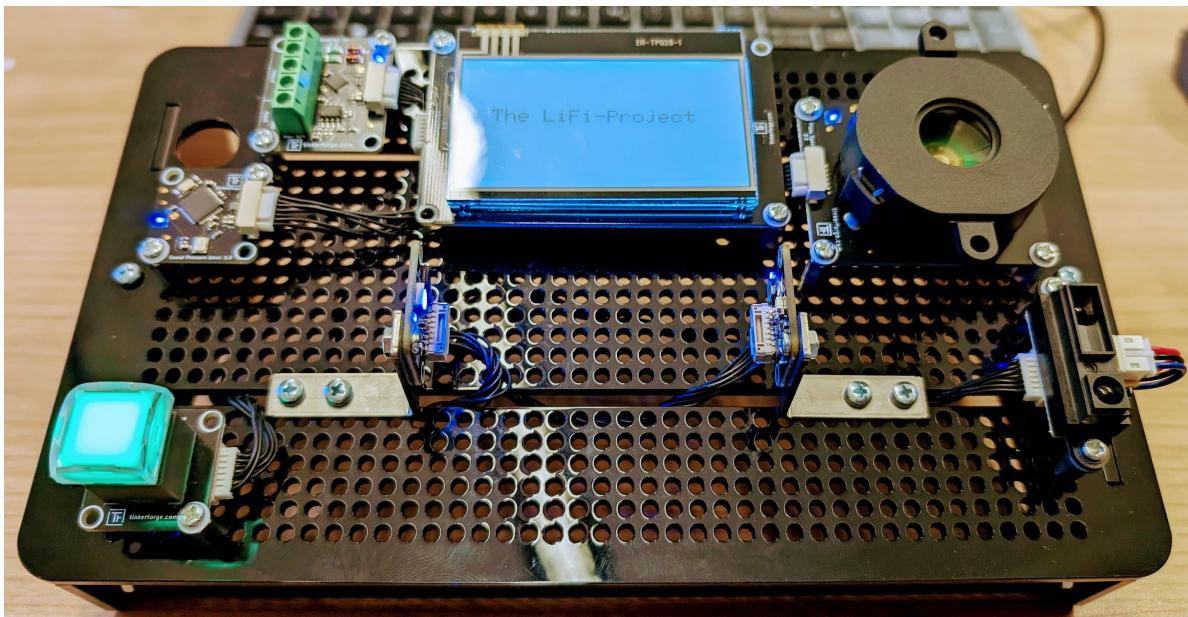


Abbildung 1: Tinkerforge Workbench mit vielen Geräten

Hier ein Überblick über die Geräte, mit denen wir gemeinsam experimentieren werden. Zusammengefasst kosten alle Komponenten 249 €. Aber keine Sorge: Wenn du das Buch im Rahmen meines Moduls „Digitalisierung und Programmierung“ an der Hochschule Osnabrück liest, erhältst du für das gesamte Semester ein komplettes Hardware-Kit.

Was?	Bauteil	Anzahl	Preis pro Stück
Bunte LED	RGB LED Bricklet 2.0	1	8 €
Button mit integrierter, bunter LED	RGB LED Button Bricklet	1	15 €
Licht- und Farbsensor	Color Bricklet 2.0	1	17 €
LCD Touchdisplay	LCD 128x64 Bricklet	1	33 €
Piezo Lautsprecher	Piezo Speaker Bricklet 2.0	1	19 €
Infrarot-Entfernungsmesser	Distance IR 4-30cm Bricklet 2.0	1	20 €
Analoger Spannungssensor	Analog In Bricklet 3.0	1	14 €
Schalldruckpegelsenor	Sound Pressure Level Bricklet	1	35 €
Mikrocontroller	Master Brick 3.2	2	35 €
Anschlusskabel 15 cm	Bricklet Kabel 15cm (7p-7p)	8	1 €
USB-A- auf USB-C Kabel	USB-A auf USB-C Kabel 100 cm	1	6 €
Montageplatte	Montageplatte 22x22 (12x12cm)	2	7 €
Schrauben, Abstandshalter und Muttern	Befestigungskit 12mm	4	2 €

Spannende Experimente

Kapitel für Kapitel werden wir an unterschiedlichen Experimenten arbeiten. Dabei lernst du nicht nur, wie man Hardware-Komponenten miteinander verbindet, sondern vor allem auch, wie man Computer – diese universellen Problemlösungsmaschinen – für eigene Ideen und Lösungen programmieren kann. Hier ist der Überblick, was dich in diesem Buch erwartet:

Kapitel	Experiment(e)
Kapitel 1	Wir lassen eine LED einen Regenbogenfarbverlauf über die Zeit erzeugen.
Kapitel 3	Wir lernen, wie man Texte umständlich und ohne Tastatur eingeben kann – über Handgesten.

Kapitel	Experiment(e)
Kapitel 4	Wir verbinden Tabellenkalkulation mit Bildern und Touchdisplays
Kapitel 5	Wir lernen Morse-Code und wie wir diesen über einen Lautsprecher ausgeben können.
Kapitel 6	Wir verwenden einfache Kippschalter, um analoge Werte in digitale Werte umzuwandeln.
Kapitel 7	
Kapitel 8	Wir bauen einen Wasserstandssensor mit einem analogen Spannungssensor.
Kapitel 9	Wir basteln einen Pulsmesser aus einem Farbsensor
Kapitel 10	Wir übertragen Nachrichten über Lichtsignale
Kapitel 11	Wir verstecken geheime Botschaften in harmlosen Nachrichten
Kapitel 12	
Kapitel 13	Wir entwickeln ein Verfahren, um Information zu komprimieren.
Kapitel 14	Wir entwickeln eine Rechenmaschine, die zwei Bytes addieren kann, mit nur einem einem Bauteil.
Kapitel 15	

Frust ist dein Freund – zumindest ein bisschen

Eins möchte ich gleich vorwegnehmen: Beim Programmierenlernen ist eine gewisse Portion Frust unvermeidbar. Klingt unangenehm? Ist es auch! Aber es ist zugleich Teil eines enorm wertvollen Lernprozesses. Jeder Fehler, den du machst, ist eine Gelegenheit, um zu verstehen, wie Computer wirklich funktionieren – nämlich absolut präzise und ohne jede Toleranz für Fehler.

Computer sind gnadenlose Lehrer. Sie zeigen dir sofort und unerbittlich, wenn etwas nicht stimmt – sei es ein vergessener Punkt, ein falscher Buchstabe oder ein simpler Zahlendreher. Das kann frustrieren, aber genau dieses direkte und sofortige Feedback hilft dir auch, schnell und effektiv zu lernen. Sobald du verstehst, wie du aus Fehlermeldungen sinnvolle Schlüsse ziehest und deine Programme entsprechend korrigierst, wirst du belohnt – mit Erfolgserlebnissen und einer steilen Lernkurve.

Also, wenn mal etwas nicht klappt: Nimm es nicht persönlich, sondern sieh es als Herausforderung. Atme tief durch, mach dir klar, dass Fehler unvermeidbar und sogar wichtig sind, und probier es noch einmal. Ich verspreche dir: Es lohnt sich!



Abbildung 2: Ein frustrierter Frosch

1 Farben

Das erste Kapitel hat es gleich in sich: Wir lernen etwas über Farben und wie sie im Computer funktionieren. Gleichzeitig steigen wir in die Programmierung ein und schreiben unser erstes Programm. Dabei nutzen wir eine LED und erzeugen einen Regenbogenfarbverlauf.

Setup

Bereit für dein erstes Hardware-Experiment? Du brauchst dafür eine LED ([RGB LED Bricklet 2.0](#)) und einen Mikrocontroller ([Master Brick 3.2](#)). Befestige beide Bauteile mit Abstandshaltern auf einer Montageplatte, wie in Abbildung [1.1](#) gezeigt. Zwei Halterungen pro Gerät reichen völlig. Denk an die kleinen, weißen Kunststoff-Unterlegscheiben – die schützen deine Platinen vor Druckstellen.

1.1 Erste Schritte mit dem Brick Viewer

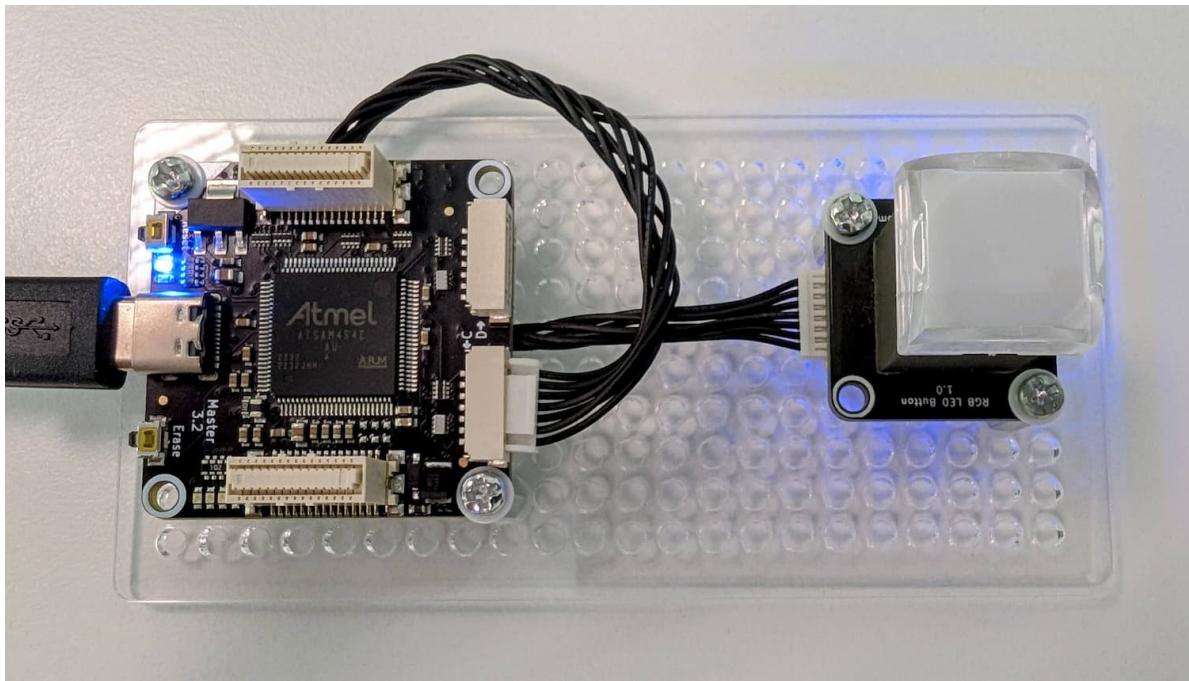
Der Button ist mehr als nur ein Knopf. Er hat zwei Tricks drauf: Er erkennt, ob er gedrückt wird, und er kann leuchten. In Farbe! Wie wir das testen? Das geht ganz leicht mit dem Brick Viewer.

Schließe zuerst den Master Brick über das USB-Kabel an deinen Computer an und öffne den Brick Viewer. Klicke dann auf den Connect-Button.

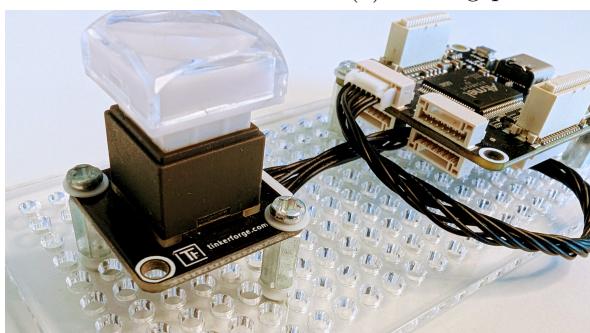
Wenn alles geklappt hat, zeigt dir der Brick Viewer alle angeschlossenen Geräte in Tabs an. Schau dir Abbildung [1.3](#) an – so etwa sollte es aussehen.

Wechsle nun zum Tab des RGB LED Buttons. Hier kannst du sowohl die Farbe der LED einstellen als auch den aktuellen Zustand des Buttons sehen. Im Moment steht da vermutlich „released“, was so viel bedeutet wie „nicht gedrückt“. Probier es direkt aus: Drück den Button! Schau, was passiert!

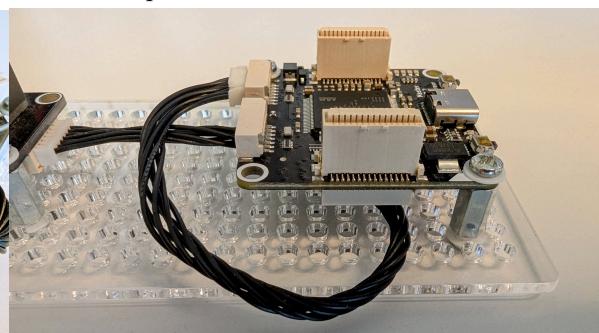
Mit den drei Schieberegbern steuerst du die Farbe – Rot, Grün, Blau. Wertebereich: 0 bis 255. Warum gerade diese Zahlen? Gute Frage. Die Antwort kommt weiter unten.



(a) Montageplatte mit allen Komponenten.



(b) Nahaufnahme des Buttons.



(c) Nahaufnahme des Master Bricks.

Abbildung 1.1: Einfaches Setup mit Mikrocontroller und Button mit integrierter LED.

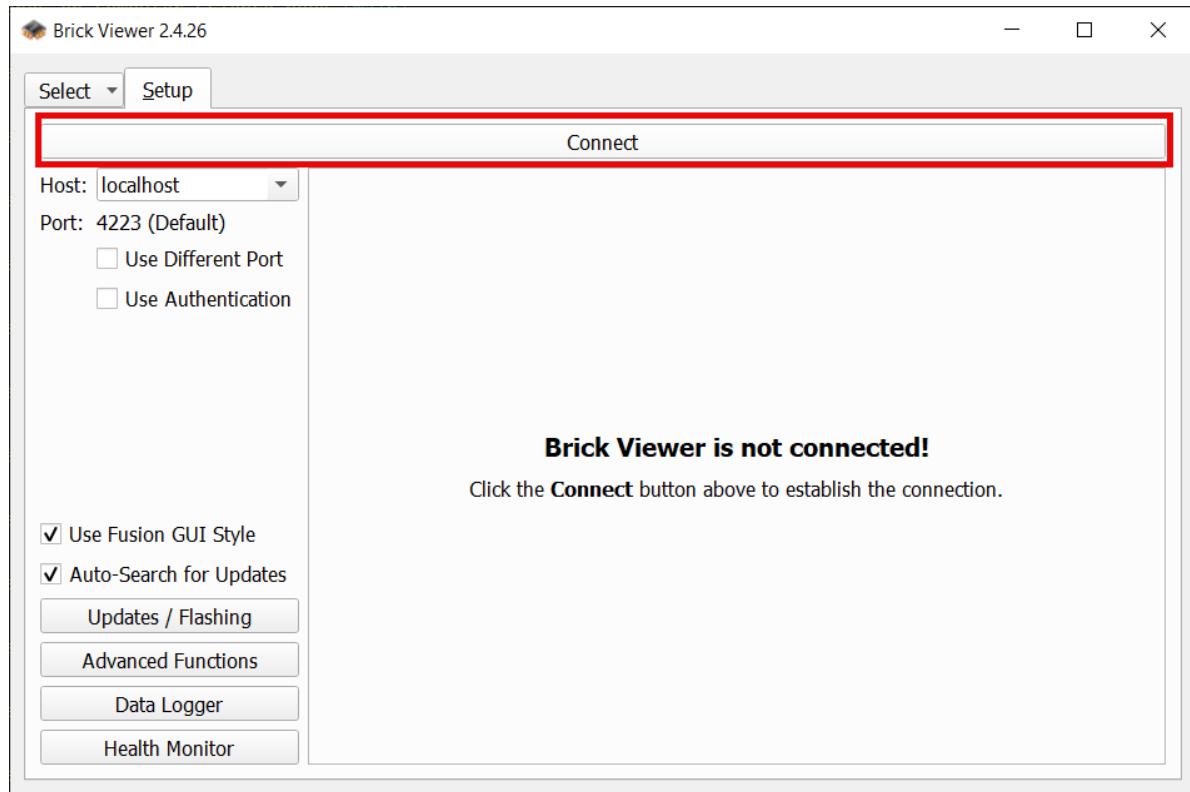


Abbildung 1.2: Über den Connect-Button verbindet sich der Brick Viewer mit dem angeschlossenen Master Brick.

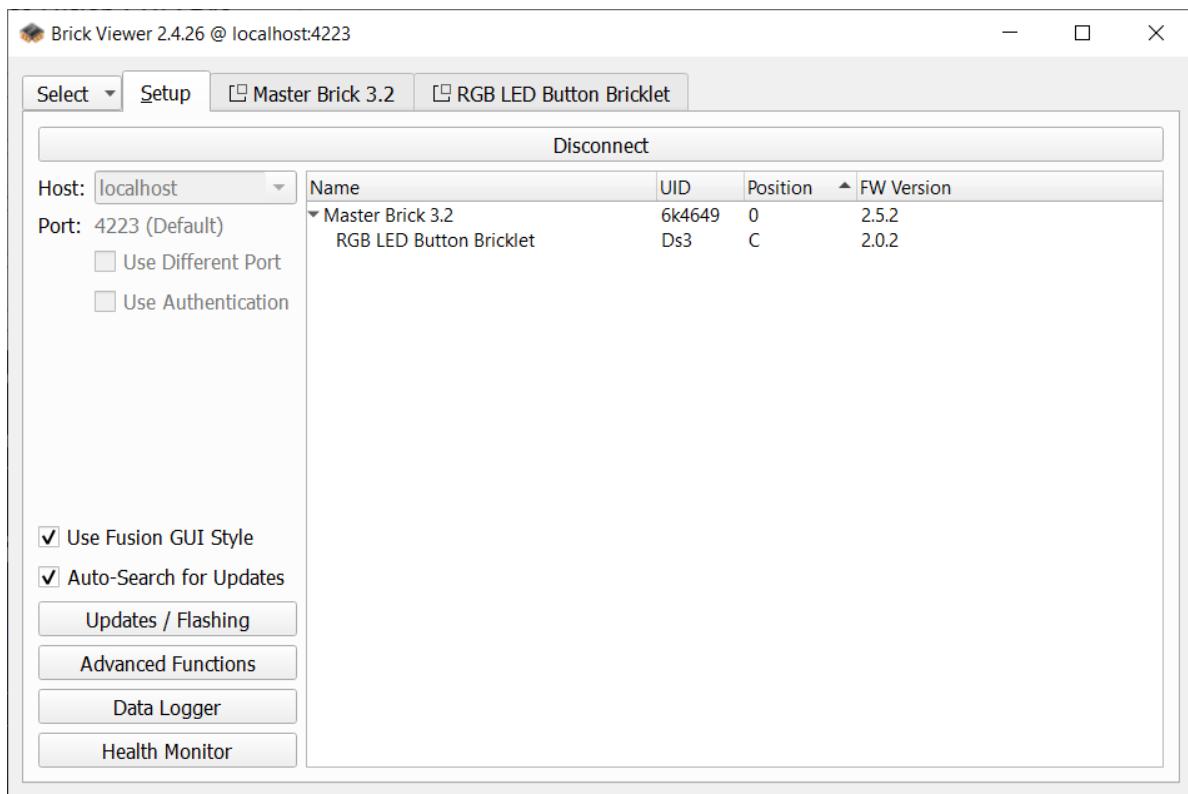


Abbildung 1.3: Der Brick Viewer nachdem ihr mit dem Master Brick verbunden seid.

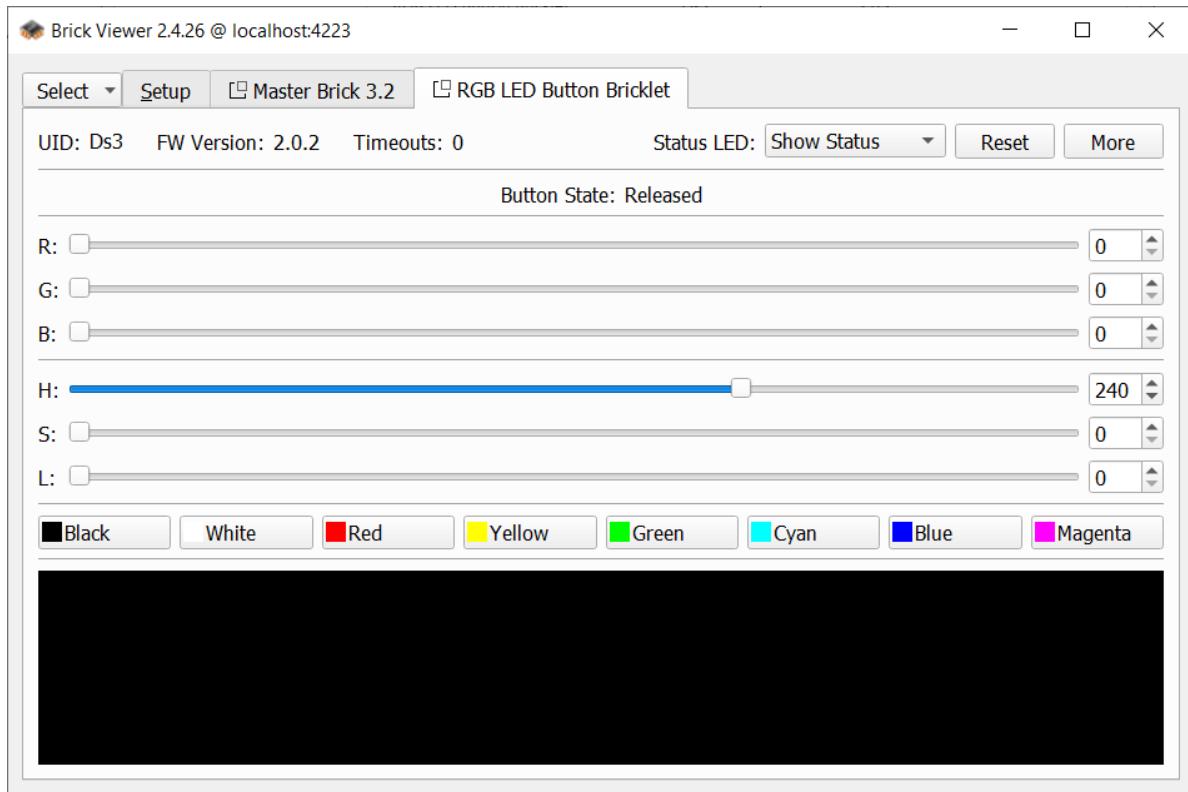


Abbildung 1.4: Die Ansicht für den RGB LED Button im Brick Viewer, in der alle Funktionen per Klick im Zugriff sind.

Fazit: Der Brick Viewer ist top zum Rumprobieren. Aber wenn du echte Projekte umsetzen willst, musst du programmieren können. Also los!

1.2 Unser erstes Programm

Wie verbindet man sich eigentlich über ein Programm mit dem Button? Die Antwort darauf findest du im folgenden kurzen Codebeispiel.

Listing 1.1 Der Boilerplate Code für die Verbindung mit den Geräten am Beispiel des Buttons.

```
from tinkerforge.ip_connection import IPConnection          ①
from tinkerforge.bricklet_rgb_led_button import BrickletRGBLEDButton    ②

ipcon = IPConnection()                                     ③
ipcon.connect("localhost", 4223)                            ④
btn = BrickletRGBLEDButton("Ds3", ipcon)                  ⑤
```

- ① Hier importieren wir ein Objekt aus einer Bibliothek zum Herstellen einer Verbindung mit dem Master Brick.
- ② Hier importieren wir ein weiteres Objekt, das wir zur Darstellung des Buttons als Python-Objekt benötigen.
- ③ Die Verbindung erfolgt über eine sogenannte IP-Verbindung, die wir hier als Objekt erstellen.
- ④ Mit `connect` stellen wir eine Verbindung zum angeschlossenen Master Brick her.
- ⑤ Schließlich holen wir uns eine virtuelle Instanz des RGB LED Button Bricklets, indem wir die UID nennen und sagen, welche Verbindung (`ipcon`) genutzt werden soll.

1.2.1 Programme

Zunächst einmal klären wir den Begriff *Programm*. Ein Programm ist eine Abfolge von Anweisungen, die ein Computer ausführt, um eine bestimmte Aufgabe zu erledigen. In unserem Fall ist das Programm später dafür zuständig, mit dem Button zu interagieren und die LED in verschiedenen Farben leuchten zu lassen. Programme werden in Programmiersprachen geschrieben, die es uns ermöglichen, dem Computer präzise Anweisungen zu geben. Wir verwenden in diesem Buch die Programmiersprache Python, die sich besonders gut für Einsteiger eignet und gleichzeitig mächtig genug ist, um komplexe Aufgaben zu lösen.

Wenn wir ein Programm ausführen, arbeitet der Computer die Anweisungen Schritt für Schritt von oben nach unten ab. Es gibt Befehle, die den Computer von dieser linearen Abfolge abweichen lassen, etwa Schleifen oder Verzweigungen. Diese lernen wir später kennen.

1.2.2 Boilerplate Code

Der Codeausschnitt in Listing 1.1 wird uns im Verlauf dieses Buches immer wieder begegnen. Wir benötigen ihn, um uns am Anfang des Programms mit den Geräten zu verbinden. In der Informatik nennen wir solchen Code, den wir häufig in der gleichen Form benötigen und fast eins zu eins kopieren können, auch *Boilerplate Code*. Wundert euch also nicht, wenn ich diesen Begriff ab und an mal verwende. Jetzt wisst ihr, was gemeint ist.

1.2.3 Bibliotheken

Beginnen wir in den ersten beiden Zeilen. Hier seht ihr zwei sehr ähnliche Befehle, die mit dem Schlüsselwort `from` beginnen. Nach dem Schlüsselwort `from` folgt der Name einer Bibliothek, aus der wir ein für unser Programm benötigtes Objekte importieren. Die Kombination der Schlüsselwörter `from ...import` lässt sich also wörtlich übersetzen: “Aus der Bibliothek X importiere das Objekt Y”.

Eine *Bibliothek* in einer Programmiersprache ist die Bündelung und Bereitstellung von Funktionen, Klassen oder Konstanten. Eine Bibliothek kannst du dir vorstellen wie einen Werkzeugkasten: Sie enthält fertige Werkzeuge (Funktionen und Klassen), damit du nicht alles von Grund auf selbst schreiben musst. Tinkerforge stellt uns genau solche Werkzeuge bereit, damit wir schnell und unkompliziert mit den Geräten loslegen können.

1.2.4 Klassen und Objekte

Mit `from ... import` importieren wir also etwas aus einer Bibliothek. Soweit so gut. Aber was bedeutet das genau? Mit *importieren* ist konkret gemeint, dass wir dem Programm mitteilen, dass wir vorhaben, die genannten Dinge in unserem Programm zu verwenden, und dass sie deshalb am besten schon einmal geladen werden sollten. Ob wir diese Dinge später wirklich nutzen steht auf einem anderen Blatt.

In dem Fall der ersten beiden Zeilen unseres Programms von oben sind es zwei *Klassen*, deren Verwendung wir ankündigen. Die erste Klasse heisst `IPConnection` und die zweite `BrickletRGBLEDButton`. Der Begriff *Klasse* ist hier verstehen analog zum Begriff *Kategorie*. Wir können zu einer Klasse gehörige *Objekte* erzeugen, und alle Objekte der selben Klasse verhalten sich gleich und haben die gleichen Funktionen. Das verstehen wir am besten an einem einfachen Beispiel.

Stellt euch vor, ihr habt eine Klasse namens `Auto`. Diese Klasse beschreibt alle Eigenschaften und Funktionen, die ein Auto haben kann, wie etwa `fahren()`, `bremsen()` oder `tanken()`. Diese Dinge sollen für jedes Auto gleich ablaufen. Jedes konkrete Auto in der Welt ist ein Objekt dieser Klasse. Du kannst also sagen: “Mein Auto ist ein Objekt der Klasse `Auto`.” Jedes `Auto` hat die gleichen Eigenschaften wie Farbe, Marke und Modell. Aber jedes `Auto` kann andere Werte für diese Eigenschaften haben.

Genauso verhält es sich mit den Klassen, die Tinkerforge für uns bereitgestellt hat. Die Klasse `IPConnection` beschreibt, wie wir eine Verbindung zu einem Mikrocontroller herstellen können, und die Klasse `BrickletRGBLEDButton` beschreibt, wie wir mit dem Button interagieren können. Wenn wir ein Objekt dieser Klasse erstellen, können wir die Funktionen nutzen, die in der Klasse definiert sind. Ein Button muss nicht fahren oder bremsen wie ein Auto. Dafür hat er andere Funktionen, wie etwa `get_button_state()` oder `set_color()`, die uns erlauben, den Status des Buttons abzufragen oder die Farbe der integrierten LED zu ändern. Eine Eigenschaft jedes Buttons ist seine UID, die eindeutig ist und uns hilft, ihn im System zu identifizieren.

1.2.5 Schlüsselwörter

Soeben haben wir mit `from` und `import` unsere ersten beiden Schlüsselwörter in Python kennengelernt! Aber was bedeutet das genau? Ein Schlüsselwort, das wir im Englischen auch *keyword* oder *reserved keyword* nennen, ist ein Begriff, der in der jeweiligen Programmiersprache eine feste Bedeutung hat und deshalb nicht anderweitig verwendet werden darf. Wir werden gleich noch sehen, dass wir bei der Programmierung auch häufig Namen vergeben müssen, etwa für Variablen oder Funktionen. Diese Namen dürfen nicht wie ein Schlüsselwort lauten, ansonsten funktioniert unser Programm nicht wie gewünscht. Welche Schlüsselwörter es in Python gibt, könnt ihr [hier](#) nachschauen.

Im Codeausschnitt oben laden wir zuerst das Objekt für die Verbindung zum angeschlossenen Mikrocontroller, die über eine IP-Verbindung hergestellt wird. Was das genau ist? Später mehr dazu. Zusätzlich zur `IPConnection` laden wir anschließend noch die benötigen Klassen für die Geräte, die wir in unserem aktuellen Setup verwenden wollen. In diesem Kapitel ist das nur der Button mit integrierter LED.

1.2.6 Objekte erzeugen

In Listing 1.1 in Zeile 4 erzeugen wir ein Objekt der Klasse `IPConnection`. Die fertige Instanz - so nennen wir ein Objekt, das aus einer Klasse erzeugt wurde - speichern wir auf einer *Variable* mit dem Namen `ipcon`. Diesen Namen haben wir uns selbst ausgedacht, damit wir später darauf zugreifen können. Eine Variable ist also ein Platzhalter für einen Wert, den wir später im Programm verwenden wollen. In diesem Fall ist `ipcon` der Platzhalter für die Verbindung zu unserem Mikrocontroller. Was eine Variable genau ist, lernen wir später noch genauer kennen.

1.2.7 Methoden

Über das Objekt `ipcon` können wir nun eine Verbindung zu unserem Mikrocontroller herstellen. Das geschieht in Zeile 5 mit der Methode `connect()`. Eine *Methode* ist eine Funktion, die zu

einem Objekt gehört. Wie etwa `fahren()` oder `bremsen()` in unserem Auto-Beispiel.

Wir können Methoden aufrufen, um eine bestimmte Aktion auszuführen. In diesem Fall stellen wir eine Verbindung zum Mikrocontroller her, indem wir die Adresse und den Port angeben, über den die Verbindung hergestellt werden soll. In unserem Fall ist das “localhost”, was für die lokale Maschine steht, und Port 4223, der durch den Brick Daemon standarmäßig so konfiguriert ist. Der Aufruf einer Methode erfolgt immer mit dem Punkt . nach dem Objekt, gefolgt vom Namen der Methode und den Klammern (), in denen wir eventuell benötigte Parameter angeben.

1.2.8 Ein Objekt für den Button

In Zeile 6 erzeugen wir schließlich ein Objekt der Klasse `BrickletRGBLEDButton`. Dieses Objekt repräsentiert unseren Button und ermöglicht es uns, mit ihm zu interagieren. Wir nennen das Objekt `btn`, was für Button steht. Auch hier haben wir uns den Namen selbst ausgedacht, um später darauf zugreifen zu können. Auch wenn wir grundsätzlich Variablennamen frei wählen können, sollten sie immer so gewählt werden, dass sie den Inhalt der Variable beschreiben. Das macht es später einfacher, den Code zu verstehen. Gleichzeitig gibt es in Python einige Regeln, die wir bei der Benennung von Variablen beachten müssen. Dazu gehören etwa, dass Variablennamen nicht mit einer Zahl beginnen dürfen und keine Leerzeichen enthalten dürfen. Eine ausführliche Liste der Regeln findest du [hier](#).

1.2.9 Zusammenfassung unseres ersten Programms

Damit haben wir unser erstes Programm von oben nach unten erläutert und dabei schon viele wichtige Konzepte der Programmierung kennengelernt:

Programme	Abfolge von Anweisungen, die nacheinander ausgeführt werden.
Boilerplate Code	Standard-Code, den man immer wieder braucht.
Importieren von Bibliotheken	Sammlung von fertigen Code-Elementen.
Schlüsselwörter	Reservierte Begriffe der Programmiersprache.
Klassen und Objekte	Kategorien und deren konkrete Instanzen.
Methoden und Funktionen	Funktionen, die zu einem Objekt gehören.
Variablen	Platzhalter für Werte.

1.2.10 Und jetzt?

Wir haben nun eine virtuelle, digitale Repräsentation unseres Buttons im Python. Damit können wir die LED des Buttons zum Leuchten bringen. Dazu verwenden wir eine Methode der Klasse `BrickletRGBLEDButton`, die `set_color()` heißt. Diese Methode erwartet drei Parameter: Rot, Grün und Blau. Mit diesen Parametern können wir die Farbe der LED einstellen.

```
btn.set_color(0, 255, 0)
```

(1)

- (1) Setzt die LED auf grün. R = 0, G = 255, B = 0. Logisch, oder?

Moment mal ... Wo steht hier eigentlich *grün*? Steht da gar nicht. Stattdessen: Zahlen. Willkommen bei der *RGB-Farbkodierung*. Jede Farbe besteht aus drei Werten zwischen 0 und 255: Rot, Grün, Blau. Null ist nix. 255 ist volle Power. Alles 0? Schwarz. Alles 255? Weiß. Nur Grün auf 255? Na klar: grün.

Aber warum machen wir das mit Zahlen? Weil Computer nun mal mit Zahlen arbeiten. Das ist einer der zentralen Gedanken dieses Buches: Wie übersetzen wir die Welt in etwas, das ein Computer versteht?

Warum aber ist das so? Warum kodieren wir in der Informatik jede Farbe mit *drei* Zahlen? Warum überhaupt mit Zahlen? Hier kommen wir zu einer zentralen Frage dieses Buches: Wie bilden Computer Informationen ab?

1.3 Farben und Licht

Pyhsik ist vielleicht schon eine Weile her und wir halten es hier auch kurz. Erinnern wir uns kurz, was Licht und damit Farben sind. Licht ist eine Form von *elektromagnetischer Strahlung*. Das bedeutet, dass es sich um Wellen handelt, die sich durch den Raum bewegen. Diese Wellen haben unterschiedliche Frequenzen und Wellenlängen. Das sichtbare Licht ist nur ein kleiner Teil des gesamten elektromagnetischen Spektrums, das von Radiowellen über Infrarotstrahlung bis hin zu Röntgenstrahlen und Gammastrahlen reicht.

Bei Wellen unterscheiden wir zwischen der Frequenz (wie oft die Welle pro Sekunde schwingt) und der Wellenlänge (der Abstand zwischen zwei aufeinanderfolgenden Wellenbergen). Die Frequenz und die Wellenlänge sind umgekehrt proportional: Je höher die Frequenz, desto kürzer die Wellenlänge und umgekehrt.

Frequenzen messen wir in Hertz (Hz), wobei 1 Hz einer Schwingung pro Sekunde entspricht. Das sichtbare Licht hat Frequenzen im Bereich von etwa 430 THz (Terahertz) bis 750 THz. Die Wellenlängen des sichtbaren Lichts liegen zwischen etwa 400 nm (Nanometer) für violettes Licht und etwa 700 nm für rotes Licht. Um sich das vorzustellen: Ein Nanometer ist ein

Milliardstel Meter. Zum Vergleich: Ein menschliches Haar hat einen Durchmesser von etwa 80.000 bis 100.000 Nanometern. Die Abstände zwischen den Wellenlängen des sichtbaren Lichts sind also extrem klein.

Was bedeutet das nun für eine LED? Eine LED (Light Emitting Diode) ist ein Halbleiterbauelement, das Licht erzeugt, wenn elektrischer Strom hindurchfließt. Die Farbe des Lichts hängt von Eigenschaften des Halbleitermaterials ab, aus dem die LED besteht. Verschiedene Materialien emittieren Licht bei unterschiedlichen Wellenlängen, was zu verschiedenen Farben führt. Zum Beispiel emittiert eine rote LED Licht mit einer Wellenlänge von etwa 620-750 nm, während eine grüne LED Licht mit einer Wellenlänge von etwa 495-570 nm emittiert.

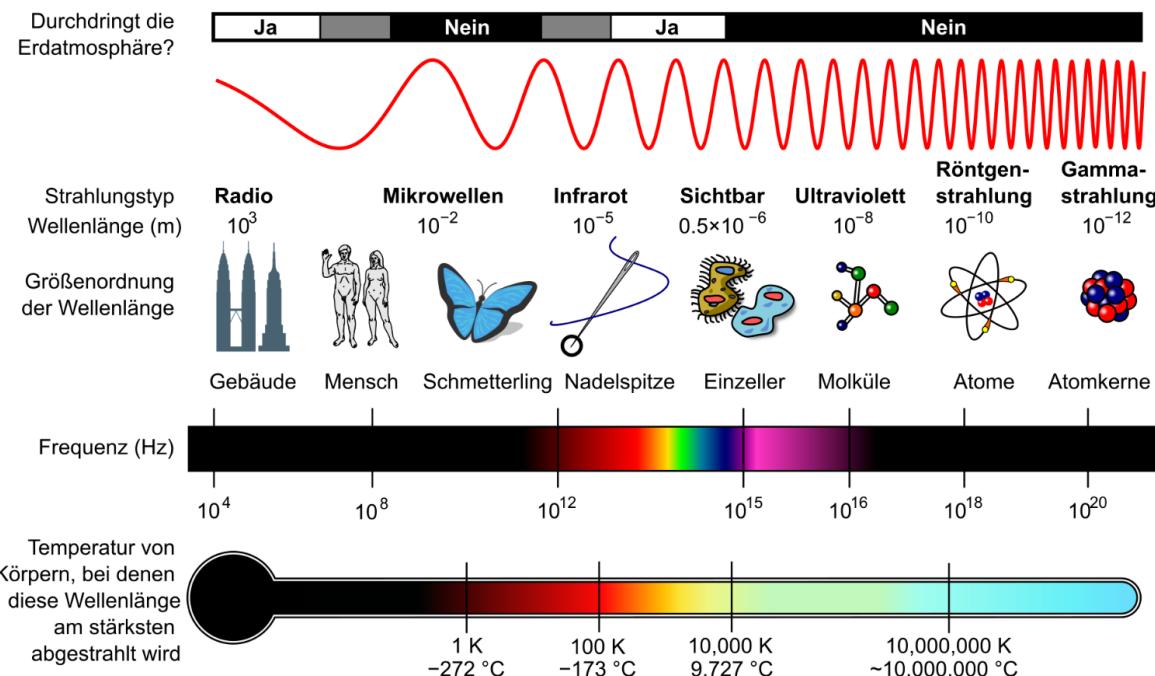


Abbildung 1.5: Das elektromagnetische Spektrum, wovon das sichtbare Licht ein Teil ist.
(Quelle: [Wikipedia](#))

Die RGB LEDs, die in unserem Button verbaut ist, besteht in Wirklichkeit aus drei einzelnen LEDs: einer roten, einer grünen und einer blauen. Jede dieser LEDs kann unabhängig voneinander angesteuert werden, um verschiedene Farben zu erzeugen. Mehr Spannung bedeutet mehr Intensität der jeweiligen Farbe. Durch die Kombination der drei Grundfarben Rot, Grün und Blau in unterschiedlichen Intensitäten können wir eine Vielzahl von Farben mischen. Das ist das Prinzip der additiven Farbmischung: Wenn wir alle drei Farben mit voller Intensität leuchten lassen, erhalten wir Weiß. Wenn wir keine Farbe leuchten lassen, erhalten wir Schwarz. Klar, die LED sind dann alle aus.

1.4 Farben im Computer

Nun wissen wir, warum die Methode `set_color()` drei Parameter erwartet: Rot, Grün und Blau. Diese Parameter sind die Intensitäten der jeweiligen Farbe, die wir in unserem Programm angeben. Mit den Werten 0 bis 255 können wir jede Farbe im sichtbaren Spektrum erzeugen.

Wie kommt es aber zu der merkwürdigen Zahl 255? Warum nicht einfach 0 bis 100? Das liegt daran, wie ein Computer grundsätzlich Werte speichert und wie dieser Speicher organisiert ist. Kurz gesagt: Der Wertebereich 0 bis 255 passt genau in ein sogenanntes *Byte*. Ein Byte ist eine Speichereinheit, die aus 8 Bits besteht. Ein Bit kann entweder 0 oder 1 sein. Mit 8 Bits können wir also $2^8 = 256$ verschiedene Werte darstellen, von 0 bis 255. Das ist genau der Bereich, den wir für die RGB-Farbkodeierung verwenden. Dazu lernen wir später noch mehr.

Wir halten also fest, dass sich ein Farbwert im Computer aus drei Zahlen zusammensetzt, die jeweils zwischen 0 und 255 liegen. Das gilt für unsere LED, aber beispielsweise auch für Pixel in Bildern oder auf eurem Bildschirm. Das beleuchten wir später auch noch genauer.

1.5 Farben mischen

Klingt alles theoretisch ja sehr gut. Aber wie sieht es mit der Praxis aus? Probieren wir es aus!

```
btn.set_color(255, 255, 0)
```

Was macht der Befehl? Welche Farbe kommt dabei heraus? Probiert es einfach mal aus!

1.5.1 Additive Farbmischung

Ihr solltet alle euren Button in Gelb aufleuchten sehen. In der *additiven Farbmischung* mischen wir Rot und Grün und erhalten dadurch Gelb. Gelb ist heller als die beiden Farben Rot und Grün, was kein Zufall ist. Das ist das Prinzip der additiven Farbmischung: Wenn wir zwei Farben mit voller Intensität leuchten lassen, erhalten wir eine neue Farbe, die stets heller ist als die Ursprungsfarben. Wir fügen mehr Licht hinzu. Wenn wir alle drei Farben mit voller Intensität mischen erhalten wir schließlich Weiß.

```
btn.set_color(255, 255, 255)
```

1.5.2 Subtraktive Farbmischung

Die *subtraktive Farbmischung* funktioniert anders, nämlich genau umgekehrt. Statt beim Mischen Licht hinzuzufügen, nehmen wir Licht weg.

Erinnert ihr euch an euren Farbkasten aus der Grundschule? Dort habt ihr auch Farben gemischt, um neue Farben zu erzeugen, die euer Farbkasten nicht direkt mitgeliefert hat. Was hat im Farbenkasten die Mischung aus Rot und Grün ergeben? Sicher nicht gelb - eher braun. Eind unklere Farbe. Das liegt daran, dass wir hier nicht von additiver, sondern von subtraktiver Farbmischung sprechen. Bei der subtraktiven Farbmischung mischen wir Pigmente, die Licht *absorbieren* und *reflektieren*. Das Mischen von Farben fungiert hier wie ein Filter: Bestimmte Teile des Lichtspektrums werden nicht mehr reflektiert, sondern absorbiert und sind damit nicht mehr sichtbar.

Was passiert mit dem absorbierten Licht? Es wird in eine andere Form der Energie umgewandelt, nämlich Wärme. Deshalb wird eine schwarze Oberfläche auch besonders heiß, wenn die Sonne darauf knallt. Sie absorbiert das gesamte Lichtspektrum und wandelt es in Wärme um. Dagegen wirken weiße Oberflächen fast wie Klimaanlagen. Es ist kein Zuall, dass wir in sonnigen Erdteilen viele weiße Fassaden sehen.

Wenn wir alle Farben mischen ergibt die subtraktive Farbmischung Schwarz, weil kein Licht mehr reflektiert wird. Alles Licht wird aufgesogen und nichts kommt mehr zurück. Das ist ein anderes Prinzip als bei der additiven Farbmischung, bei der wir Lichtquellen *kombinieren*, um neue Farben zu erhalten.

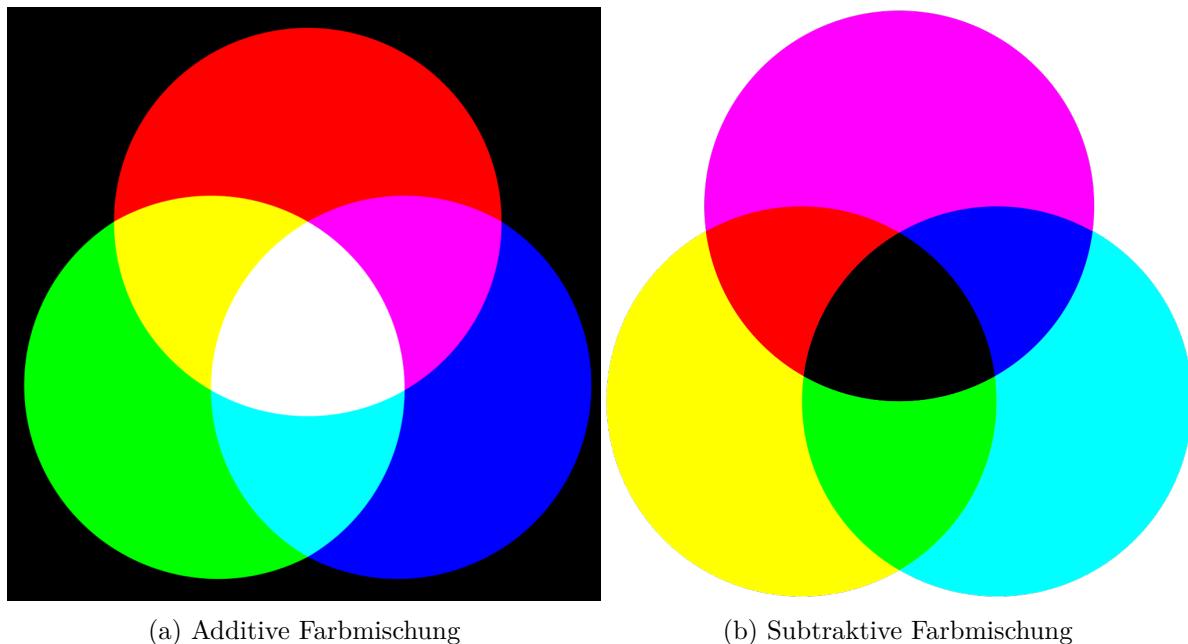


Abbildung 1.6: Additive und subtraktive Farbmischung.

Die Grundfarben, die wir bei der subtraktiven Variante benötigen, um daraus alle weiteren Farben zu erhalten sind Cyan, Magenta sowie Gelb. Im Englischen ist die Abkürzung CMY, wo das "Y" für *Yellow* steht. In der additiven Farbmischung sind es Rot, Grün und Blau. Wenn ihr Abbildung 1.6b betrachtet dann erkennt ihr, dass genau diese drei Farben durch das Mischen jeweils zweier Grundfarben in der subtraktiven Farbmischung entstehen. Und umgekehrt gilt das gleiche Prinzip! Ob das Zufall ist?

Das war natürlich eine rhetorische Frage. Wenn wir in der additiven Farbmischung Grün und Rot mischen, um Gelb zu erhalten, muss es im Umkehrschluss bedeuten, dass

Damit können wir auch erklären, warum Farbdrucker vier unterschiedliche Kartuschen benötigen (Abbildung 1.7). Mit den Grundfarben der subtraktiven Farbmischung Cyan, Magenta und Gelb können wir jede beliebige Farbe mischen. Zusätzlich haben Drucker eine Kartusche für Schwarz, um erstens ein sattes Schwarz drucken zu können und zweitens die Farbkartuschen zu schonen. Denn schließlich müssen alle drei Farben gemischt werden, um Schwarz zu erhalten. Und weil viele Drucksachen schwarz enthalten (oder sogar ausschließlich) ist eine schwarze Kartusche einfach effizienter.

Farben spielen eine so wichtige Rolle für uns und die Arbeit mit Computern. Deshalb lohnt es sich, ein wenig über die Hintergründe von Farben und deren Mischung zu verstehen. Wir werden später noch lernen, wie Bildschirme Farben darstellen. Spätestens dann wird uns das Thema der Farbmischung wieder begegnen.

1.6 Der RGB-Code

Wir wollen jetzt weiter mit unserer LED experimentieren und den RGB-Code, mit dem Computer Farben abilden, besser verstehen. Bisher haben wir gelernt, dass wir die Farbe der LED über die Methode `set_color()` verändern können, wenn wir wissen, welcher RGB-Code die Farbe repräsentiert, die wir uns wünschen. Da wir jetzt mehr über die Farbmischung wissen, können wir die LED also ganz einfach in der Farbe Magenta leuchten lassen:

```
btn.set_color(255, 0, 255)
```

Gemäß der Theorie der additiven Farbmischung müsste Rot und Blau Magenta ergeben. Probiert es aus!

1.7 Schleifen

Das Ziel unseres ersten Experiments ist es, einen Regenbogenfarbverlauf zu erzeugen. Dazu müssen wir die Farbe der LED kontinuierlich ändern, sodass sie von Rot über Gelb, Grün, Cyan, Blau und Violett wieder zurück zu Rot wechselt. Das erreichen wir mit einer



Abbildung 1.7: Ein typisches Set mit CMY-Druckerkartuschen inklusive Schwarz.

Schleife. Eine Schleife ist ein Konstrukt in der Programmierung, das es uns ermöglicht, einen bestimmten Codeabschnitt mehrfach auszuführen, ohne ihn jedes Mal neu schreiben zu müssen. Das spart Zeit und macht den Code übersichtlicher.

1.7.1 Abzählbare Wiederholungen

Wir beginnen wie immer einfach. Wir nähern uns dem Regenbogen schrittweise an. Zunächst wäre es cool, wenn wir die LED einfach Rot pulsieren lassen könnten. Dazu müssen wir nämlich nur den Rot-Kanal und nicht alle drei Kanäle der LED ansteuern.

Aber was bedeutet es, die LED pulsieren zu lassen? Und was müssen wir dafür tun? Pulsieren bedeutet, dass die LED langsam immer heller wird, kurz in der vollen Helligkeit verweilt, aber dann sofort wieder kontinuierlich dunkler wird. Sobald sie schwarz ist fängt der Zyklus von Vorne an.

Den Ausdruck *immer heller werden* können wir bezogen auf die LED so übersetzen, dass wir den Anteil des Rot-Kanals schrittweise erhöhen. Wenn die LED zu Beginn aus ist, also alle Kanäle auf 0 stehen, können wir den Rot-Kanal von 0 auf 255 erhöhen und so die LED immer heller in Rot aufleuchten lassen.

Wir beginnen also mit einer schwarzen LED:

```
btn.set_color(0, 0, 0)
```

Anschließend setzen wir den Wert für Rot auf 1:

```
btn.set_color(1, 0, 0)
```

Und erhöhen ihn schrittweise:

```
btn.set_color(2, 0, 0)
btn.set_color(3, 0, 0)
btn.set_color(4, 0, 0)
# ...
```

Wenn wir nach diesem Muster fortfahren, hätten wir bis zum vollen Rot 255 Zeilen Code geschrieben, eine für Zeile jeden ErhöhungsSchritt. Und anschließend das gleiche nochmal rückwärts, damit wir wieder zu Schwarz kommen. Mit 510 Zeilen Code hätten wir dann eine Pulsierungszykls durchlaufen. Wollen wir die LED öfters pulsieren lassen vervielfacht sich unser Code entsprechend. Das kann nicht die Lösung sein.

Und tatsächlich gibt es in der Programmierung eine einfache Möglichkeit, um sich wiederholende Abläufe zu vereinfachen: die Schleife. In einem Fall, bei dem wir genau wissen, wie oft wir etwas wiederholen wollen, bietet sich eine Zählerschleife an:

```
for r in range(256):
    btn.set_color(r, 0, 0)
```

Voilá! Unsere 510 Zeilen Code können wir mit einer Schleife auf zwei Zeilen reduzieren! Dazu müssen wir im Kopf der Schleife `for ... in ... :` festlegen, wie oft der eingerückte Codeblock nach dem Doppelpunkt ausgeführt werden soll. In Python funktioniert das über die Angabe einer Menge, für die jedes Element einmal durchlaufen wird. Das aktuelle Element ist in der Schleife als `r` verfügbar. Und `r` nimmt nacheinander jeden Wert der Menge an, die nach dem Schlüsselwort `in` folgt. Diese Menge erzeugt die Funktion `range(256)`, die, wie der Name preisgibt, eine *Zahlenspanne* von 0 bis zum angegebenen Wert minus eins erzeugt. In unserem Fall also von 0 bis 255.

Um das besser nachvollziehen zu können, geben wir den Wert für `r` einfach mal aus:

```
for r in range(256):
    btn.set_color(r, 0, 0)
    print(r)
```

Jetzt wird es deutlich - mit jedem Durchlauf der Schleife wird ein neuer Wert für `r` gesetzt und ausgegeben. Und zwar jeweils um eins erhöht.

Rückwärts erreichen wir das gleiche Ergebnis mit einer weiteren Schleife, deren Menge wir umkehren, so dass sie von 255 bis 0 geht:

```
for r in range(255, -1, -1):
    btn.set_color(r, 0, 0)
```

Warum hat `range()` auf einmal drei Argumente? Ganz einfach: Standardmäßig erstellt die Funktion eine Menge von 0 bis zur angegebenen Zahl minus eins. Wir können die Menge aber beeinflussen, indem wir einen Startwert und einen Schrittwert angeben. In unserem Fall oben beginnen wir bei 255 (erster Parameter) und gehen bis -1 (zweiter Parameter), wobei wir in jedem Schritt um -1 verringern (dritter Parameter). Warum zählen wir bis -1, wo wir doch eigentlich die 0 als kleinste Zahl benötigen? Das liegt daran, dass die Menge von `range()` immer bis zum zweiten Parameter minus eins geht. Wenn wir also 0 als kleinste Zahl benötigen, müssen wir bis -1 zählen.

1.7.2 Bedingte Wiederholung

- `if .. elif .. else`

1.8 Das fertige Programm

Listing 1.2 Das fertige Programm, das die LED rot pulsieren lässt.

```
import time
from tinkerforge.ip_connection import IPConnection
from tinkerforge.bricklet_rgb_led_v2 import BrickletRGBLEDV2

ipcon = IPConnection()
ipcon.connect('localhost', 4223)
led = BrickletRGBLEDV2('ZEP', ipcon)

# Turn LED off initially
led.set_rgb_value(0, 0, 0)

while True:

    # Slowly light up
    for r in range(255):
        led.set_rgb_value(r, 0, 0)
        time.sleep(0.001)

    # Keep full brightness for a quarter of a second
    time.sleep(0.25)

    # Slowly darken
    for r in range(255, 0, -1):
        led.set_rgb_value(r, 0, 0)
        time.sleep(0.001)

    # Keep dark for a quarter second
    time.sleep(0.25)
```

Listing 1.3 Das fertige Programm, das die LED in einem Regenbogenfarbverlauf leuchten lässt.

```
import keyboard
import time
import colorsys
from tinkerforge.ip_connection import IPConnection
from tinkerforge.bricklet_rgb_led_v2 import BrickletRGBLEDV2

ipcon = IPConnection()
ipcon.connect('localhost', 4223)
led = BrickletRGBLEDV2('ZEP', ipcon)

# Turn button of initially
led.set_rgb_value(0, 0, 0)

# Initialize the hue value
hue = 1

# Set the duration for one rainbow cycle
duration_seconds = 4
duration_seconds_per_step = duration_seconds / 360

# Loop until user presses escape key
while not keyboard.is_pressed('esc'):

    # Calculate RGB from hue
    r, g, b = colorsys.hsv_to_rgb(hue / 360.0, 1.0, 1.0)
    r = int(r * 255)
    g = int(g * 255)
    b = int(b * 255)

    led.set_rgb_value(r, g, b)

    # Wait a bit
    time.sleep(duration_seconds_per_step)

    hue = hue + 1
    if hue > 360:
        hue = 1

led.set_rgb_value(0, 0, 0)
```

2 Eingaben

Im zweiten Kapitel knüpfen wir an den Regenbogen an und lernen, wie wir den Computer über unsere Eingabe steuern können.

Setup

Wir erweitern unser Setup aus Kapitel 1 und fügen ein Eingabegerät hinzu: Einen einfachen Drehknopf ([Rotary Encoder Bricklet 2.0](#)). Den schraubt ihr neben die LED, wie auf der Abbildung zu sehen ist.

💡 Experiment

- RGB LED Button als Eingabegerät
- Einführung in andere Eingabegeräte

2.1 Das fertige Programm

```
import keyboard
import time
import colorsys
from tinkerforge.ip_connection import IPConnection
from tinkerforge.bricklet_rgb_led_v2 import BrickletRGBLEDV2

ipcon = IPConnection()
ipcon.connect('localhost', 4223)
led = BrickletRGBLEDV2('ZEP', ipcon)

# Turn button off initially
led.set_rgb_value(0, 0, 0)
```

```
# Initialize the hue value
hue = 1

# Set the duration for one rainbow cycle
duration_seconds = 4
duration_seconds_per_step = duration_seconds / 360

# Loop until user presses escape key
while not keyboard.is_pressed('esc'):

    # Calculate RGB from hue
    r, g, b = colorsys.hsv_to_rgb(hue / 360.0, 1.0, 1.0)
    r = int(r * 255)
    g = int(g * 255)
    b = int(b * 255)

    led.set_rgb_value(r, g, b)

    # Wait a bit
    time.sleep(duration_seconds_per_step)

    hue = hue + 1
    if hue > 360:
        hue = 1

led.set_rgb_value(0, 0, 0)
```

3 Texte

Wir nutzen Computer ständig für Texte – sei es für eine WhatsApp-Nachricht, eine E-Mail, die Einladung zur Hochzeitsfeier oder vielleicht sogar für deine Bachelorarbeit. Ständig tippen wir etwas in unser Smartphone, Tablet oder den Computer ein. Aber hast du dich schon einmal gefragt, wie das eigentlich genau funktioniert? Um das zu verstehen, wollen wir einen kleinen Umweg gehen.

Setup

Für dieses Kapitel benötigen wir die LED ([RGB LED Bricklet 2.0](#)) und den Infrarot-Entfernungsmeßer [Distance IR 4-30cm Bricklet 2.0](#). Beide Geräte schließen wir an den Mikrocontroller ([Master Brick 3.2](#)) und fixieren alle drei auf einer Montageplatte. Wie in der Abbildung gezeigt, soll der Entfernungsmeßer dabei nach oben zeigen.

i Kommt bald

Dieses Kapitel ist in Arbeit und wird in Kürze fertiggestellt.

3.1 Texte – ganz ohne Tastatur?

Ja, ich gebe zu: Eine Tastatur ist schon richtig praktisch. Schnell tippen wir damit Buchstaben, Zahlen, Satz- und Sonderzeichen (wie neue Zeilen und Leerzeichen) ein. Aber stell dir mal vor: Wie könnte ich eigentlich dieses Buch schreiben, wenn ich keine Tastatur hätte?

Tatsächlich gibt es spannende Alternativen! Hast du schon mal versucht, Texte zu diktieren? Google Docs oder Microsoft Word bieten mittlerweile ziemlich gute Möglichkeiten dafür. Anfangs fühlt sich das ungewohnt an, aber ich kenne Menschen, die darauf schwören und damit sogar produktiver sind.

Heute wollen wir aber bewusst einmal einen Schritt zurückgehen und das Tippen absichtlich erschweren. Klingt komisch? Macht aber Sinn! Denn wenn wir das Eingeben von Texten

schwieriger gestalten, lernen wir einerseits, unsere Tastaturen wieder richtig zu schätzen. Andererseits verstehen wir dann besser, wie Texte im Hintergrund verarbeitet und gespeichert werden.

Wie wäre es also mit einer Tastatur, die auf Handgesten basiert? Klingt verrückt? Perfekt – genau das machen wir!

3.2 Klein anfangen: Ein einfacher Button mit Handgesten

Starten wir ganz simpel. Wir bauen zuerst eine Art “Knopf” oder Button, der auf Handgesten reagiert. Dafür verwenden wir einen der Infrarot-Abstandssensoren – oder wie er bei Tinkerforge heißt: *Distance IR 4-30cm Bricklet 2.0*. Wie der Name schon sagt, misst der Sensor Entferungen von 4 bis maximal 30 cm.

Um den Sensor in unserem Programm zu nutzen, können wir ganz entspannt den Grundcode (Boilerplate-Code) aus den vorherigen Kapiteln übernehmen. Nur zwei kleine Anpassungen brauchen wir noch speziell für unseren Sensor:

```
from tinkerforge.ip_connection import IPConnection  
from tinkerforge.bricklet_distance_ir_v2 import BrickletDistanceIRV2      ①  
  
ip_con = IPConnection()  
ip_con.connect("localhost", 4223)  
  
ir_sensor = BrickletDistanceIRV2("", ip_con)                                ②
```

- ① Hier importieren wir die Klasse für den IR-Sensor.
- ② Hier erzeugen wir eine konkrete Instanz und verbinden uns mit dem angeschlossenen Sensor.
Vergiss nicht, deine UID hier einzutragen!

Was kann der Sensor? Eigentlich hat er nur eine Funktion:

```
distance = ir_sensor.get_distance()                                         ①  
print(f"Objekt ist {distance/100} cm entfernt")                           ②
```

- ① Die Funktion `get_distance()` fragt den aktuell gemessenen Wert ab.
- ② Der Sensor liefert den Wert in Zentimeter * 100 zurück.

Aber wie lässt sich mit den Entfernungswert ein Button umsetzen? Dazu vergegenwärtigen wir uns, was ein herkömmlicher Button eigentlich ist und wie er funktioniert. Ein Button, wie wir ihn auf einer **Tastatur** finden, kennt zwei Zustände: Gedrückt und nicht gedrückt. An oder aus. Wenn wir eine Taste drücken, so schließen wir darüber einen Stromkreis, der in einem winzigen Mikroprozessor mündet. Dieser Prozessor erkennt durch laufendes Scannen

sämtlicher Verbindungen zu den Tasten (die als Matrix angeordnet sind), durch welche Kreise Strom fließt und kann so auf die gedrückte Taste schließen. Der Mikroprozessor ermittelt dann anhand der Koordinate in der Matrix einen so genannten **Scan Code**, der für jede Taste anders lautet. Dieser Code wird als binäre Folge von Nullen und Einsen über das USB-Kabel an den Computer geschickt und dort von dem aktiven Programm - etwa einer Textverarbeitungssoftware - in ein Codesystem umgewandelt, das Zeichen abbilden kann. So wie etwa das ASCII-Codesystem, das wir weiter unten kennenlernen.

4 Bilder

 Kommt bald

Dieses Kapitel ist in Arbeit und wird in Kürze fertiggestellt.

 Experiment

- Einführung in die Idee einer Pixelmatrix aus Farbwerten als Bild
- Verwendung des LCD-Displays zur Veranschaulichung, auch wenn nur schwarz/weiß
- Verwendung von Zeichen auf dem LCD, um Pixel-Bitmaps für Schriftarten hervorzuheben
- Studierende nutzen Bitmap-Sheet, um ein eigenes Logo zu entwerfen und auf dem LCD anzuzeigen

5 Codes

💡 Experiment

Morse-Code über Piezo Speaker

- Einführung des Piezo Speaker
- Codesysteme

6 Umwandlung

Setup

💡 Experiment

Mit 4 Kippschaltern und 4 Widerständen bauen wir einen Digital-To-Analog-Converter (DAC). Dazu kommt ein Breadboard und dieses Überbrückungskabel zum Einsatz. Außerdem brauchen wir den Analog In 3.0 von Tinkerforge.

6.1 Der Weg in den Computer hinein

6.2 Der Weg aus dem Computer heraus

7 Information

Setup

i Kommt bald

Dieses Kapitel ist in Arbeit und wird in Kürze fertiggestellt.

8 Sensoren

Setup

i Kommt bald

Dieses Kapitel ist in Arbeit und wird in Kürze fertiggestellt.

- Analog vs. digital
- Baue einen Wasserstandssensor mit einem Widerstand und dem Analog-In-Bricklet.
- Oder: Feuchtigkeitssensor in eine Pflanze stecken
- Oder: Berührungssensor
- Oder: Pulssensor (Farbsensor)
- Integriere die RGB-LED irgendwie
- Farbsensor

Aufgaben

- Programmiere einen Batteriedoktor mithilfe des Analog In Sensors

9 Signale

Setup

i Kommt bald

Dieses Kapitel ist in Arbeit und wird in Kürze fertiggestellt.

💡 Experiment

Mithilfe des Farbsensors bauen wir einen Pulsmesser.

Hast du dich schon einmal gefragt, wie dein Fitness-Tracker deinen Puls messen kann, obwohl du nur einen Finger auflegst? Diese Frage führt uns direkt zu einem spannenden Konzept: **Signalen**. Alles, was wir mit Sensoren messen, sind zunächst einmal beliebige Größen. Doch nicht alles, was wir messen, ist für uns relevant. Wir suchen nach Mustern in diesen Daten — eben genau diesen Mustern, die wir als Signale bezeichnen. Alles andere, was uns von diesen Signalen ablenkt, nennen wir Rauschen. Unser Ziel: Wir möchten herausfinden, wie wir Signale effektiv vom Rauschen unterscheiden können.

9.1 Pulsmesser: Dein Finger als Signalquelle

Erinnerst du dich noch an den Farbsensor aus Kapitel 8? Er misst nicht nur die Intensität des RGB-Spektrums, sondern auch die allgemeine Lichtintensität, auch *Illuminance* genannt. Hier kommt die spannende Tatsache ins Spiel: Genau dieses Prinzip steckt hinter den Pulsmessern in Fitnesstrackern. Ja, genau der Sensor, der Licht misst, verrät dir, wie schnell dein Herz schlägt!

Aber wie genau funktioniert das? Stell dir vor, du legst deinen Zeigefinger direkt auf den Sensor und schaltest die integrierte weiße LED an. Das Licht der LED trifft auf deinen Finger und wird reflektiert. Dein Finger sieht für dich immer gleich aus, aber tatsächlich sorgt dein Herzschlag dafür, dass dein Finger mal minimal heller und mal dunkler erscheint. Das liegt daran, dass Blut in rhythmischen Schüben durch die Gefäße gepumpt wird. Diese winzigen Veränderungen, die du mit bloßem Auge nicht sehen kannst, werden vom sensiblen Farbsensor deutlich wahrgenommen.

Schauen wir uns das einmal genauer an: Wenn du dir die gemessene Lichtintensität über den Zeitverlauf im Brick Viewer ansiehst, kannst du deinen Pulsschlag tatsächlich erkennen—er wird sichtbar als kleine, regelmäßige Ausschläge oder *Peaks*. Faszinierend, oder? So kannst du beobachten, wie aus etwas so scheinbar Einfachem wie Licht ein Signal entsteht, das dir Informationen über deinen Körper liefert.

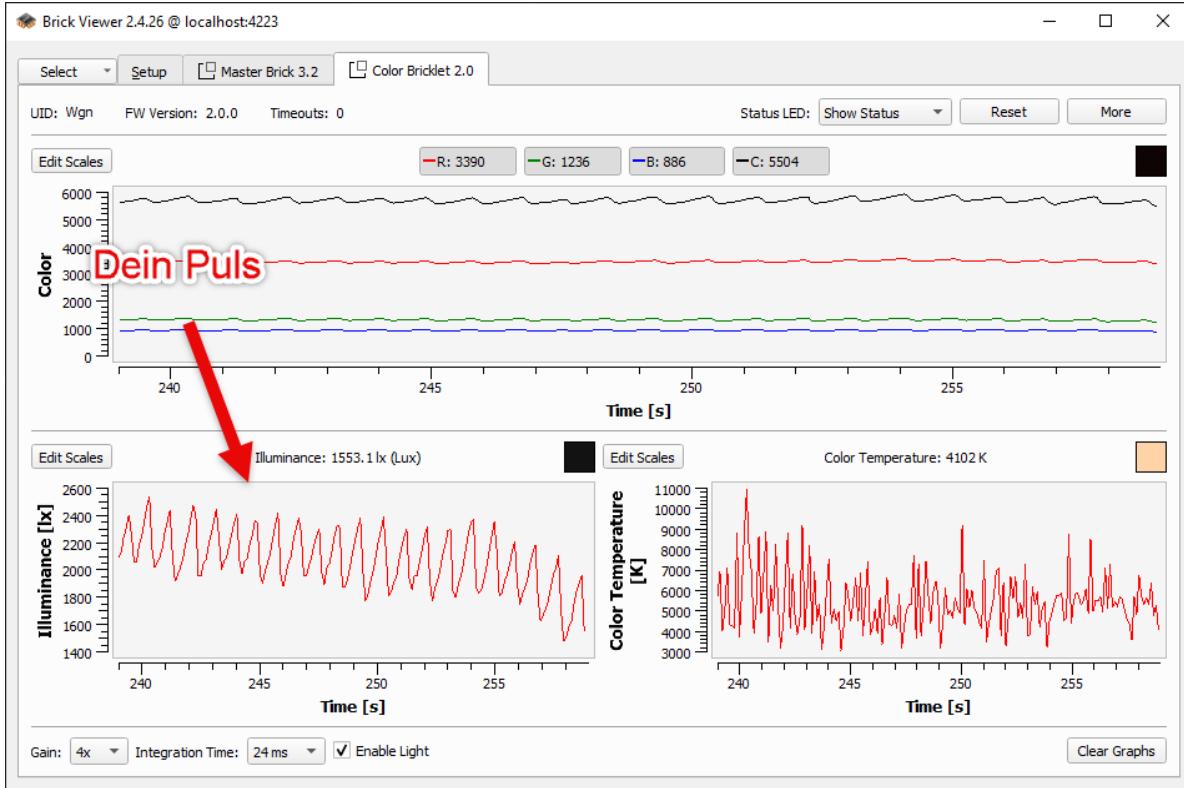


Abbildung 9.1: Dein Puls im Brick Viewer

9.2 Vom Diagramm zur Zahl

Auch wenn du das Signal im Liniendiagramm in Abbildung 9.1 bereits deutlich sehen kannst, bleibt eine spannende Herausforderung bestehen: Wie schreibst du ein Programm, das aus diesen Daten deinen Puls als konkrete Zahl, zum Beispiel “60 Schläge pro Minute”, berechnet? Genau dafür sind wir ja hier – um herauszufinden, wie man solche kniffligen Aufgaben löst. Lass uns gemeinsam starten!

Zunächst müssen wir den Farbsensor in unserem Python-Programm auslesen. Den notwendigen Code dafür haben wir im vorherigen Kapitel 8 bereits kennengelernt.

```

from tinkerforge.ip_connection import IPConnection
from tinkerforge.bricklet_color_v2 import BrickletColorV2

ip_con = IPConnection()
ip_con.connect("localhost", 4223)

color_sensor = BrickletColorV2("Wgn", ip_con)

```

①

- ① Achtung: Vergiss nicht, hier deine eigene UID einzusetzen. Diese findest du im Brick Viewer.

Jetzt haben wir Zugriff auf die Funktionen des Sensors und können etwa die Lichtintensität messen:

```

color_sensor.set_light(True)                                ①
color_sensor.set_configuration(1, 1)                      ②
illuminance = color_sensor.get_illuminance()

```

①

②

- ① Schaltet die integrierte LED des Sensors an.
 ② Stellt die Werte für *Gain* und *Integration Time* auf 4x und 24ms. Diese Einstellung scheinen gut zu funktionieren, um den Puls zu messen.

Der zurückgegebene Wert hängt von zwei Einstellungen ab: *Gain* (Verstärkung) und *Integration Time* (Messzeit). Je länger die Messzeit, desto genauer die Werte – allerdings können dann weniger Messungen pro Sekunde durchgeführt werden. Laut Dokumentation können wir aus dem gemessenen Wert die Lichtintensität in Lux wie folgt berechnen:

```
illuminance_lux = illuminance * 700 / 4 / 24
```

①

- ① Der Wert 4 beschreibt ein 4-fache Verstärkung (*Gain*) und die 24 steht für 24ms *Integration Time*

Um deinen Puls zu berechnen, müssen wir jetzt mehrere Werte in kurzen Abständen messen. Warum? Weil wir die regelmäßigen Tief- und Hochpunkte erkennen wollen. Ein Tiefpunkt entsteht, wenn dein Finger am dunkelsten ist – hier ist also gerade besonders viel Blut im Finger. Die Hochpunkte markieren dagegen den Moment, in dem das Blut größtenteils wieder zurückgeflossen ist. Jeder Herzschlag erzeugt genau einen Tief- und einen Hochpunkt. Finden wir diese Punkte, können wir einfach die Zeitabstände messen und daraus die Pulsfrequenz berechnen.

Beginnen wir damit, unsere Messungen in einer Schleife durchzuführen. Das ist eine praktische Methode, kontinuierlich Daten zu erfassen:

```

while True:
    illuminance = color_sensor.get_illuminance()
    illuminance_lux = illuminance * 700 / 4 / 24
    print(f"Lichtintensität in Lux: {illuminance_lux}")

```

Lass uns das Programm einmal ausprobieren. Es sieht aktuell so aus:

```

from tinkerforge.ip_connection import IPConnection
from tinkerforge.bricklet_color_v2 import BrickletColorV2

ip_con = IPConnection()
ip_con.connect("localhost", 4223)

color_sensor = BrickletColorV2("Wgn", ip_con)           ①
color_sensor.set_light(True)
color_sensor.set_configuration(1, 1)

while True:
    illuminance = color_sensor.get_illuminance()
    illuminance_lux = illuminance * 700 / 4 / 24
    print(f"Lichtintensität in Lux: {illuminance_lux:.2f} ", end="\r")  ②

```

- ① Denke daran, die UID durch die deines Sensors zu ersetzen.
- ② Der Parameter `end="\r"` sorgt dafür, dass nicht jede Ausgabe in eine neue Zeile geschrieben wird. Stattdessen wird immer an den Anfang der selben Zeile gesprungen. Das `\r` ist das Symbol für *Carriage Return*.

10 Protokolle

Setup

i Kommt bald

Dieses Kapitel ist in Arbeit und wird in Kürze fertiggestellt.

11 Verschlüsselung

Setup

i Kommt bald

Dieses Kapitel ist in Arbeit und wird in Kürze fertiggestellt.

12 Algorithmen

Setup

i Kommt bald

Dieses Kapitel ist in Arbeit und wird in Kürze fertiggestellt.

13 Kompression

Setup

i Kommt bald

Dieses Kapitel ist in Arbeit und wird in Kürze fertiggestellt.

14 Computer

 Kommt bald

Dieses Kapitel ist in Arbeit und wird in Kürze fertiggestellt.

Setup

14.1 Logik und Arithmetik

- Logikgatter
- 8-Bit-Additionsmaschine
- Subtraktion, Division, Multiplikation

14.2 Die von-Neumann-Architektur

14.3 Der Arbeitsspeicher oder das Kurzzeitgedächtnis des Computers

```
x = 10
print(f"Adresse im Speicher der Variable 'x': {hex(id(x))}")
x= 20
print(f"Neue Adresse im Speicher der Variable 'x': {hex(id(x))}")
```

Adresse im Speicher der Variable 'x': 0x7ffe22f97448
Neue Adresse im Speicher der Variable 'x': 0x7ffe22f97588

```
names = ["Max", "Kim", "Hildegard"]
print(f"Adresse im Speicher der Variable 'names': {hex(id(names))}")
names.append("Heinrich")
print(f"Die Adresse im Speicher der Variable 'names' bleibt identisch: {hex(id(names))}")
```

```
Adresse im Speicher der Variable 'names': 0x11055f70d40
Die Adresse im Speicher der Variable 'names' bleibt identisch: 0x11055f70d40
```

Leseempfehlung

Um tiefer in die Themen dieses Kapitels einzusteigen, empfehle ich euch Petzold (2022) zu lesen . Es lohnt sich, das Buch von Vorne nach Hinten zu verschlingen.

15 Probleme

Setup

i Kommt bald

Dieses Kapitel ist in Arbeit und wird in Kürze fertiggestellt.

Literaturverzeichnis

- Adami, Christoph. 2016. „What is Information?“ *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 374 (2063): 20150230. <https://doi.org/10.1098/rsta.2015.0230>.
- Brookshear, J. Glenn, und Dennis Brylow. 2020. *Computer science: an overview*. 13th edition, global edition. NY, NY: Pearson.
- Petzold, Charles. 2022. *Code: the hidden language of computer hardware and software*. 2. Aufl. Hoboken: Microsoft Press.
- Pólya, George, und John Horton Conway. 2004. *How to solve it: a new aspect of mathematical method*. Expanded Princeton Science Library ed. Princeton science library. Princeton [N.J.]: Princeton University Press.
- Scott, John C. 2009. *But how do it know?: the basic principles of computers for everyone*. Oldsmar, FL: John C. Scott.

Index

‘BrickletRGBLEDButton’, 18
‘IPConnection’, 18

Bibliothek, 18

Byte, 23

elektromagnetische Strahlung, 21

Klasse, 18

Methode, 19

Objekt, 18

Objektinstanz, 19

Programm, 17

RGB-Farbkodeierung, 21

Scan Code, 35

Schleife, 27

Signal, 41

Tastatur, 34