Hands-On Computer Science

Nicolas Meseth

13. Mai 2025

Inhaltsverzeichnis

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 vorhabe. 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 *Agrarsystemtechnologien* – 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.

Das LiFi-Projekt

Im Rahmen des LiFi-Projekts arbeiten wir mit 8 spannenden Geräten, die uns alle unterschiedliche Aspekte der digitalen Welt näherbringen können und mit denen wir gleichzeitig das Programmieren erlernen.

- 1 x RGB LED Bricklet 2.0
- 1 x RGB Button Bricklet
- 1 x Color Bricklet 2.0
- 1 x OLED 128x64 Bricklet 2.0
- 1 x Piezo speaker bricklet
- 2 x Distance IR 4-30cm Bricklet 2.0
- 1 x Analog In Bricklet 3.0

1 An oder Aus

Wie versprochen starten wir direkt mit unserem ersten kleinen Python-Programm:

```
from tinkerforge.ip_connection import IPConnection                    ①
from tinkerforge.bricklet_dual_button import DualButton               ②

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

① Importieren des Objekts zum Herstellen einer Verbindung mit dem Master Brick.
② Importieren des Objekts zur Darstellung des Dual Buttons als Python-Objekt.
③ Eine neue Instanz einer IP-Verbindung erstellen.
④ Eine Verbindung zu angeschlossenen Master Brick herstellen.
⑤ Eine virtuelle Instanz des Dual Button Bricklets in unserem Programm erzeugen.

- Das Binärsystem
- Bits, Bytes etc.

The concept of a switch with its two states is very important to the field of computer science. That's why introduce it early in this book.

- Introduce the dual button
- Introduce the boilerplate code necessary to connect to the button
- As as side note, introduce programs and how to write and run them with VS code
- Show how we can either press the button or not and how to read its current state from our first simple program
- Leave the chapter with some intriguing questions about the meaning of 0 and 1 in computer science

Mini Reaction Game

Theme: "Are you faster than a computer?"

Setup:

- Flash an LED randomly after a delay, and students must press the correct button as fast as possible.

- Wrong button or slow response → "fail" message.
- Display or rank response times.

Teachable moment: Real-time input processing, event handling, and how computers handle asynchronous input.

Bonus Concept: Binary Memory Challenge

Theme: "Simon Says: Binary Edition"

Setup:

- System shows a sequence of binary numbers via LED flashes (e.g., 01 → left off, right on).
- Students must repeat the sequence by pressing the buttons.
- Each round adds another number.

Teachable moment: Binary perception, memory, encoding sequences, and reinforcing input logic.

2 Farben

```
from tinkerforge.ip_connection import IPConnection                    ①
from tinkerforge.bricklet_rgb_led_v2 import BrickletRGBLEDV2          ②

ipcon = IPConnection()                                                ③
ipcon.connect("localhost", 4223)                                      ④
btn = BrickletRGBLEDV2("abC", ipcon)                                  ⑤
```

① Importieren des Objekts zum Herstellen einer Verbindung mit dem Master Brick.
② Importieren des Objekts zur Darstellung der RGB LED als Python-Objekt.
③ Eine neue Instanz einer IP-Verbindung erstellen.
④ Eine Verbindung zu angeschlossenen Master Brick herstellen.
⑤ Eine virtuelle Instanz des RGB LED Bricklets in unserem Programm erzeugen.

Think about how traffic lights use colors to communicate important information—red means stop, green means go, and yellow means caution. Similarly, on your computer or phone, colors are used everywhere: buttons turn blue when you hover over them, notifications might appear in red, and apps use different colors to help you navigate. All these colors are created and controlled using systems like RGB.

Colors are also essential in images and video. Every photo you see on your screen is made up of tiny dots called pixels, and each pixel's color is defined using the RGB system. When you watch a video, your device rapidly displays a sequence of images, each made up of millions of colored pixels, to create the illusion of motion. By mixing different amounts of red, green, and blue light for each pixel, computers can display realistic photos, vibrant animations, and lifelike videos.

The RGB system stands for Red, Green, and Blue, which are the primary colors of light. By combining these three colors in varying intensities, we can create a wide spectrum of colors. This system is widely used in computer screens, digital art, and programming to define and manipulate colors.

For example, in programming, the RGB system lets us set the color of an LED or a graphic by specifying how much red, green, and blue light to mix. Each color component—Red, Green, and Blue—is given a value from 0 (no intensity) to 255 (full intensity). By choosing different values for each component, we can create any color we want.

- Introduce RGB LED
- Introduce changing it to any color
- Introduce the RGB system as THE code system to represent colors in a computer (not going into too much detail with binary already)

Programming concepts:

- Function parameters (`set_rgb_value(r, g, b)`)

```
led.set_rgb_value(255, 255, 255)
```

3 Texte

Wir nutzen Computer ständig für Texte – sei es für eine WhatsApp-Nachnachricht, 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?

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 Entfernungen 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 gennanten *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 einen Code umgewandelt, der Zeichen abbilden kann. So wie etwa der ASCII-Code, den wir weiter unten kennenlernen.

4 Bilder

- Introduce the idea of a pixel matrix of color codes as image
- Use the OLED to ilustrate, although only b/w
- Use characters on the OLED to highlight pixel bitmaps for fonts

5 Information

6 Sensoren

- Analog vs. digital
- Build a water level sensor with a resistor and the analog in bricklet.
- Or: Moisture Sensor stuck in a plant
- Or: Touch sensor
- Or: Heartbeat sensor (color sensor)
- Integrate the RGB LED somehow
- Farbsensor

7 Signale

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.

7.1 Pulsmesser: Dein Finger als Signalquelle

Erinnerst du dich noch an den Farbsensor aus Kapitel 6? 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 Peaks. Faszinierend, oder? So kannst du beobachten, wie aus etwas so scheinbar Einfachem wie Licht ein Signal entsteht, das dir wichtige Informationen über deinen Körper liefert.

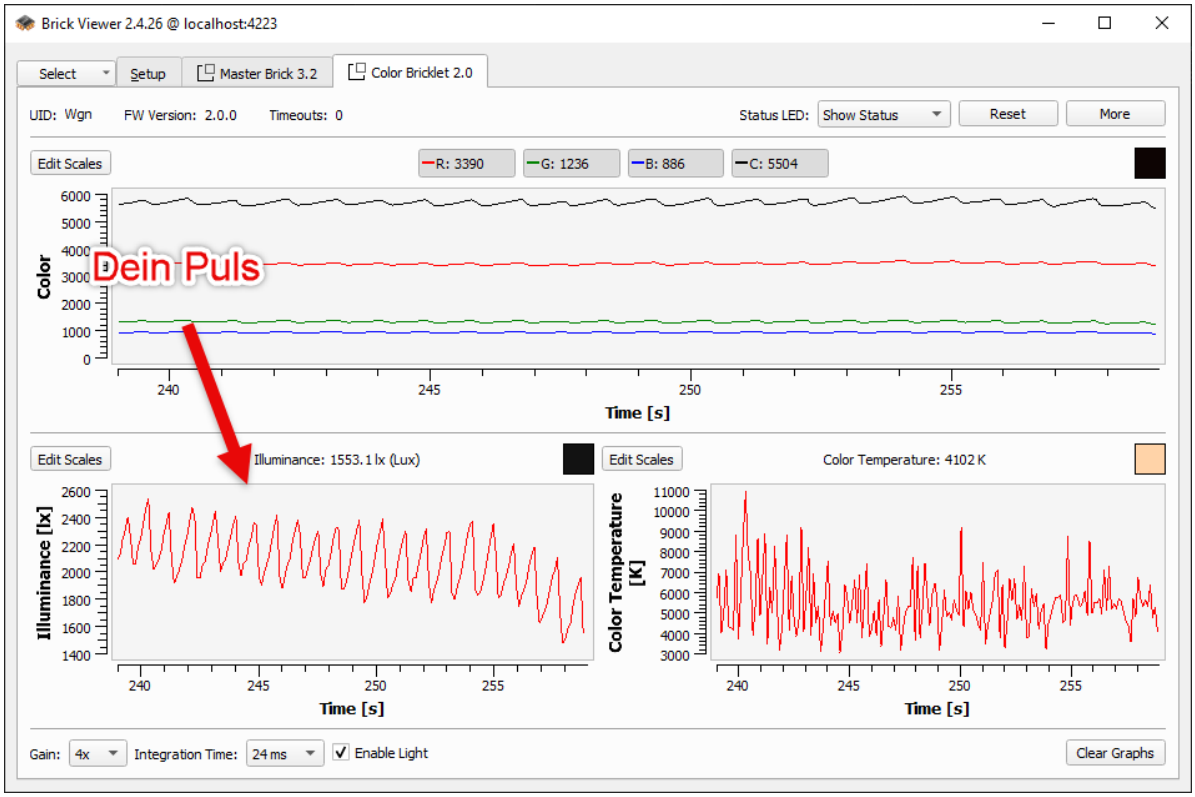

Abbildung 7.1: Dein Puls im Brick Viewer

7.2 Vom Diagramm zur Zahl

Auch wenn du das Signal im Liniendiagramm in Abbildung 7.1 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 6 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) 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*.

8 Protokolle

9 Verschlüsselung

10 Algorithmen

11 Kompression

12 Computer

12.1 Logik und Arithmetik

12.2 Die von-Neumann-Architektur

12.3 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': 0x7ffbbbf87448
Neue Adresse im Speicher der Variable 'x': 0x7ffbbbf87588
```

```
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': 0x2a8c4ac0c40
Die Adresse im Speicher der Variable 'names' bleibt identisch: 0x2a8c4ac0c40
```

13 Probleme

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