Kapitel 1

Begegnung mit Python

In diesem Kapitel verwenden Sie Python im interaktiven Modus. Sie geben in der Kommandozeile der Python-Shell einzelne Befehle ein, die der Python-Interpreter sofort ausführt. Sie lernen Operatoren, Datentypen, die Verwendung von Funktionen und den Aufbau von Termen kennen. Dabei bekommen Sie einen ersten Eindruck vom Charme und der Mächtigkeit der Programmiersprache Python. Ich gehe davon aus, dass Sie bereits ein fertig konfiguriertes Computersystem besitzen, bestehend aus SD-Karte, Tastatur, Netzteil, Monitor und natürlich – als Herzstück – den Raspberry Pi, der meist als RPi abgekürzt wird. Auf der SD-Speicherkarte ist als Betriebssystem Raspbian installiert.

Falls Sie noch nicht so weit sind, können Sie in Anhang A nachlesen, welche Komponenten Sie benötigen und wie Sie bei der Einrichtung Ihres RPi-Systems vorgehen.

1.1 Was ist Python?

Python ist eine Programmiersprache, die so gestaltet wurde, dass sie leicht zu erlernen ist und besonders gut lesbare Programmtexte ermöglicht. Ihre Entwicklung wurde 1989 von Guido van Rossum am Centrum voor Wiskunde en Informatica (CWI) in Amsterdam begonnen und wird nun durch die nichtkommerzielle Organisation »Python Software Foundation« (PSF, `http://www.python.org/psf`) koordiniert. Das Logo der Programmiersprache ist eine stilisierte Schlange. Dennoch leitet sich der Name nicht von diesem Schuppenkriechtier ab, sondern soll an die britische Comedy-Gruppe Monty Python erinnern.

Ein Python-Programm – man bezeichnet es als Skript – wird von einem Interpreter ausgeführt und läuft auf den gängigen Systemplattformen (Unix, Windows, Mac OS). Ein Programm, das auf Ihrem Raspberry Pi funktioniert, läuft in der Regel auch auf einem Windows-Rechner. Python ist kostenlos und kompatibel mit der GNU General Public License (GPL).

Python ist objektorientiert, unterstützt aber auch andere Programmierparadigmen (z. B. funktionale und imperative Programmierung). Python ist eine universelle

Programmiersprache mit vielen Einsatzmöglichkeiten. Es wird in der Wissenschaft und Technik verwendet (z. B. im Deutschen Zentrum für Luft- und Raumfahrt), aber auch für visuell-kreative Projekte (z. B. bei Disney zur Entwicklung von Computerspielen). Python hat gegenüber älteren Programmiersprachen drei Vorteile:

- Python ist leicht zu erlernen und hat eine niedrige »Eingangsschwelle«. Ohne theoretische Vorkenntnisse kann man sofort die ersten Programme schreiben. Im interaktiven Modus kann man einzelne Befehle eingeben und ihre Wirkung beobachten. Es gibt nur wenige Schlüsselwörter, die man lernen muss. Gewohnte Schreibweisen aus der Mathematik können verwendet werden, z. B. mehrfache Vergleiche wie $0 < a < 10$.
- Python-Programme sind kurz und gut verständlich. Computerprogramme werden von Maschinen ausgeführt, aber sie werden für Menschen geschrieben. Software wird meist im Team entwickelt. Programmtext muss für jedermann gut lesbar sein, damit er verändert, erweitert und verbessert werden kann. Der berühmte amerikanische Informatiker Donald Knuth hat deshalb schon vor drei Jahrzehnten vorgeschlagen, Programme als Literatur zu betrachten, so wie Romane und Theaterstücke. Nicht nur Korrektheit und Effizienz, auch die Lesbarkeit ist ein Qualitätsmerkmal.
- Programme können mit Python nachweislich in kürzerer Zeit entwickelt werden als mit anderen Programmiersprachen. Das macht Python nicht nur für die Software-Industrie interessant; auch Universitäten verwenden immer häufiger Python, weil so weniger Zeit für den Lehrstoff benötigt wird.

1.2 Python-Versionen

Auf dem Raspberry Pi sind zwei Versionen von Python installiert:

- Python 2 (in der Version 2.7)
- Python 3 (in der Version 3.7).

Dieses Buch bezieht sich allein auf das modernere Python 3. Aber es ist wichtig zu wissen, dass es noch eine andere Version gibt. Python 3 ist nicht kompatibel zu Python 2. Das heißt, ein Python-3-Interpreter kann mit einem Programm, das in Python 2 geschrieben worden ist, in der Regel nichts anfangen.

Das neuere Python 3 ist konsistenter und konzeptionell »sauberer«. Das bedeutet, es gibt mehr Regelmäßigkeit und weniger Ausnahmen. Deshalb ist Python 3 vielleicht etwas leichter zu erlernen und die Programme, die in Python 3 geschrieben worden sind, sind etwas leichter zu durchschauen. Aber groß sind die Unterschiede nicht.

Beide Versionen existieren parallel. Auch Python 2 wird weiterhin gepflegt.

1.3 IDLE

Zur Standardinstallation von Python gehört eine integrierte Entwicklungsumgebung namens IDLE. Der Name erinnert an den englischen Schauspieler, Autor und Komponisten Eric Idle, ein Mitglied der Comedy-Gruppe Monty Python.

IDLE ist nicht (mehr) in der Standarddistribution von Raspberry Pi OS enthalten. Sie können es aber in wenigen Schritten installieren:

- Klicken Sie oben links auf den STARTBUTTON mit der Himbeere, dann auf EINSTELLUNGEN und dann auf ADD/REMOVE SOFTWARE (SOFTWARE HINZUFÜGEN ODER ENTFERNEN). Es erscheint ein Fenster wie in Abbildung 1.1.
- Geben Sie oben links als Suchbegriff `Idle` ein und drücken Sie Enter. Es erscheint eine Liste von Software-Angeboten.
- Rollen Sie etwas herunter und wählen Sie DIE IDLE-Version, die zu Ihrer Python-Version passt (*idle-3.7.3*), indem Sie die Checkbox vor dem Eintrag anklicken.
- Klicken Sie unten rechts auf OK.
- Zur Sicherheit müssen Sie Ihr Passwort eingeben. (Wenn Sie das vorgegebene Passwort nicht geändert haben, ist es `raspberry`.)
- Dann wird IDLE installiert.

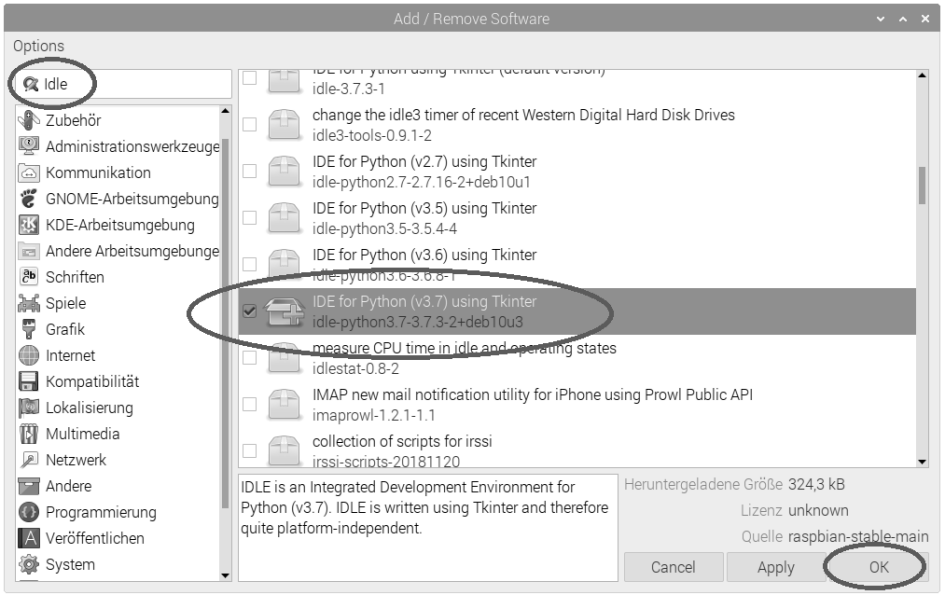

Abb. 1.1: IDLE installieren

Klicken Sie oben links auf den STARTBUTTON mit der Himbeere und öffnen Sie das Untermenü ENTWICKLUNG. Sie sehen, dass das Menü einen neuen Eintrag enthält: PYTHON 3 (IDLE). Wenn Sie diesen Eintrag anklicken, öffnet sich die Entwicklungsumgebung.

IDLE besteht im Wesentlichen aus drei Komponenten:

- **Die Python-Shell.** Wenn Sie IDLE starten, öffnet sich zuerst das Python-Shell-Fenster. Die Shell ist eine Anwendung, mit der Sie direkt mit dem Python-Interpreter kommunizieren können: Sie können auf der Kommandozeile einzelne Python-Anweisungen eingeben und ausführen lassen. Ein Python-Programm, das eine Bildschirmausgabe liefert, gibt diese in einem Shell-Fenster aus.
- **Der Programmeditor.** Das ist eine Art Textverarbeitungsprogramm zum Schreiben von Programmen. Sie starten den Programmeditor vom Shell-Fenster aus (FILE|NEW FILE).
- **Der Debugger.** Er dient dazu, den Lauf eines Programms zu kontrollieren und zu überwachen, um logische Fehler zu finden.

Neben IDLE gibt es natürlich noch viele andere Entwicklungsumgebungen für Python. Auf dem RPi sind standardmäßig noch GEANY und THONNY installiert.

1.3.1 Die Python-Shell

Anweisungen sind die Bausteine von Computerprogrammen. Mit der Python-Shell können Sie einzelne Python-Anweisungen ausprobieren. Man spricht auch vom interaktiven Modus, weil Sie mit dem Python-Interpreter eine Art Dialog führen: Sie geben über die Tastatur einen Befehl ein – der Interpreter führt ihn aus und liefert eine Antwort.

Öffnen Sie das Shell-Fenster der Python-3-Version auf Ihrem Rechner (STARTBUTTON|ENTWICKLUNG|PYTHON 3 (IDLE)). Da Sie ständig mit IDLE arbeiten werden, lohnt es sich, das Programmicon auf den Desktop zu bringen. Das geht so: Sie klicken mit der rechten Maustaste auf das Icon PYTHON 3 und wählen im Kontextmenü den Befehl DEM DESKTOP HINZUFÜGEN. Die Python-Shell meldet sich immer mit einer kurzen Information über die Version und einigen weiteren Hinweisen.

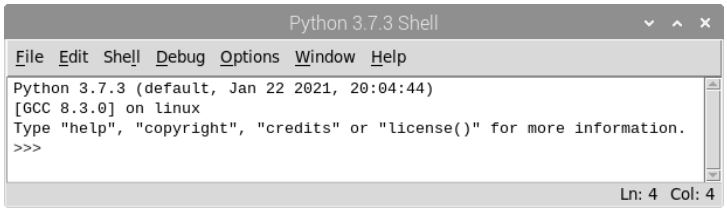

Abb. 1.2: Die Python-Shell der Entwicklungsumgebung IDLE

Die unterste Zeile beginnt mit einem Promptstring aus drei spitzen Klammern >>> als Eingabeaufforderung. Das ist die Kommandozeile. Hinter dem Prompt können Sie eine Anweisung eingeben. Wenn Sie die Taste ↵ drücken, wird der Befehl ausgeführt. In den nächsten Zeilen kommt entweder eine Fehlermeldung, ein Ergebnis oder manchmal auch *keine* Systemantwort. Probieren Sie aus:

```
>>> 2+2
4
```

Hier ist die Anweisung ein mathematischer Term. Wenn Sie ↵ drücken, wird der Term ausgewertet (also die Rechnung ausgeführt) und in der nächsten Zeile (ohne Prompt) das Ergebnis dargestellt.

```
>>> 2 +
SyntaxError: invalid syntax
```

Jetzt haben Sie einen ungültigen Term eingegeben (der zweite Summand fehlt). Dann folgt eine Fehlermeldung.

```
>>> a = 1257002
>>>
```

Eine solche Anweisung nennt man eine Zuweisung. Der Variablen `a` wird der Wert `1257002` zugewiesen. Dabei ändert sich zwar der Zustand des Python-Laufzeitsystems (es merkt sich eine Zahl), aber es wird nichts ausgegeben. Sie sehen in der nächsten Zeile sofort wieder den Prompt. Die gespeicherte Zahl können Sie wieder zum Vorschein bringen, indem Sie den Variablennamen eingeben:

```
>>> a
1257002
```

1.3.2 Hotkeys

Um effizient mit der Python-Shell arbeiten zu können, sollten Sie einige Tastenkombinationen (Hotkeys) beherrschen.

Manchmal möchten Sie ein früheres Kommando ein zweites Mal verwenden – vielleicht mit kleinen Abänderungen. Dann benutzen Sie die Tastenkombination Alt+P. Beispiel:

```
>>> 1 + 2*3 + 4
11
>>>
```

Wenn Sie jetzt *einmal* die Tastenkombination Alt+P betätigen, erscheint hinter dem letzten Prompt wieder das vorige Kommando (*previous*):

```
>>> 1 + 2*3 + 4
```

Wenn Sie nochmals diesen Hotkey drücken, verschwindet der Term wieder und es erscheint das *vorletzte* Kommando, beim nächsten Mal das vorvorletzte und so weiter. Die Shell merkt sich alle Ihre Eingaben in einer Folge, die man *History* nennt. Mit Alt+P und Alt+N können Sie in der History rückwärts- und vorwärtsgehen (Abbildung 1.3).

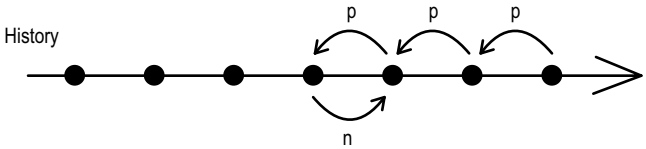

Abb. 1.3: Navigieren in der History mit Alt+P und Alt+N

Eine dritte wichtige Tastenkombination, die Sie sich merken sollten, ist Strg+C. Damit können Sie die Ausführung des gerade laufenden Programms abbrechen, z. B. wenn sie zu lange dauert.

Tastenkombination	Wirkung
Alt+P	Previous Command. Die vorige Anweisung in der History (Liste der bisherigen Anweisungen) erscheint hinter dem Prompt.
Alt+N	Next Command. Die nachfolgende Anweisung in der History erscheint hinter dem Prompt.
Strg+C	Keyboard Interrupt. Der Abbruch eines Programms (z. B. bei einer Endlosschleife) wird erzwungen.

Tabelle 1.1: Wichtige Tastenkombinationen der Python-Shell

1.4 Die Python-Shell als Taschenrechner

Die Python-Shell können Sie wie einen mächtigen und komfortablen Taschenrechner benutzen. Sie geben einen mathematischen Term ein, drücken ↵ und erhalten das Ergebnis in der nächsten Zeile.

1.4.1 Operatoren und Terme

Ein mathematischer Term (Ausdruck) kann aus Zahlen, Operatoren und Klammern aufgebaut werden. Die Schreibweise ist manchmal ein kleines bisschen

anders als in der Schulmathematik. Zum Beispiel dürfen Sie beim Multiplizieren den Multiplikationsoperator * niemals weglassen. Das Kommando

```
>>> (1 + 1) (6 - 2)
```

führt zu einer Fehlermeldung. Korrekt ist:

```
>>> (1 + 1) * (6 - 2)
8
```

Es gibt keine langen Bruchstriche. Für Zähler oder Nenner müssen Sie Klammern verwenden:

```
>>> (2 + 3) / 2
2.5
```

Python unterscheidet zwischen der exakten Division / und der ganzzahligen Division //. Die ganzzahlige Division liefert eine ganze Zahl, und zwar den nach unten gerundeten Quotienten. Probieren Sie aus:

```
>>> 3/2
1.5
>>> 3//2
1
```

Die Modulo-Operation % liefert den Rest, der bei einer ganzzahligen Division übrig bleibt. Beispiel: 7 geteilt durch 3 ist 2 Rest 1.

```
>>> 7 // 3
2
>>> 7 % 3
1
```

Zum Potenzieren einer Zahl verwenden Sie den Operator **. Beachten Sie, dass Sie mit Python fast beliebig große Zahlen berechnen können.

```
>>> 2**3
8
>>> 2**-3
0.125
>>> 2**0.5
```

```
1.4142135623730951
>>> 137 ** 57
62097244310190258855130481009768710553783221891824568918264378730801621731509707020422858215922309341135893663853254591817
```

Bei Termen mit mehreren Operatoren müssen Sie deren Prioritäten beachten (Tabelle 1.2). Ein Operator mit höherer Priorität wird zuerst ausgewertet. Der Potenzoperator hat die höchste Priorität, Addition und Subtraktion die niedrigste.

```
>>> 2*3**2
18
>>> (2*3)**2
36
```

Operator	Bedeutung
`**`	Potenz, `x**y` = x^y
`*, /, //`	Multiplikation, Division und ganzzahlige Division
`%`	Modulo-Operation. Der Rest einer ganzzahligen Division.
`+, -`	Addition und Subtraktion

Tabelle 1.2: Arithmetische Operatoren in der Reihenfolge ihrer Priorität

1.4.2 Zahlen

Wer rechnet, verwendet Zahlen. Mit Python können Sie drei Typen von Zahlen verarbeiten:

- Ganze Zahlen (`int`)
- Gleitpunktzahlen (`float`)
- Komplexe Zahlen (`complex`)

Was ist überhaupt eine Zahl? In der Informatik unterscheidet man zwischen dem abstrakten mathematischen Objekt und der Ziffernfolge, die eine Zahl darstellt. Letzteres nennt man auch *Literal*. Für ein und dieselbe Zahl im mathematischen Sinne, sagen wir die `13`, gibt es unterschiedliche Literale, z. B. `13` oder `13.0` oder `13.0000`. Drei Schreibweisen – eine Zahl.

Ganze Zahlen (Typ int)

Literale für ganze Zahlen sind z. B. `12`, `0`, `-3`. Ganze Dezimalzahlen bestehen aus den zehn Ziffern `0`, `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`. Es kann ein negatives Vorzeichen vor die erste Ziffer geschrieben werden. Eine ganze Dezimalzahl darf nicht mit einer Null beginnen. Probieren Sie es aus!

```
>>> 023
SyntaxError: invalid token
```

Ganze Zahlen dürfen bei Python beliebig lang sein. Eine Grenze ist nur durch die Speicherkapazität des Rechners gegeben. Probieren Sie es aus:

Ganze Zahlen sind vom Typ `int` (*engl.* integer = unversehrt, ganz). Mit der Funktion `type()` können Sie den Typ eines Literals abfragen:

```
>>> type(13)
<class 'int'>
>>> type(13.0)
<class 'float'>
```

Sie sehen, dass die Literale `13` und `13.0` zu verschiedenen Typen gehören, obwohl sie den gleichen mathematischen Wert darstellen. Was hat es mit dem Begriff `class` auf sich? Der Typ `int` wird durch eine Klasse (engl. *class*) realisiert. Eine Klasse kann man sich als eine Art Bauplan für Objekte eines Typs vorstellen. In der Klasse `int` ist zum Beispiel definiert, wie die Operationen für ganze Zahlen ausgeführt werden. Mehr Informationen zu Klassen finden Sie in Abschnitt 3.8.2.

Binär, oktal und hexadezimal – andere Schreibweisen für ganze Zahlen

Üblicherweise verwenden wir das dezimale Zahlensystem. Es gibt aber auch Binärzahlen (mit nur zwei Ziffern 0 und 1), Oktalzahlen (mit acht verschiedenen Ziffern) und Hexadezimalzahlen (mit 16 Ziffern).

Wie das Dezimalsystem ist auch das *Binärsystem* (Dualsystem, Zweiersystem) ein Stellenwertsystem für Zahlen. Aber anstelle von zehn Ziffern gibt es nur zwei, die Null und die Eins. Jede Zahl lässt sich als Summe von Zweierpotenzen (`1`, `2`, `4`, `8`, `16`, ...) darstellen. Die Binärzahl `10011` hat den Wert

```
1*16 + 0*8 + 0*4 + 1*2 + 1*1 = 16 + 2 + 1 = 19
```

Nun muss man natürlich erkennen können, ob eine Ziffernfolge wie `10011` als Dezimalzahl (zehntausendundelf) oder als Binärzahl gemeint ist. Deshalb beginnen bei Python Literale für Binärzahlen immer mit der Ziffer `0` und dem Buchstaben `b`, also z. B. `0b10011`. Wenn Sie in der Python-Shell eine solche Ziffernfolge eingeben und ↵ drücken, erscheint in der nächsten Zeile der Wert als Dezimalzahl:

```
>>> 0b10011
19
```

Mit der Funktion `bin()` können Sie zu einer Dezimalzahl die Binärdarstellung berechnen lassen:

```
>>> bin(19)
'0b10011'
```

Das *Oktalsystem* verwendet die Zahl 8 als Zahlenbasis. Jede Oktalzahl repräsentiert eine Summe aus Achterpotenzen (1, 8, 64, 512, ...). Bei Python beginnen die Literale von Oktalzahlen mit der Ziffer `0` und dem Buchstaben `o` oder `O`. Beispiel:

```
>>> 0o210
136
```

Der Dezimalwert berechnet sich so: `2*64 + 1*8 + 0*1 = 128 + 8 = 136`

Im *Hexadezimalsystem* ist 16 die Zahlenbasis. Eine Hexadezimalzahl repräsentiert also eine Summe aus Potenzen der Zahl 16. Die 16 Ziffern werden durch die üblichen Dezimalziffern `0` bis `9` und die sechs ersten Buchstaben des Alphabets dargestellt. Dabei hat `A` den Wert `10`, `B` den Wert `11` usw. Bei Python beginnen Hexadezimalzahlen immer mit den Zeichen `0x` oder `0X`. Das erste Zeichen ist die Ziffer null und nicht der Buchstabe O. Beispiel:

```
>>> 0x10A
266
```

Der Dezimalwert berechnet sich so: `1*256 + 0*16 + 10*1 = 256 + 10 = 266`

Die Tatsache, dass die 16 Ziffern der Hexadezimalzahlen auch Buchstaben enthalten, hat Programmierer zum *Hexspeak* inspiriert. Zahlen, die in einem Programmsystem eine besondere Bedeutung haben (*magical numbers*), werden gerne so gewählt, dass ihre Hexadezimaldarstellung ein sinnvolles Wort ist.

```
>>> 0xCAFE
51966
>>> 0xBADBEEF
195935983
>>> xABAD1DEA
2880249322
```

Gleitkommazahlen (Typ float)

Gleitpunktzahlen oder Gleitkommazahlen (engl. *floating point numbers*) sind Dezimalbrüche. Meist schreibt man eine Gleitkommazahl als eine Folge von Ziffern mit einem einzigen Punkt auf. In der Schulmathematik verwenden wir in Deutschland ein Komma, in Python und allen anderen Programmiersprachen

wird die angelsächsische Schreibweise verwendet, bei der ein Punkt an die Stelle des Kommas tritt. Es ist ein bisschen seltsam, von einer Gleitkommazahl zu sprechen und dann einen Punkt zu schreiben. Um diesen Widerspruch zu vermeiden, verwenden viele Leute den Begriff Gleich*punkt*zahl. »Gleitkommazahl« ist übrigens ein Gegenbegriff zu »Festkommazahl«. Eine Festkommazahl ist ein Dezimalbruch mit einer festen Anzahl von Nachkommastellen. Zum Beispiel geben wir Geldbeträge in Euro immer mit zwei Nachkommastellen an. Wir schreiben 3,50 Euro anstelle von 3,5 Euro.

Gültige Python-Gleitkommazahlen sind

- `3.14` oder `0.2` oder `0.00012`
- `.2` (eine Null vor dem Punkt darf man auch weglassen)
- `5.` (eine Null nach dem Punkt darf man weglassen)

Das Literal 5 ist dagegen keine Gleitkommazahl (Punkt fehlt).

Für Zahlen, die sehr nahe bei 0 liegen oder sehr groß sind, wird die Exponentialschreibweise verwendet. Dabei wird die Zahl als Produkt einer rationalen Zahl `m` (Mantisse) mit einer Zehnerpotenz mit dem Exponenten `e` dargestellt:

$z = m*10^e$

Beispiele:

$123000000 = 123*10^6$
$0.00012 = 1.2*10^{-4}$

Bei Python ist eine Gleitkommazahl in Exponentialschreibweise so aufgebaut: Sie beginnt mit einem Dezimalbruch oder einer ganzen Zahl (ohne Punkt) für die Mantisse. Danach folgt der Buchstabe `e` oder `E`, ein Vorzeichen (+ oder -), das bei positiven Exponenten auch weggelassen werden kann, und schließlich eine ganze Zahl als Exponent.

Gültige Literale sind:

- `1.0e-8` – entspricht der Zahl 0.00000001
- `2.1E+7` – entspricht der Zahl 21000000
- `.2e0` – entspricht der Zahl 0.2
- `001e2` – entspricht der Zahl 100 (mehrere führende Nullen sind erlaubt)

Ungültig sind:

- `0.1-E7` – (Minuszeichen vor dem E)
- `1.2e0.3` – (keine ganze Zahl als Exponent)

Mantisse und Exponent sind immer Dezimalzahlen und niemals Oktal- oder Hexadezimalzahlen.

Gleitkommazahlen sind vom Datentyp `float`. Mit der Funktion `type()` können Sie das nachprüfen:

```
>>> type(1.2)
<class 'float'>
```

Im Unterschied zu ganzen Zahlen (Typ `int`), die immer exakt sind, haben Gleitkommazahlen nur eine begrenzte Genauigkeit. Gibt man längere Ziffernfolgen ein, so werden die letzten Stellen einfach abgetrennt.

```
>>> 1.2345678901234567890
1.2345678901234567
```

Komplexe Zahlen (Typ complex)

Komplexe Zahlen werden als Summe aus einem Real- und einem Imaginärteil beschrieben:

```
c = a + b*i
```

Dabei bezeichnet der Buchstabe `i` die Wurzel aus –1. Python verwendet (wie in der Elektrotechnik üblich) den Buchstaben `j` oder `J` anstelle von `i`, um Verwechslungen mit dem Symbol für die Stromstärke zu vermeiden. Beispiele sind:

```
0.3j
1.2e-3J
12 + 20j
```

Keine gültige komplexe Zahl ist übrigens der Buchstabe `j` alleine. Die imaginäre Zahl `i` schreiben Sie `1j`.

1.4.3 Mathematische Funktionen

Wenn Sie die Python-Shell als Taschenrechner verwenden wollen, benötigen Sie auch mathematische Funktionen wie Sinus, Kosinus, die Quadratwurzelfunktion oder die Exponentialfunktion. Nun stellt Python eine Reihe von Standardfunktionen zur Verfügung, die gewissermaßen fest in die Sprache eingebaut sind (*built-in functions*). Wir haben schon die Funktion `type()` verwendet, die den Typ eines Objekts (z. B. einer Zahl) zurückgibt. Nur wenige Standardfunktionen haben eine mathematische Bedeutung (Tabelle 1.3). Der Aufruf einer Funktion ist so aufgebaut: Zuerst kommt der Name der Funktion, dahinter folgen in runden Klammern

die Argumente. Das sind Objekte, die die Funktion verarbeitet, um daraus einen neuen Wert zu berechnen und zurückzugeben. Statt *Argument* sagt man manchmal auch *Parameter*. Beispiel:

```
>>> abs(-12)
12
```

Hier ist `abs` der Name der Funktion und die Zahl `-12` das Argument. Die Funktion `abs()` liefert den positiven Wert einer Zahl. Die Funktion ist einstellig, das heißt, sie akzeptiert immer nur genau ein Argument. Wenn Sie zwei Argumente angeben, erhalten Sie eine Fehlermeldung:

```
>>> abs(-2, 5)
Traceback (most recent call last):
  File "<pyshell#33>", line 1, in <module>
    abs(-2, 5)
TypeError: abs() takes exactly one argument (2 given)
```

Es gibt aber auch Funktionen, die man mit einer unterschiedlichen Anzahl von Argumenten aufrufen kann. So liefert `min()` die kleinste Zahl von den Zahlen, die als Argumente übergeben worden sind.

```
>>> min (3, 2)
2
>>> min (2.5, 0, -2, 1)
-2
```

Die Funktion `round()` können Sie mit einem oder zwei Argumenten aufrufen. Das erste Argument ist die Zahl, die gerundet werden soll. Das zweite Argument ist optional und gibt die Anzahl der Nachkommastellen an, auf die gerundet werden soll. Fehlt das zweite Argument, gibt die Funktion eine ganze Zahl zurück.

```
>>> round(1.561)
2
>>> round(1.561, 1)
1.6
```

Funktion	Erklärung
`abs(`*x*`)`	Liefert den absoluten (positiven) Wert einer Zahl *x*.
`float(`*x*`)`	Liefert zu einer Zahl (oder einem anderen Objekt) eine Gleitkommazahl.

Tabelle 1.3: Mathematische Standardfunktionen (müssen nicht importiert werden)

Funktion	Erklärung
`int(x)`	Liefert zu einer Gleitkommazahl (oder einem anderen Objekt) eine nach unten gerundete ganze Zahl.
`max(x0, ..., xn)`	Liefert die größte Zahl von *x0, ... , xn*.
`min(x0, ..., xn)`	Liefert die kleinste Zahl von *x0, ... , xn*.
`round(x,[n])`	Die Zahl *x* wird auf *n* Stellen nach dem Komma gerundet und das Ergebnis zurückgegeben.

Tabelle 1.3: Mathematische Standardfunktionen (müssen nicht importiert werden) (Forts.)

Module importieren

Die meisten mathematischen Funktionen gehören nicht zu den Standardfunktionen. Will man sie benutzen, muss man zunächst das Modul `math` importieren. Module sind Sammlungen von Funktionen, Klassen und Konstanten zu einer Thematik. Sie sind so etwas wie Erweiterungen der Grundausstattung. Das Modul `math` enthält z. B. die Konstanten `e` und `pi` und mathematische Funktionen wie die Quadratwurzelfunktion `sqrt()`. Für Python gibt es Tausende von Modulen. Die wichtigsten gehören zum Standardpaket von Python und sind auf Ihrem RPi bereits installiert. Speziellere Module müssen zuerst aus dem Internet heruntergeladen werden. Dazu später mehr (z. B. in Abschnitt 1.7).

Sie können ein Modul auf verschiedene Weisen importieren. Beispiele:

```
>>> import math
>>> import math as m
>>> from math import *
>>> from math import pi, sqrt
```

Mit dem Befehl

```
>>> import math
```

importieren Sie den Modulnamen. Wenn Sie eine Funktion aufrufen wollen, müssen Sie dem Funktionsnamen noch den Modulnamen voranstellen.

```
>>> math.sqrt(4)
2.0
>>> math.pi
3.141592653589793
```

Sie können ein Modul unter einem anderen Namen importieren. Das ist vor allem dann praktisch, wenn ein Modul einen langen Namen hat, den man im Programmtext nicht immer wieder ausschreiben will.

```
>>> import math as m
>>> m.pi
3.141592653589793
```

Mit dem Befehl

```
>>> from math import *
```

importieren Sie alle Namen des Moduls `math`. Sie können dann die Funktionen und Konstanten ohne vorangestellten Modulnamen verwenden.

```
>>> sqrt(4)
2.0
>>> pi
3.141592653589793
```

Sie können auch gezielt nur die Namen importieren, die Sie auch verwenden wollen. Dann müssen Sie die Namen der benötigten Funktionen und Konstanten auflisten. Beispiel:

```
>>> from math import pi, sqrt
>>> sqrt(4)
2.0
>>> exp(1)
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    exp(1)
NameError: name 'exp' is not defined
```

Die Funktion `exp()` wurde nicht importiert und bleibt deswegen unbekannt.

Mathematische Funktionen und Konstanten

Wenn Sie die Python-Shell als wissenschaftlichen Taschenrechner verwenden wollen, geben Sie einmal die Import-Anweisung

```
>>> from math import *
```

ein. Dann können Sie auf einen Fundus mathematischer Funktionen und Konstanten zurückgreifen. Hier einige Beispiele:

```
>>> degrees (2*pi)
360.0
>>> radians(180)
3.141592653589793
```

Winkel können in Grad ° oder als Bogenmaß angegeben werden. Der Winkel von 360° entspricht dem Bogenmaß 2π.

Mithilfe der Angaben aus Abbildung 1.4 soll die Höhe des Turms berechnet werden.

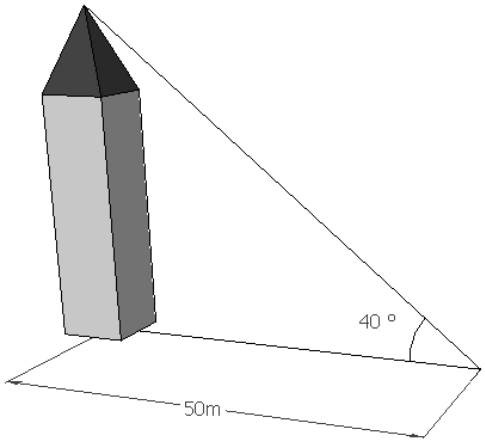

Abb. 1.4: Bestimmung der Höhe eines Turms durch Anpeilen der Turmspitze

Nun verarbeiten die trigonometrischen Funktionen (`sin()`, `cos()`, ...) des Moduls `math` keine Winkel (0° bis 360°), sondern Bogenmaße (Radianten). Deshalb muss der Winkel aus der Abbildung in den Radianten umgerechnet werden. Die Höhe des Turms können Sie mit folgendem Ausdruck berechnen:

```
>>> tan(radians(40))*50
41.954981558864
```

Funktion/Konstante	Erklärung
`acos(x)`	Arcuskosinus von *x* (Bogenmaß)
`asin(x)`	Arcussinus von *x* (Bogenmaß)
`atan(x)`	Arcustangens von *x* (Bogenmaß)
`cos(x)`	Kosinus von *x* (*x* ist das Bogenmaß eines Winkels)
`cosh(x)`	Hyperbolischer Kosinus von *x*
`degrees(x)`	Liefert zu einem Winkel, der als Bogenmaß angegeben ist, den Winkel in Grad.
`e`	Die mathematische Konstante `e == 2.7182...`
`exp (x)`	e^x
`fabs(x)`	Absolutwert der Gleitkommazahl *x*

Tabelle 1.4: Die wichtigsten Funktionen und Konstanten des Moduls `math`

Funktion/Konstante	Erklärung
`floor(x)`	Der nach unten gerundete Wert von *x*
`log (x)`	Natürlicher Logarithmus von *x* (Basis e)
`log10(x)`	Logarithmus von *x* zur Basis 10
`modf(x)`	Liefert ein Paar bestehend aus dem Nachkommateil und dem ganzzahligen Teil der Gleitkommazahl *x*. Beispiel: `modf(4.2)` ergibt `(0.2, 4.0)`.
`radians(x)`	Berechnet zum Winkel von *x* Grad das Bogenmaß.
`sin(x)`	Sinus von *x* (*x* ist das Bogenmaß eines Winkels)
`sinh(x)`	Hyperbolischer Sinus von *x*
`pi`	Die mathematische Konstante `pi == 3.1415...`
`sqrt(x)`	Quadratwurzel von *x*
`tan(x)`	Tangens von *x* (*x* ist das Bogenmaß eines Winkels)
`tanh(x)`	Hyperbolischer Tangens von *x*

Tabelle 1.4: Die wichtigsten Funktionen und Konstanten des Moduls `math` (Forts.)

1.5 Hilfe

Mit der Funktion `help()` können Sie detaillierte Informationen zu allen Sprachelementen von Python abfragen. Sie verwenden `help()` auf zwei unterschiedliche Weisen: mit und ohne Argument.

Wenn Sie Informationen zu einer bestimmten Funktion (oder einem anderen Objekt) benötigen, übergeben Sie den Namen als Argument. Beispiel:

```
>>> help(round)
Help on built-in function round in module builtins:
round(...)
    round(number[, ndigits]) -> number
...
```

Wenn Sie nicht genau wissen, was Sie suchen, geben Sie `help()` ohne Argument ein. Dann wechselt die Python-Shell in einen interaktiven Hilfe-Modus. Sie sehen einen neuen Prompt:

```
help>
```

Das Hilfesystem gibt Ihnen Hinweise, was Sie nun tun können. Wenn Sie z. B. eine Liste aller verfügbaren Module haben wollen, geben Sie `modules` ein. Um den Hilfe-Modus zu verlassen, geben Sie `quit` ein.

```
help> quit
You are now leaving help and returning to the Python interpreter ...
>>>
```

1.6 Namen und Zuweisungen

Namen für Objekte spielen in der Programmierung eine wichtige Rolle. Bei einer Zuweisung wird ein Name mit einem Objekt (z. B. die Zahl 12) verbunden. Über den Namen kann man auf das Objekt später wieder zugreifen. Der Zuweisungsoperator ist ein Gleichheitszeichen. Links vom Gleichheitszeichen steht ein Name, rechts ein Ausdruck.

Die einfachste Form der Zuweisung hat die Form

```
name = wert
```

Beispiel:

```
>>> x = 12
```

Über den Namen `x` kann man auf den zugeordneten Wert wieder zugreifen:

```
>>> x
12
```

Eine solche Zuweisung kann man sich anschaulich als Speichern von Daten vorstellen. Die Variable ist eine Art Behälter. Der Name `x` ist ein Etikett auf dem Behälter und der Zahlenwert `12` ist der Inhalt des Behälters.

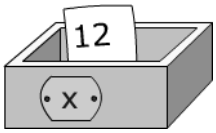

Abb. 1.5: Visualisierung einer Variablen als Behälter mit Inhalt

Wenn ein Name einmal in einer Zuweisung verwendet worden ist, kann er in Ausdrücken anstelle des Zahlenwerts verwendet werden.

```
>>> x*2
24
```

Das Einführen von Namen ist bei Python ein sehr einfacher Mechanismus. Im Unterschied zu anderen Programmiersprachen (wie z. B. Java) müssen Namen nicht deklariert und mit einem Datentyp verknüpft werden.

Statt *Name* sagt man im Deutschen häufig auch *Bezeichner*. Beides meint dasselbe. Das Wort *Bezeichner* ist eine schlechte Übersetzung des englischen Begriffs *identifier*. Der englische Begriff weist auf die wichtigste Funktion von Namen hin, nämlich Objekte eindeutig zu identifizieren.

Der Inhalt einer Variablen kann einer anderen Variablen zugewiesen werden.

```
>>> y = x
>>> y
12
```

Nun trägt y denselben Wert wie x. Diese Zuweisung stellen sich viele Leute wie im Abbildung 1.6 vor. Die Variable y erhält eine Kopie des Inhalts von x.

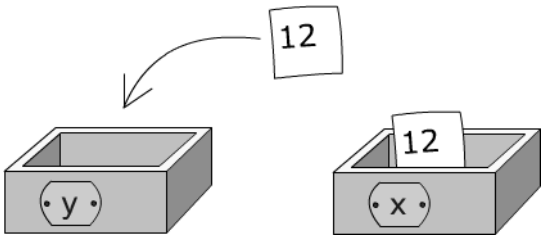

Abb. 1.6: Visualisierung einer Wertübertragung

Namen kann man sich auch als Etiketten vorstellen, die an Objekten (wie Zahlen) kleben. Man kann auch sagen, dass die Zahl 12 nun an zwei Namen gebunden ist. In vielen Fällen ist diese Etiketten-Intuition angemessener.

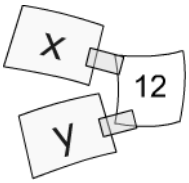

Abb. 1.7: Zuweisen als Hinzufügen eines neuen Namens

1.6.1 Zuweisungen für mehrere Variablen

Die Programmiersprache Python ist so gestaltet, dass man möglichst kurze und verständliche Programmtexte schreiben kann. Das merkt man an vielen Stellen –

auch bei Zuweisungen. Sie können in einer einzigen Anweisung mehrere Variablen mit dem gleichen Wert belegen. Anstelle von

```
>>> x = 1
>>> y = 1
```

schreiben Sie

```
>>> x = y = 1
```

Und so ordnen Sie in einer einzigen Zuweisung mehreren Variablen *unterschiedliche* Werte zu:

```
>>> x, y = 1, 2
```

Prüfen Sie nach:

```
>>> x
1
>>> y
2
```

Es ist möglich, in einer einzigen Zuweisung ohne Hilfsvariable die Werte zweier Variablen zu vertauschen.

```
>>> x, y = 1, 2
>>> x, y = y, x
>>> x
2
>>> y
1
```

1.6.2 Rechnen mit Variablen in der Shell

Im Unterschied zur Mathematik sind Namen in Computerprogrammen meist nicht nur einzelne Buchstaben (wie z. B. `x`), sondern ganze Wörter, die eine sinnvolle Bedeutung haben (»sprechende Namen«). Das erleichtert das Verstehen von Berechnungen.

```
>>> from math import pi
>>> erddurchmesser = 12756
>>> erdradius = erddurchmesser/2
>>> erdvolumen = 4/3 * pi * erdradius ** 3
```

```
>>> erdoberfläche = 4 * pi * erdradius ** 2
>>> erdvolumen
1086781292542.8892
>>> round(erdoberfläche)
511185933
```

1.6.3 Syntaxregeln für Bezeichner

Man darf nicht jede beliebige Zeichenfolge als Namen (Bezeichner) verwenden. Es sind folgende Regeln zu beachten:

1. Ein Bezeichner darf kein Python-Schlüsselwort sein (*keyword*). Schlüsselwörter sind reservierte Wörter, die eine festgelegte Bedeutung haben, z. B. `not` (Negation einer logischen Aussage) oder `if` (Start einer bedingten Anweisung). Eine vollständige Liste der Schlüsselwörter erhalten Sie in der Python-Shell über die Hilfe-Funktion. Geben Sie zuerst `help()` ein und wechseln Sie in den Hilfemodus (der Prompt ist nun `help>`). Dann geben Sie das Kommando `keywords` ein. Mit `quit` gelangen Sie wieder in den Interpreter-Modus der Shell.
2. Ein Bezeichner besteht nur aus Buchstaben (`a...z`), Ziffern (`0...9`) oder Unterstrichen (`_`).
3. Ein Bezeichner muss mit einem Buchstaben oder Unterstrich beginnen.

Gültige Bezeichner sind: `summe`, `__summe`, `summe_1`, nicht aber `1_summe` (Ziffer zu Beginn ist nicht erlaubt), `summe-1` (Minuszeichen ist nicht erlaubt).

Tückisch ist es, wenn Sie (vielleicht aus Versehen) einen Funktionsnamen als Variablennamen verwenden. Das ist zwar erlaubt, aber Sie können anschließend die Funktion nicht mehr aufrufen. Beispiel:

```
>>> int(1.2)
1
```

Die Funktion `int()` liefert zu einer Zahl eine ganze Zahl. Nun ordnen wir dem Namen `int` eine Zahl zu.

```
>>> int = 12.78
>>> int
12.78
```

Damit ist die Funktion `int()` nicht mehr verfügbar:

```
>>> int(1.2)
Traceback (most recent call last):
```

```
...
TypeError: 'float' object is not callable
```

Die Fehlermeldung ist so zu lesen: Der Name `int` ist nun einer Gleitkommazahl (Objekt vom Typ `float`) zugeordnet und kann nicht (wie eine Funktion) aufgerufen werden.

1.6.4 Neue Namen für Funktionen und andere Objekte

Mittels einer Zuweisung können Sie einer Funktion einen neuen (zusätzlichen) Namen geben:

```
>>> runden = round
```

Nun haben Sie für die Funktion `round()` einen zweiten (deutschen) Namen eingeführt. Probieren Sie aus:

```
>>> runden(1.23)
1
```

Solche Zuweisungen kann man sich als Benennungen vorstellen. Es ist so, als ob Sie einem Freund einen Spitznamen geben. Damit verschwindet der richtige Name nicht. Aber der Spitzname ist eine zweite Möglichkeit, die Person anzusprechen.

1.6.5 Erweiterte Zuweisungen

Eine erweiterte Zuweisung ist eine Kombination aus einer Operation und einer Zuweisung. Sie bewirkt die Änderung des Wertes einer Variablen. Der Operator einer erweiterten Zuweisung endet mit einem Gleichheitszeichen: `+=`, `-=`, `*=`, `/=`, `//=`, `**=`.

Die Anweisung

```
>>> x += 2
```

liest man so: »Der Wert von `x` wird um `2` erhöht.« Diese Anweisung hat die gleiche Wirkung wie

```
>>> x = x + 2
```

Bei einer erweiterten Zuweisung wird der aktuelle Wert der Variablen als erster Operand gewählt. Der zweite Operand ist der Wert des Ausdrucks, der hinter dem Gleichheitszeichen steht. Auf beide Werte wird die Operation angewendet und das Ergebnis der Variablen vor dem Gleichheitszeichen zugewiesen.

Weitere Beispiele:

```
>>> a = 2
>>> a *= 3
>>> a
6
>>> a //= 2
>>> a
3
>>> a **= 3
>>> a
27
```

Inkrementierungen und Dekrementierungen (wie z. B. x++, x-- bei Java oder C) gibt es bei Python übrigens nicht.

1.7 Mit Python-Befehlen Geräte steuern

Bisher ging es um die reine Programmierung mit Python. In diesem Abschnitt soll nun erstmals die Elektronik des Raspberry Pi angesprochen werden. Mit Python-Programmen können Sie elektrische Geräte steuern. Wir fangen mit dem Einfachsten an und schalten eine LED über Python-Befehle in der Kommandozeile an und aus. Bevor es an die Programmierung geht, müssen Sie eine kleine Schaltung aufbauen und an den GPIO anschließen. Zusätzliche Informationen zu den Hardware-Grundlagen finden Sie im Anhang B.

1.7.1 Projekt: Eine LED ein- und ausschalten

Für dieses Projekt benötigen Sie

- eine Steckplatine (*Breadboard*)
- Jumperkabel (female-male)
- eine LED
- Kabelbrücken für die Steckplatine
- einen elektrischen Widerstand mit R = 130 Ohm

Abbildung 1.8 zeigt, wie die Schaltung in der Realität aussieht. Die (+)-Leiterbahn (rot) der Steckplatine ist mit einem Jumperkabel mit Pin 1 des GPIO verbunden. Pin 10 ist über ein Jumperkabel mit einem Ende des Widerstands verbunden. Auf der Steckplatine sind Widerstand und die LED in Reihe geschaltet.

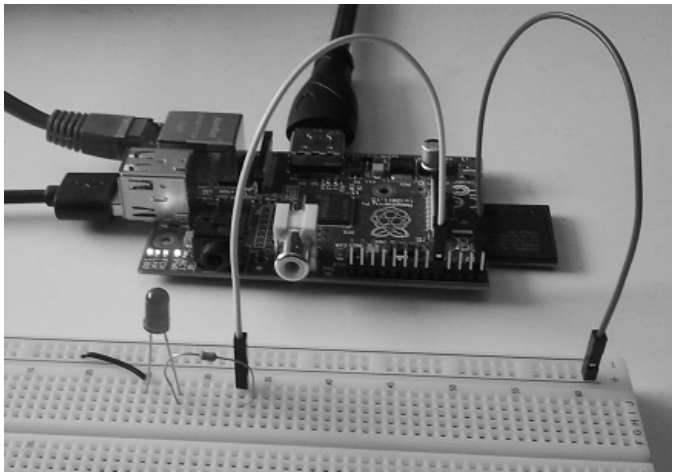

Abb. 1.8: Die Steckplatine trägt die Schaltung und ist über Jumperkabel mit dem GPIO verbunden.

Besser zu lesen ist meist ein Schaltdiagramm mit Schaltsymbolen wie in Abbildung 1.9.

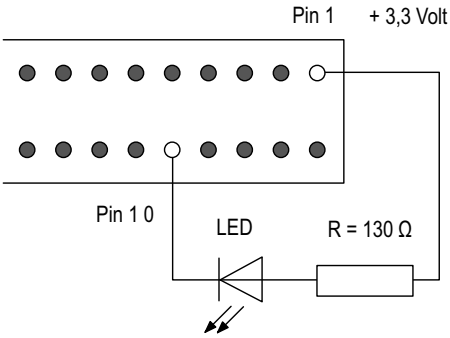

Abb. 1.9: Schaltdiagramm für den Anschluss einer LED an den GPIO des Raspberry Pi

Wenn Sie wie bei diesem Projekt elektronische Bauteile über Jumperkabel an den GPIO anschließen, sollten Sie genauestens aufpassen, nicht einen der 5-Volt-Pins zu berühren. Ein versehentlicher Kontakt kann die Elektronik Ihres RPi zerstören.

Eine Leuchtdiode (oder LED von engl. *light-emitting diode*) ist ein Halbleiterbauteil, das Licht aussendet und sich elektrisch wie eine Diode verhält. Das bedeutet insbesondere, dass LEDs nur in einer Richtung den Strom leiten. Sie wirken wie ein Ventil für Elektronen. Das Schaltsymbol ähnelt ein bisschen einem Pfeil und deutet die technische Fließrichtung des Stroms von Plus nach Minus an. Bei einer

neuen LED ist die Anode (+) etwas länger als die Kathode (-). Leuchtdioden haben nur einen äußerst geringen Widerstand und vertragen nur eine begrenzte Stromstärke. Bei den normalen LEDs, die Sie von Anzeigeleuchten in Elektrogeräten und Armaturenbrettern kennen und die wir auch für Schaltungen mit dem RPi verwenden, beträgt die zulässige Maximalstromstärke etwa 20 mA (Milliampere). Um die Stromstärke zu begrenzen, muss ein Widerstand in den Stromkreis geschaltet werden. Ein guter Widerstandswert für unsere Zwecke ist R = 130 Ohm. Das können Sie selbst nachrechnen: Nach dem ohmschen Gesetz gilt immer die Beziehung R = U/I. Nun haben wir an den steuerbaren Pins des GPIO eine Spannung von 3,3 Volt. An einer Leuchtdiode ist meist eine Spannung von 2 Volt. Damit liegt am Widerstand eine Spannung von 1,3 Volt an. Wenn wir den Strom auf 10 mA begrenzen wollen, ergibt sich für den Vorwiderstand: R = 1,3V/0,01A = 130 Ω.

1.7.2 Das Modul RPI.GPIO

Kommen wir nun zur Programmierung. Öffnen Sie eine Python-Shell (IDLE).

```
sudo idle3
```

Versuchen Sie zunächst, das Modul `RPi.GPIO` zu importieren. Achten Sie dabei auf das kleine `i` bei `RPi`.

```
>>> from RPi import GPIO
```

Falls es nun eine Fehlermeldung gibt, ist das Modul noch nicht installiert. Öffnen Sie dann ein LX-Fenster und geben Sie das folgende Kommando ein:

```
sudo apt-get install python-rpi.gpio
```

Nun geht es wieder in der Python-Shell weiter. Stellen Sie den Modus der Pin-Nummerierung ein:

```
>>> GPIO.setmode(GPIO.BOARD)
```

Machen Sie Pin 10 zum Ausgabekanal.

```
>>> GPIO.setup(10, GPIO.OUT)
```

Schalten Sie die LED aus, indem Sie den Ausgang mit dem Wahrheitswert `True` belegen. Dann hat der Pin 10 den Spannungspegel von +3,3 V. Beide Anschlüsse der LED haben dann das gleiche elektrische Potenzial (+3,3 V gegenüber der Masse). Das heißt, an der LED liegt keine Spannung an und es fließt auch kein Strom.

```
>>> GPIO.output(10, True)
```

Wenn Sie den Wahrheitswert `False` an den Ausgang legen, ist am Pin 10 der Spannungspegel 0 V. Nun gibt es an der LED einen Spannungsabfall, es fließt Strom und die LED leuchtet.

```
>>> GPIO.output(10, False)
```

1.7.3 Steuern mit Relais

Die Ausgänge des GPIO können Sie mit maximal 50 mA belasten. Für die Steuerung größerer elektrischer Geräte (Motoren, Lampen) brauchen Sie Relais. Es gibt preiswerte Module (etwa ab 5 Euro) mit mehreren Relais, die Sie verwenden können. Oft werden sie für den Arduino (`http://www.arduino.cc/`) angeboten und sind deshalb auf TTL (Transistor-Transistor-Logik) zugeschnitten. Das heißt, diese Relais werden mit 5 Volt angesteuert. Zum Glück funktionieren sie auch mit 3,3 Volt, also der Spannung, die an einem programmierbaren GPIO-Ausgang anliegt, wenn er im Zustand `True` ist.

Abbildung 1.10 zeigt ein populäres Modul mit vier Relais. Eingekreist sind rechts die sechs Eingänge (GND, IN1, IN2, IN3, IN4, VCC). GND (Masse) verbinden Sie mit Pin 6 (GND) und VCC mit Pin 2 (+5 V) des GPIO. Die anderen Pins werden an programmierbare Pins des GPIO angeschlossen.

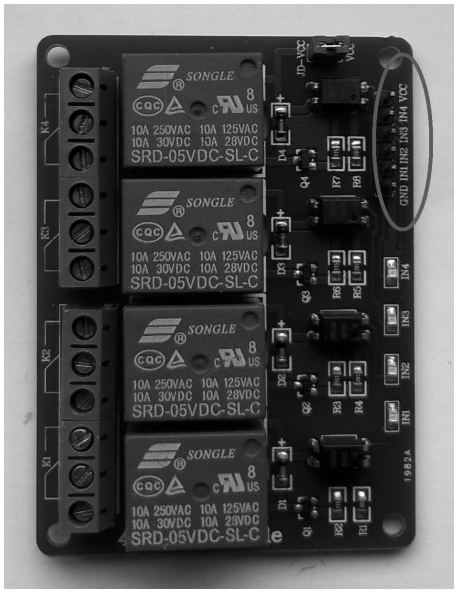

Abb. 1.10: 5-Volt-Relais-Modul für den Arduino. Es funktioniert auch mit dem RPi.

Die Relais auf dieser Platine sind vier Blöcke. Grundsätzlich ist ein Relais nichts weiter als ein elektrisch gesteuerter Schalter. Es kann einen Stromkreis öffnen und schließen. Links neben jedem Relais sind drei Schraubklemmen. Abbildung 1.11 zeigt die Arbeitsweise des Umschaltrelais. Wenn der Steuereingang des Relais im Zustand `True` ist (positiver Spannungspegel), wird der mittlere Anschluss mit dem unteren verbunden. Das ist übrigens der Grundzustand, in dem sich das Relais befindet, wenn am Eingang nichts angeschlossen ist. Meist ist an dem Relais eine kleine Grafik, die den Grundzustand anzeigt. Wenn der Steuereingang den Zustand `False` hat (0 V), wird der mittlere Anschluss mit dem oberen verbunden. Mit dem Umschaltrelais können Sie also zwei Stromkreise gleichzeitig steuern. Wenn der eine geöffnet ist, ist der andere geschlossen.

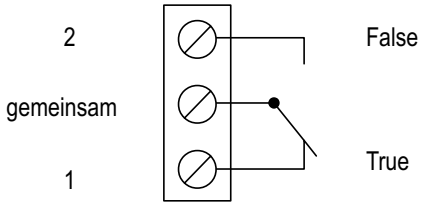

Abb. 1.11: Schaltung eines Umschaltrelais

Auf der Oberseite der Relais finden Sie Angaben zur maximalen Belastbarkeit. Bei den Relais in Abbildung 1.11 sind 10 A und 250 V zugelassen.

1.7.4 Projekt: Eine Taschenlampe an- und ausschalten

In diesem Abschnitt wird erklärt, wie man mit Alltagsmaterialien (Kabel, Pappe, Alufolie, Kreppklebeband und einem Gummiband) eine Taschenlampe mit einem Relais steuern kann. Es gibt natürlich verschiedene Typen von Taschenlampen. Diese Anleitung bezieht sich auf eine Taschenlampe mit Metallgehäuse. Der Schalter sitzt auf der Rückseite. Wenn Sie eine andere Art von Taschenlampe verwenden, müssen Sie etwas anderes erfinden. Abbildung 1.12 zeigt das Endergebnis. Sie erkennen, dass an das erste Relais zwei Kabel angeschlossen sind, die zur Taschenlampe führen.

Und so gehen Sie vor: Schrauben Sie die Verschlusskappe des Batteriefachs der Taschenlampe ab. Im Inneren sehen Sie einen Pol des Batteriesets. Wenn man ihn mit der Metallhülle der Taschenlampe verbindet, leuchten die LEDs.

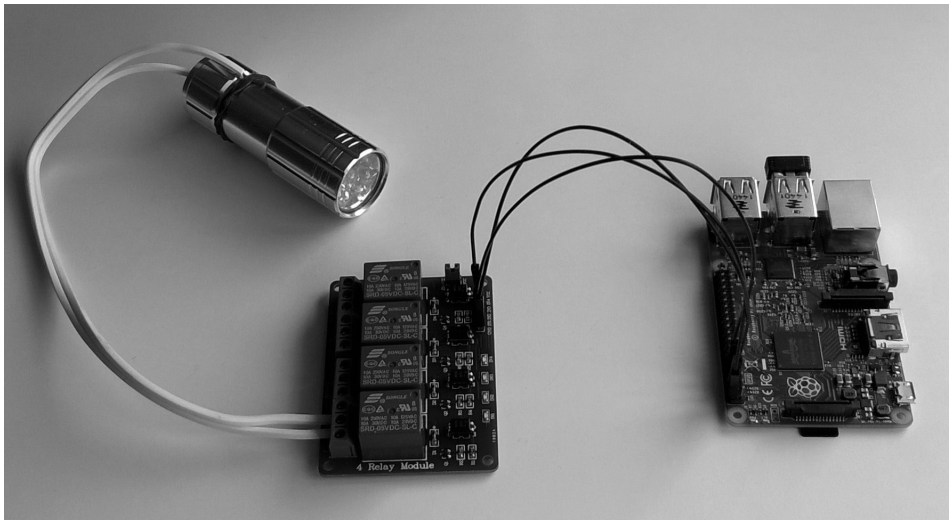

Abb. 1.12: Steuerung einer Taschenlampe mit einem Relais

Sie brauchen ein Kabel mit zwei Adern. Entfernen Sie am Ende einer Ader etwa 8 cm Isolierung. Wickeln Sie um die Mitte des entisolierten Bereichs etwas Alufolie, sodass dort eine Verdickung entsteht (Abbildung 1.13, linkes Bild).

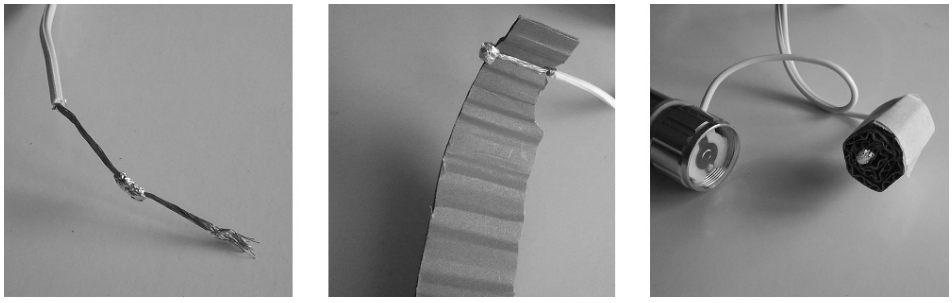

Abb. 1.13: Anschluss eines Steuerkabels an die Taschenlampe

Schneiden Sie aus Wellpappe einen etwa 3 cm schmalen Streifen aus und wickeln Sie ihn an einer Seite um das entisolierte Kabelende, sodass die Verdickung aus Alufolie an einer Kante der Pappe hervorragt (Abbildung 1.13, mittleres Bild). Sorgen Sie dafür, dass die Metalllitze fest verdrillt ist.

Rollen Sie die Pappe zu einem Zylinder zusammen. An der einen Seite ragt in der Mitte die Verdickung aus Aluminiumfolie hervor, auf der anderen Seite kommt das Kabel heraus (Abbildung 1.13, rechtes Bild).

Mit dem Gummiband befestigen Sie die zweite Ader an dem Metallgehäuse der Taschenlampe. Eventuell müssen Sie das Metall an der Berührstelle etwas ankratzen, damit es besser leitet.

Dann schließen Sie die beiden anderen Enden des Kabels wie in Abbildung 1.14 an das erste Relais an. Im Grundzustand sollte der Stromkreis mit der Lampe geöffnet sein. Verbinden Sie mit Jumperkabeln die Eingänge GND, IN1 und VCC des Moduls mit dem GPIO wie in Abbildung 1.14 rechts.

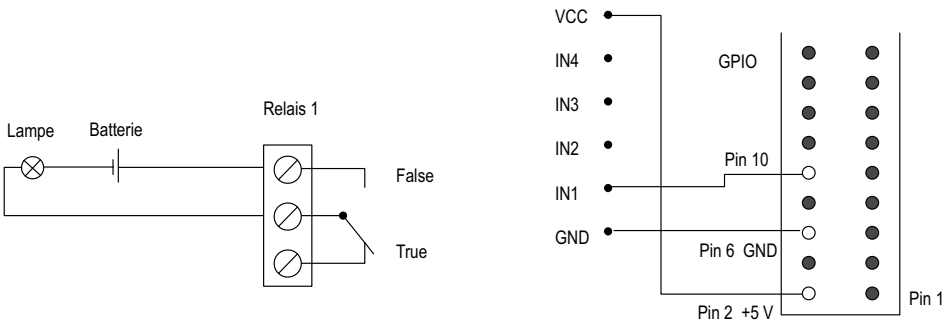

Abb. 1.14: Schaltplan für die Steuerung mit einem Umschaltrelais

Testen Sie Ihren Aufbau mit Python-Befehlen:

```
>>> from RPi import GPIO
>>> GPIO.setmode(GPIO.BOARD)
>>> GPIO.setup(10, GPIO.OUT)  # Pin 10 ist Ausgang
>>> GPIO.output(10, False)    # Lampe einschalten
>>> GPIO.output(10, True)     # Lampe ausschalten
```

1.8 Aufgaben

Aufgabe 1: Formeln

Berechnen Sie die drei handschriftlichen Formeln in der Python-Shell.

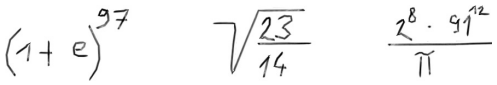

Abb. 1.15: Drei Formeln in mathematischer Schreibweise

Aufgabe 2: Gebäude

Berechnen Sie in der Python-Shell das Volumen des Gebäudes in Abbildung 1.16.

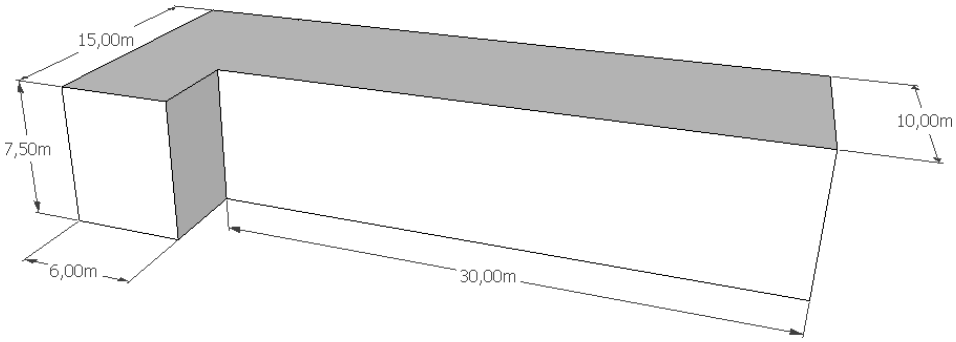

Abb. 1.16: Wohnblock

Aufgabe 3: Zylinder

Berechnen Sie in der Python-Shell das Volumen des Zylinders aus Abbildung 1.17.

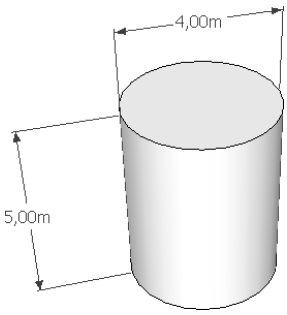

Abb. 1.17: Zylinder

Aufgabe 4: Anweisungen

Welche Ergebnisse liefern folgende Anweisungen?

```
>>> float(9)
>>> int(7/2)
>>> 0b100 + 0b100
>>> bin(2**4)
>>> bin(2**5 + 2**0)
>>> 10*10/10
>>> 2/2
>>> 3**2
>>> 9**0.5
```

Aufgabe 5: Visualisierungen interpretieren

Formulieren Sie zu den drei Bildern in Abbildung 1.18 passende Python-Anweisungen.

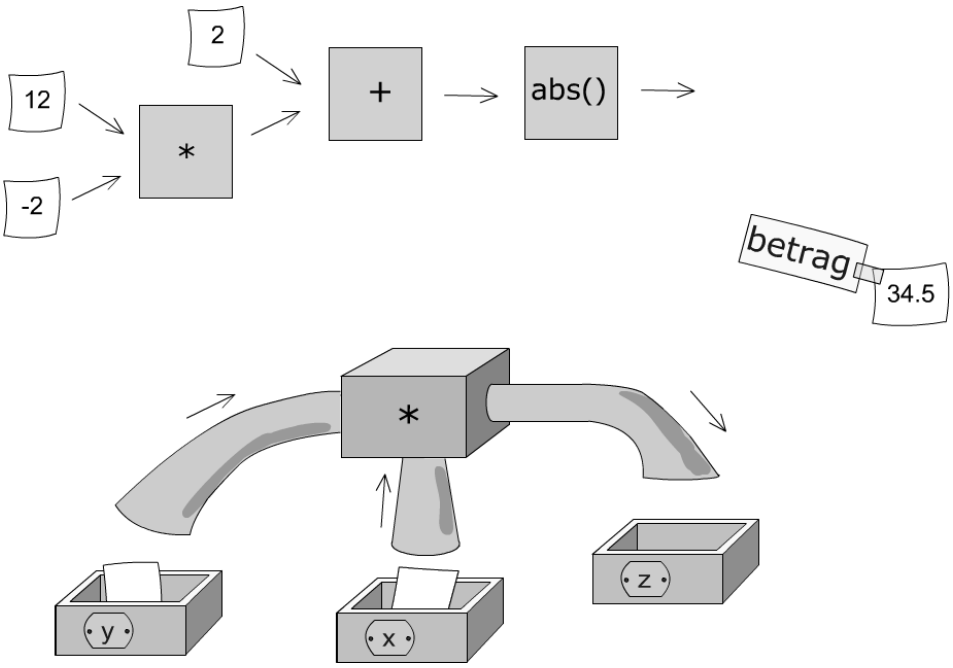

Abb. 1.18: Visualisierungen von Python-Anweisungen

Aufgabe 6

Welche der folgenden Wörter sind nicht als Bezeichner zulässig?

```
name
Erdumfang
ErdUmfang
False
false
round
x(1)
x1
```

1.9 Lösungen

Lösung 1

```
>>> from math import *
>>> (1 + e)**97
2.1047644793816324e+55

>>> sqrt(23/14)
1.2817398889233114

>>> (2**8 * 91**12)/pi
2.6277666738095865e+25
```

Lösung 2

```
>>> (30*10 + 6*15)*7.5
2925.0
```

Lösung 3

```
>>> pi*(4/2)**2 * 5
62.83185307179586
```

Lösung 4

```
>>> float(9)
9.0
```

```
>>> int(7/2)
3
```

Kommentar: 7/2 ergibt 3.5, die `int()`-Funktion lässt die Nachkommastellen weg.

```
>>> 0b100 + 0b100
8
```

Kommentar: Die Summe zweier Binärzahlen wird als Dezimalzahl ausgegeben.

```
>>> bin(2**4)
'0b10000'
bin(2**5 + 2**0)
'0b100001'
```

Kommentar: An den Beispielen erkennt man den Aufbau von Binärzahlen als Summe von Zweierpotenzen.

```
>>> 10*10/10
10.0
>>> 2/2
1.0
>>> 3**2
9
>>> 9**0.5
3.0
```

Kommentar: Division `/` und Potenzieren `**` mit einer Gleitkommazahl liefern immer ein Objekt vom Typ `float`, auch wenn – rein mathematisch – eine ganze Zahl herauskommt.

Lösung 5

```
abs(12*-2 + 2)
betrag = 34.5
z = y * x
```

Lösung 6

Nicht erlaubt sind `False` (Schlüsselwort) und `x(1)` (Klammern sind nicht erlaubt). Dagegen ist das Wort `false` als Name erlaubt, da es mit einem kleinen `f` beginnt und sich somit von dem Schlüsselwort `False` unterscheidet.