

Inhaltsverzeichnis

	Einleitung	15
1	Begegnung mit Python	19
1.1	Was ist Python?	19
1.2	Python-Versionen	20
1.3	IDLE	21
1.3.1	Die Python-Shell	21
1.3.2	Hotkeys	23
1.4	Die Python-Shell als Taschenrechner	24
1.4.1	Operatoren und Terme	24
1.4.2	Zahlen	25
1.4.3	Mathematische Funktionen	29
1.5	Hilfe	34
1.6	Namen und Zuweisungen	35
1.6.1	Zuweisungen für mehrere Variablen	37
1.6.2	Rechnen mit Variablen in der Shell	37
1.6.3	Syntaxregeln für Bezeichner	38
1.6.4	Neue Namen für Funktionen und andere Objekte	39
1.6.5	Erweiterte Zuweisungen	39
1.7	Mit Python-Befehlen Geräte steuern	40
1.7.1	Projekt: Eine LED ein- und ausschalten	40
1.7.2	Das Modul RPI.GPIO	42
1.7.3	Steuern mit Relais	43
1.7.4	Projekt: Eine Taschenlampe an- und ausschalten	44
1.8	Aufgaben	46
1.9	Lösungen	49
2	Python-Skripte	51
2.1	Ein Skript mit IDLE erstellen	51
2.1.1	Ein neues Projekt starten	51
2.1.2	Programmtext eingeben	52
2.1.3	Das Skript ausführen	52
2.1.4	Shortcuts	53

2.2	Programme ausführen	53
2.2.1	Programm in der Konsole starten	53
2.2.2	Anklicken des Programmicons im File-Manager.	55
2.3	Interaktive Programme – das EVA-Prinzip	57
2.3.1	Format mit Bedeutung – Aufbau eines Python-Programmtextes	58
2.3.2	Eingabe – die input()-Funktion	59
2.3.3	Verarbeitung – Umwandeln von Datentypen und Rechnen.	59
2.3.4	Ausgabe – die print()-Funktion	60
2.4	Programmverzweigungen	61
2.4.1	Einfache Bedingungen	62
2.4.2	Wie erkennt man eine gute Melone? Zusammengesetzte Bedingungen	64
2.4.3	Einseitige Verzweigungen und Programmblöcke	65
2.4.4	Haben Sie Ihr Idealgewicht?	66
2.4.5	Eine Besonderheit von Python: Wahrheitswerte für Objekte.	69
2.5	Bedingte Wiederholung – die while-Anweisung.	70
2.5.1	Projekt: Zahlenraten.	71
2.5.2	Have a break! Abbruch einer Schleife.	72
2.6	Projekte mit dem GPIO.	72
2.6.1	Blinklicht	73
2.6.2	Schalter	73
2.6.3	Zähler	75
2.6.4	Grundzustände festlegen und Flankensteuerung	76
2.7	Projekt: Eine Alarmanlage.	77
2.7.1	Aufbau und Arbeitsweise der Alarmanlage	77
2.7.2	Programmierung	78
2.8	Aufgaben	80
2.9	Lösungen	83
3	Kollektionen: Mengen, Listen, Tupel und Dictionaries	87
3.1	Die Typhierarchie	87
3.2	Gemeinsame Operationen für Kollektionen	89
3.3	Kollektionen in Bedingungen.	90
3.3.1	Projekt: Kundenberatung.	91
3.3.2	Projekt: Sichere Kommunikation	91

3.4	Iteration – die for-Anweisung	92
3.4.1	Verwendung von break	93
3.5	Sequenzen	94
3.5.1	Konkatenation und Vervielfältigung	94
3.5.2	Direkter Zugriff auf Elemente – Indizierung	95
3.5.3	Slicing	95
3.5.4	Projekt: Lesbare Zufallspasswörter	96
3.6	Tupel	98
3.7	Zeichenketten (Strings)	99
3.7.1	Strings durch Bytestrings codieren	100
3.7.2	Der Formatierungsoperator %	101
3.8	Listen	102
3.8.1	Listen sind Objekte und empfangen Botschaften	102
3.8.2	Klasse, Typ und Instanz	104
3.8.3	Kopie oder Alias?	104
3.8.4	Listenoperationen	105
3.8.5	Projekt: Zufallsnamen	107
3.8.6	Projekt: Telefonliste	108
3.8.7	Listen durch Comprehensions erzeugen	109
3.9	Zahlen in einer Folge – range()-Funktion	110
3.10	Projekt: Klopfzeichen	111
3.11	Mengen	115
3.11.1	Projekt: Häufigkeit von Buchstaben in einem Text	116
3.12	Projekt: Zufallssounds	117
3.12.1	Wie kommen Töne aus dem Raspberry Pi?	117
3.12.2	Sounds mit PyGame	118
3.12.3	Programmierung	119
3.13	Dictionaries	120
3.13.1	Operationen für Dictionaries	122
3.13.2	Projekt: Morsen	123
3.14	Projekt: Der kürzeste Weg zum Ziel	124
3.15	Aufgaben	127
3.16	Lösungen	129
4	Funktionen	131
4.1	Aufruf von Funktionen	131
4.1.1	Unterschiedliche Anzahl von Argumenten	132
4.1.2	Positionsargumente und Schlüsselwort-Argumente	132
4.1.3	Für Experten: Funktionen als Argumente	133

4.2	Definition von Funktionen	134
4.3	Funktionen in der IDLE-Shell testen	136
4.4	Docstrings.	136
4.5	Veränderliche und unveränderliche Objekte als Parameter	137
4.6	Voreingestellte Parameterwerte	139
4.7	Beliebige Anzahl von Parametern	140
4.8	Die return-Anweisung unter der Lupe	141
4.9	Mehr Sicherheit! Vorbedingungen testen	143
4.10	Namensräume: Global und lokal	145
4.11	Rekursive Funktionen – die Hohe Schule der Algorithmik	147
	4.11.1 Projekt: Rekursive Summe	147
	4.11.2 Projekt: Quicksort.	148
4.12	Experimente zur Rekursion mit der Turtle-Grafik	149
	4.12.1 Turtle-Befehle im interaktiven Modus	149
	4.12.2 Projekt: Eine rekursive Spirale aus Quadraten.	151
	4.12.3 Projekt: Pythagorasbaum	153
	4.12.4 Projekt: Eine Koch-Schneeflocke	155
4.13	Projekt: Der Sierpinski-Teppich	157
4.14	Funktionen per Knopfdruck aufrufen: Callback-Funktionen.	159
	4.14.1 Projekt: Digitaler Türgong	160
	4.14.2 Projekt: Verkehrszählungen – Zählen mit mehreren Knöpfen	161
4.15	Aufgaben	164
4.16	Lösungen	167
5	Fenster für den RPi – Grafische Benutzungsoberflächen	171
5.1	Wie macht man eine Benutzungsoberfläche?	171
5.2	Projekt: Die digitale Lostrommel	172
	5.2.1 Die Gestaltung der Widgets	174
	5.2.2 Das Layout-Management	175
5.3	Bilder auf Widgets	177
	5.3.1 Projekt: Ein visueller Zufallsgenerator	178
	5.3.2 Bilder verarbeiten	179
	5.3.3 Projekt: Schwarzweißmalerei	181
5.4	Projekt: Der Krimiautomat	182
	5.4.1 Texteingabe	182
	5.4.2 Programmierung	184

5.5	Wer die Wahl hat, hat die Qual: Checkbutton und Radiobutton	185
5.5.1	Projekt: Automatische Urlaubsgrüße	186
5.5.2	Projekt: Digitaler Glückskeks	188
5.6	Viele Widgets schnell platziert: Das Grid-Layout	190
5.6.1	Projekt: Rechenquiz	191
5.7	Projekt: Farbmixer	194
5.8	Projekt: Editor mit Pulldown-Menüs	196
5.8.1	Aufbau einer Menüstruktur	197
5.8.2	Programmierung	198
5.9	Aufgaben	200
5.10	Lösungen	202
6	Daten finden, laden und speichern	207
6.1	Dateien	207
6.1.1	Daten speichern	207
6.1.2	Daten laden	208
6.2	Ein Blick hinter die Kulissen: Die SD-Karte	208
6.3	Datenstrukturen haltbar machen mit pickle	211
6.4	Versuch und Irrtum – Mehr Zuverlässigkeit durch try-Anweisungen	212
6.5	Projekt: Karteikasten	212
6.5.1	Der Editor	213
6.5.2	Der Presenter	216
6.6	Benutzungsoberfläche zum Laden und Speichern	219
6.6.1	Dialogboxen	219
6.6.2	Erweiterung des Editors für Karteikarten	221
6.6.3	Erweiterung des Presenters	224
6.7	Daten aus dem Internet	226
6.8	Projekt: Goethe oder Schiller?	227
6.8.1	Methoden der String-Objekte	228
6.8.2	Programmierung	230
6.9	Daten finden mit regulären Ausdrücken	233
6.9.1	Reguläre Ausdrücke	233
6.9.2	Die Funktion findall()	235
6.9.3	Projekt: Staumelder	235
6.9.4	Programmierung	236
6.10	Aufgaben	239
6.11	Lösungen	241

7	Projekte mit Zeitfunktionen	243
7.1	Projekt: Fünf Sekunden stoppen und gewinnen.	243
7.2	Datum und Zeit im Überblick	245
7.3	Projekt: Digitaluhr	246
7.3.1	Woher bekommt der RPi die Zeit?	246
7.3.2	Was ist ein Prozess?	247
7.3.3	Vollbildmodus	249
7.3.4	Event-Verarbeitung	252
7.3.5	Autostart	253
7.4	Projekt: Ein digitaler Bilderrahmen	253
7.4.1	Zugriff auf das Dateisystem: Das Modul os	254
7.4.2	Python Imaging Library (PIL)	255
7.4.3	Die Programmierung	258
7.5	Projekt: Wahrnehmungstest	260
7.5.1	Die Programmierung	261
7.6	Projekt: Stoppuhr mit Gong	264
7.7	Aufgaben	267
7.8	Lösungen	268
8	Objektorientierte Programmierung	273
8.1	Überall Objekte	273
8.2	Klassen und Vererbung bei Python	275
8.2.1	Einführendes Beispiel: Alphabet	276
8.2.2	Qualitätsmerkmal Änderbarkeit	279
8.2.3	Vererbung	280
8.3	Pong revisited.	282
8.3.1	Bau eines Fußschalters	283
8.3.2	Die Klasse Canvas	285
8.3.3	Die Programmierung	289
8.4	Renn, Lola renn!	293
8.4.1	Vorbereitung	294
8.4.2	Struktur des Programms	294
8.4.3	Background	296
8.4.4	Switch	296
8.4.5	Display	298
8.4.6	Clock	298
8.4.7	Die Klasse Runner	299
8.4.8	Controller	300
8.4.9	Module	302

8.5	Aufgaben	304
8.6	Lösungen	307
9	Sensortechnik	315
9.1	Was ist ein digitaler Temperatursensor?	315
9.2	Den DS1820 anschließen	316
9.3	Temperaturdaten lesen	317
9.3.1	Temperaturdaten eines Sensors automatisch auswerten ...	318
9.4	Projekt: Ein digitales Thermometer mit mehreren Sensoren	320
9.4.1	Ein Modul für die Messwerterfassung	321
9.4.2	Die grafische Oberfläche	323
9.4.3	Temperaturdaten per E-Mail senden	324
9.5	Projekt: Ein Temperaturplotter	326
9.5.1	Temperatur-Zeitdiagramme	326
9.5.2	Programmierung	327
9.6	Projekt: Mobile Datenerfassung	330
9.6.1	Experimente mit mobiler Temperaturerfassung	332
9.6.2	Programmierung	332
9.6.3	Wiedergabe der Daten	334
9.7	Spannung messen	334
9.7.1	Das SPI-Protokoll	336
9.7.2	Bitverarbeitung	338
9.7.3	Programmierung	341
9.8	Aufgaben	343
9.9	Lösungen	345
10	Projekte mit der Kamera	355
10.1	Das Kameramodul anschließen	355
10.2	Die Kamerasoftware	357
10.2.1	Einzelbilder	358
10.3	Projekt: Kameraoptionen testen	359
10.4	Projekt: Überwachungskamera – Livebild auf dem Bildschirm ...	361
10.5	Projekt: Bewegung erfassen	363
10.6	Projekt: Gerichtete Bewegungen erfassen	366
10.6.1	Files verarbeiten mit subprocess und StringIO	367
10.6.2	Die Programmierung	368
10.7	Projekt: Birnen oder Tomaten?	373
10.7.1	Magische Methoden – das Überladen von Operatoren	374
10.7.2	Programmierung	377
10.7.3	Weiterentwicklungen	380

10.8	Das Modul picamera	380
10.8.1	Die Klasse PiCamera	381
10.8.2	Projekt: Einen Film aufnehmen	382
10.8.3	Projekt: Fotos per E-Mail verschicken	383
10.9	Was ist los am Autobahnkreuz? Bilder einer Webcam auswerten. . .	385
10.9.1	Webcams im Internet anzapfen	385
10.9.2	Auf einem Foto zeichnen – das Modul ImageDraw	386
10.9.3	Projekt: Verkehrsdichte auf der Autobahn	388
10.10	Randbemerkung: Was darf man? Was soll man?	392
10.11	Aufgabe	393
10.12	Lösung	394
II	Webserver	397
II.1	Der RPi im lokalen Netz	397
II.1.1	WLAN	397
II.1.2	Eine dauerhafte IP-Adresse für den RPi	398
II.1.3	Über SSH auf dem RPi arbeiten	399
II.2	Ein Webserver	399
II.2.1	Den Server starten	400
II.2.2	Die Startseite	401
II.2.3	Den Server testen	401
II.3	Was ist los im Gartenteich?	402
II.3.1	Projekt: Einfache Webcam mit statischer Webseite	402
II.3.2	CGI-Skripte	406
II.3.3	Hilfe, mein CGI-Skript läuft nicht!	409
II.3.4	Interaktive Webseiten	411
II.3.5	Eingabekomponenten in einem HTML-Formular	413
II.3.6	Verarbeitung von Eingaben in einem CGI-Skript	414
II.3.7	Zugriff aus der Ferne	416
II.4	Geräte über das Internet steuern	417
II.4.1	Privilegierte Rechte für ein CGI-Skript	417
II.4.2	Programmierung	418
II.5	Datenbanken	420
II.5.1	Das Modul sqlite3	420
II.5.2	Projekt: Freies Obst	423
II.6	Aufgaben	430
II.7	Lösungen	432

A	Den Raspberry Pi einrichten.	437
A.1	Hardware-Ausstattung	437
A.2	Das Betriebssystem installieren	437
A.2.1	Download der Software	437
A.2.2	Die SD-Karte formatieren	438
A.3	Den Raspberry Pi das erste Mal starten und konfigurieren	439
A.4	Die grafische Oberfläche von Raspbian	440
B	Der GPIO	443
B.1	Der GPIO	443
B.2	Ein Flachbandkabel mit Pfostenverbindern	446
C	Autostart.	449
D	So entstand das Titelbild.	451
	Stichwortverzeichnis	455

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 Wheezy 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 nicht-kommerzielle 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 Ihrem Raspberry Pi (der meist mit RPi abgekürzt wird) sind zwei Versionen von Python installiert:

- Python 2 in der Version Python 2.7
- Python 3 in der Version Python 3.2

Dieses Buch bezieht sich fast ausschließlich 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. Warum eigentlich? Nun, es gibt viele Module für spezielle Anwendungen (z. B. für die Bildbearbeitung und wissenschaftliche Berechnungen), die in Python 2 geschrieben worden sind. Ein Modul ist eine Sammlung von Programmtexten, die man wiederverwenden kann. In vielen Fällen wäre es zu aufwendig (und zu teuer), diese Module umzuschreiben. Wenn jemand ein Projekt plant, das auf Python-2-Modulen aufsetzt, muss er notgedrungen auch für seine eigenen Programmtexte Python 2 verwenden.

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 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 Programmierer.** Das ist eine Art Textverarbeitungsprogramm zum Schreiben von Programmen. Sie starten den Programmierer vom Shell-Fenster aus (FILE|NEW WINDOW).
- **Der Debugger.** Er dient dazu, den Lauf eines Programms zu kontrollieren, um logische Fehler zu finden.

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 (MENU|ENTWICKLUNG|PYTHON 3). 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.

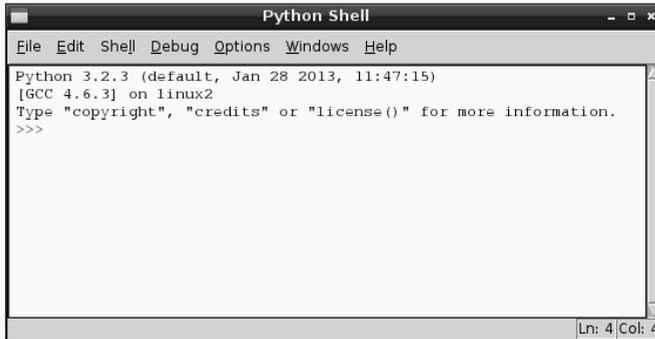


Abb. 1.1: Die Python-Shell der Entwicklungsumgebung IDLE

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 das 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 1.2).

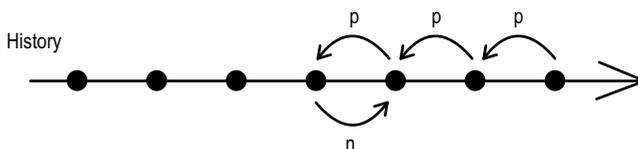


Abb. 1.2: Navigieren in der History mit `[Alt]+[P]` und `[Alt]+[N]`

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
<code>[Alt]+[P]</code>	Previous Command. Die vorige Anweisung in der History (Liste der bisherigen Anweisungen) erscheint hinter dem Prompt.
<code>[Alt]+[N]</code>	Next Command. Die nachfolgende Anweisung in der History erscheint hinter dem Prompt.
<code>[Strg]+[C]</code>	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
620972443101902588551304810097687105537832218918245689182643787308016217315097
07020422858215922309341135893663853254591817
```

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
<code>**</code>	Potenz, $x**y = x^y$
<code>*</code> , <code>/</code> , <code>//</code>	Multiplikation, Division und ganzzahlige Division
<code>%</code>	Modulo-Operation. Der Rest einer ganzzahligen Division.
<code>+</code> , <code>-</code>	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 *Gleichpunktzahl*. »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 Exponential-schreibweise verwendet. Dabei wird die Zahl als Produkt einer rationalen Zahl m (Mantisse) mit einer Zehnerpotenz mit dem Exponenten e dargestellt:

$$z = m \cdot 10^e$$

Beispiele:

$$\begin{aligned} 123000000 &= 123 \cdot 10^6 \\ 0.00012 &= 1.2 \cdot 10^{-4} \end{aligned}$$

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 "<pyshe11#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
<code>abs(x)</code>	Liefert den absoluten (positiven) Wert einer Zahl x .
<code>float(x)</code>	Liefert zu einer Zahl (oder einem anderen Objekt) eine Gleitkommazahl.
<code>int(x)</code>	Liefert zu einer Gleitkommazahl (oder einem anderen Objekt) eine nach unten gerundete ganze Zahl.
<code>max(x0, ..., xn)</code>	Liefert die größte Zahl von $x0, \dots, xn$.
<code>min(x0, ..., xn)</code>	Liefert die kleinste Zahl von $x0, \dots, xn$.
<code>round(x, [n])</code>	Die Zahl x wird auf n Stellen nach dem Komma gerundet und das Ergebnis zurückgegeben.

Tabelle 1.3: Mathematische Standardfunktionen (müssen nicht importiert werden)

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 $^{\circ}$ oder als Bogenmaß angegeben werden. Der Winkel von 360° entspricht dem Bogenmaß 2π .

Mithilfe der Angaben aus Abbildung 1.3 soll die Höhe des Turms berechnet werden.

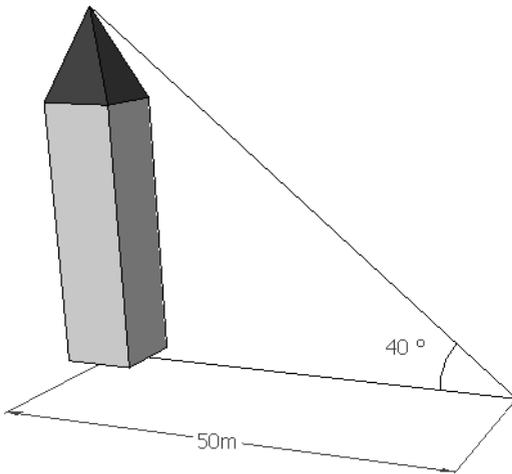


Abb. 1.3: Bestimmung der Höhe eines Turms durch Anpeilen der Turmspitze

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
<code>acos(x)</code> -->	Arcuskosinus von x (Bogenmaß)
<code>asin(x)</code>	Arcussinus von x (Bogenmaß)
<code>atan(x)</code>	Arcustangens von x (Bogenmaß)
<code>cos(x)</code>	Kosinus von x (x ist das Bogenmaß eines Winkels)

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

Funktion/Konstante	Erklärung
<code>cosh(x)</code>	Hyperbolischer Kosinus von x
<code>degrees(x)</code>	Liefert zu einem Winkel, der als Bogenmaß angegeben ist, den Winkel in Grad.
<code>e</code>	Die mathematische Konstante $e \approx 2.7182\dots$
<code>exp(x)</code>	e^x
<code>fabs(x)</code>	Absolutwert der Gleitkommazahl x
<code>floor(x)</code>	Der nach unten gerundete Wert von x
<code>log(x)</code>	Natürlicher Logarithmus von x (Basis e)
<code>log10(x)</code>	Logarithmus von x zur Basis 10
<code>modf(x)</code>	Liefert ein Paar bestehend aus dem Nachkommateil und dem ganzzahligen Teil der Gleitkommazahl x . Beispiel: <code>modf(4.2)</code> ergibt $(0.2, 4.0)$.
<code>radians(x)</code>	Berechnet zum Winkel von x Grad das Bogenmaß.
<code>sin(x)</code>	Sinus von x (x ist das Bogenmaß eines Winkels)
<code>sinh(x)</code>	Hyperbolischer Sinus von x
<code>pi</code>	Die mathematische Konstante $\pi \approx 3.1415\dots$
<code>sqrt(x)</code>	Quadratwurzel von x
<code>tan(x)</code>	Tangens von x (x ist das Bogenmaß eines Winkels)
<code>tanh(x)</code>	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 Sprach-elementen 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 ein neues 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.

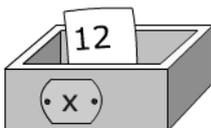


Abb. 1.4: Visualisierung einer Variablen als Behälter mit Inhalt

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 1.5 vor. Die Variable *y* erhält eine Kopie des Inhalts von *x*.

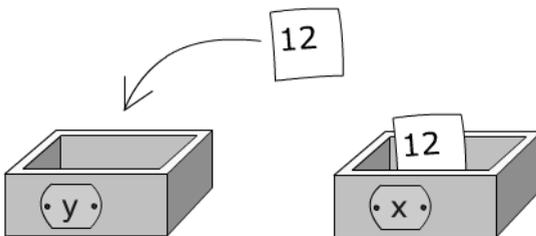


Abb. 1.5: Visualisierung einer Wertübertragung

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.

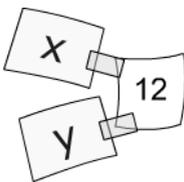


Abb. 1.6: Zuweisen als Hinzufügen eines neuen Namens

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 (das 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 $R = 130 \text{ Ohm}$

Abbildung 1.7 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.

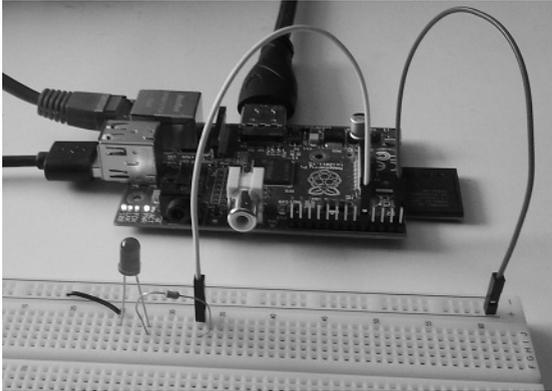


Abb. 1.7: Die Steckplatine trägt die Schaltung und ist über Jumperkabel mit dem GPIO verbunden.

Besser zu lesen ist meist ein Schaltdiagramm mit Schaltsymbolen wie in Abbildung 1.8.

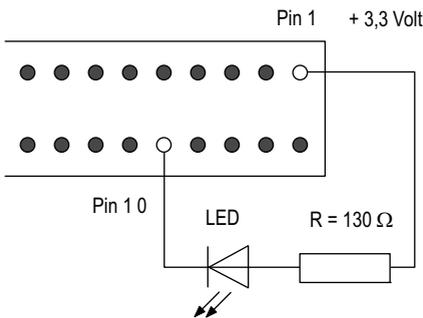


Abb. 1.8: Schaltdiagramm für den Anschluss einer LED an den GPIO des Raspberry Pi

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 insbe-

sondere, 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 $R = 130 \text{ Ohm}$. 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: $R = 1,3\text{V}/0,01\text{A} = 130 \text{ }\Omega$.

1.7.2 Das Modul RPi.GPIO

Kommen wir nun zur Programmierung. Sie können nur mit Root-Rechten (als Administrator) auf die GPIO-Schnittstelle zugreifen. Öffnen Sie ein LX-Fenster und starten Sie die Python-Shell mit dem Befehl

```
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 1.9 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.

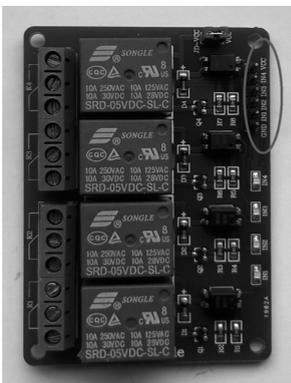


Abb. 1.9: 5-Volt-Relais-Modul für den Arduino. Es funktioniert auch mit dem RPi.

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 1.10 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.

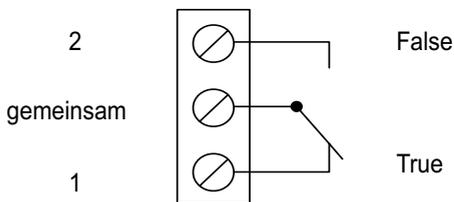


Abb. 1.10: Schaltung eines Umschaltrelais

Auf der Oberseite der Relais finden Sie Angaben zur maximalen Belastbarkeit. Bei den Relais in Abbildung 1.10 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 1.11 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.

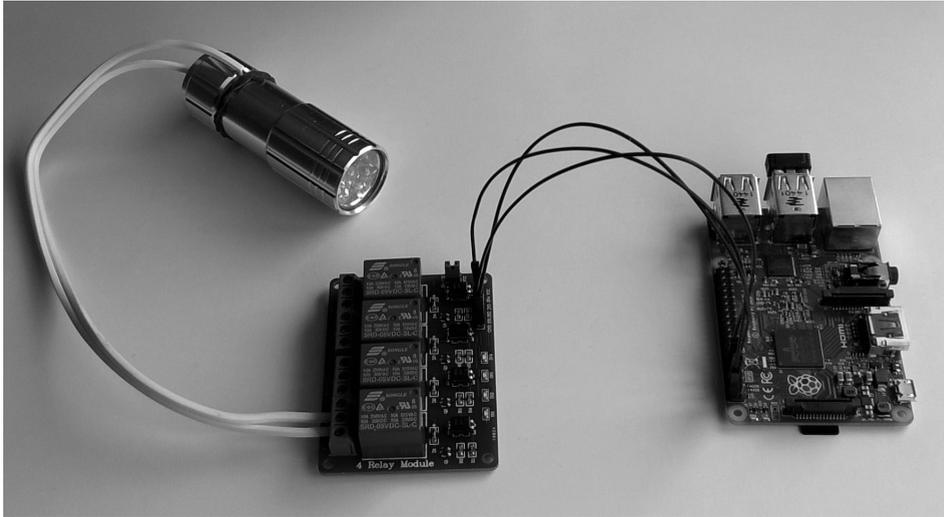


Abb. 1.11: Steuerung einer Taschenlampe mit einem Relais

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 1.12, linkes Bild).



Abb. 1.12: 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 entisolerte Kabelende, sodass die Verdickung aus Alufolie an einer Kante der Pappe hervorragt (Abbildung 1.12, mittleres Bild). Sorgen Sie dafür, dass die Metallitze 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 1.12, 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 1.13 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 1.13 rechts.

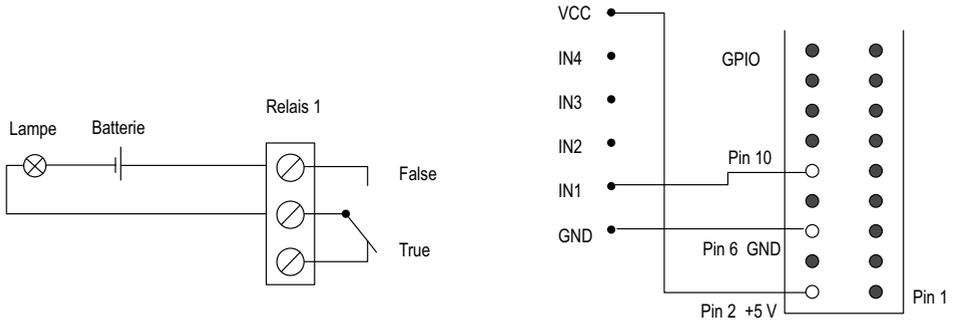


Abb. 1.13: Schaltplan für die Steuerung mit einem Umschaltrelais

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.

$$(1 + e)^{97} \quad \sqrt{\frac{23}{14}} \quad \frac{2^8 \cdot 91^{12}}{\pi}$$

Abb. 1.14: Drei Formeln in mathematischer Schreibweise

Aufgabe 2: Gebäude

Berechnen Sie in der Python-Shell das Volumen des Gebäudes in Abbildung 1.15.

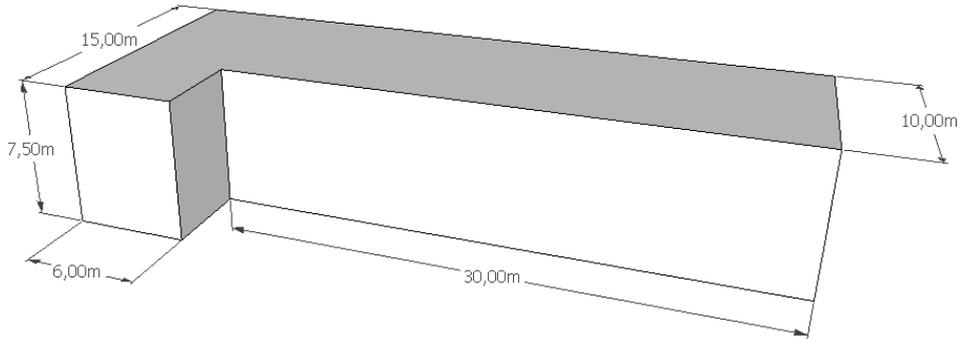


Abb. 1.15: Wohnblock

Aufgabe 3: Zylinder

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

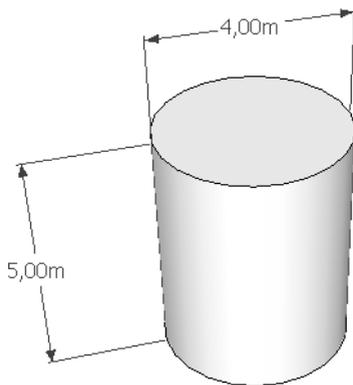


Abb. 1.16: Zylinder

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 1.17 passende Python-Anweisungen.

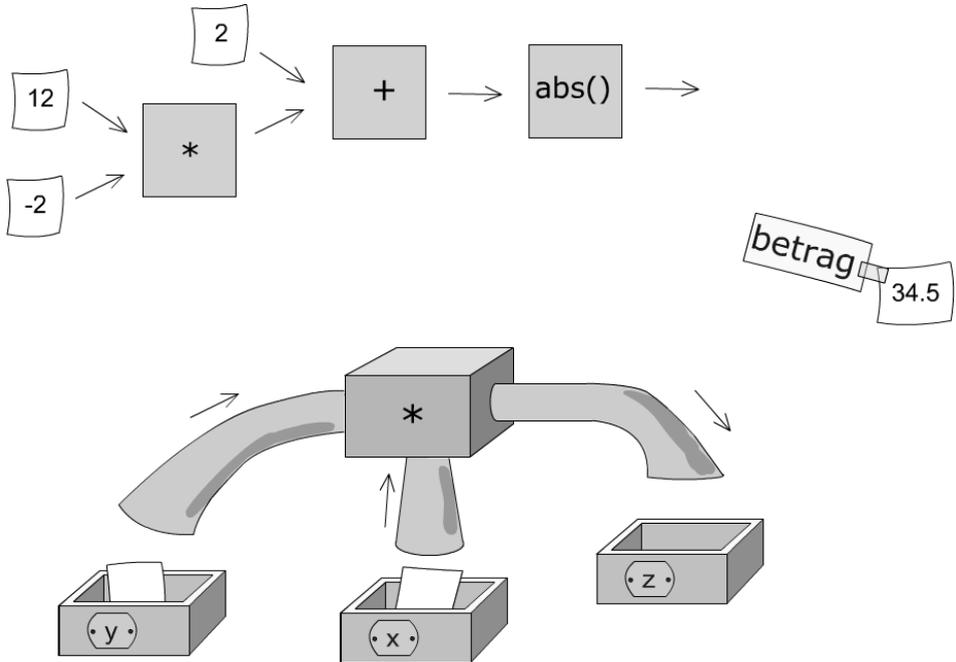


Abb. 1.17: Visualisierungen von Python-Anweisungen

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.

Stichwortverzeichnis

A

action 414
add_cascade() 197
add_checkbutton() 197
add_command() 197
add_radiobutton() 197
add_separator() 197
Aggregat 279
Aggregation 279
Akku 331
Aktueller Parameter 131
Alarmanlage (Projekt) 77
Alias 104
Alphabet (Projekt) 276
anchor 177
Anweisungsblock 134
Application Launch Bar 441
Argument 30, 131
asctime() 245
Ausdruck 24
 regulärer 233
Ausführbar 55
Ausführbar machen 408
Autosimulator 345
Autostart 253, 449

B

Background 296
backward() 151
Bedingung 62
Beenden 406
Benutzungsoberfläche
 grafische 171
Betriebssystem
 installieren 437
Bewegung erfassen (Projekt) 363
Bezeichner 38
 Syntaxregeln 38
Bildrate 383
Binärmodus 211
BINARY(n) 421
bind() 252

Birnen oder Tomaten (Projekt) 373
Bitverarbeitung 338
Blinken (Projekt) 82
Blinklicht 73
Body-Mass-Index 66
Boolescher Ausdruck 81
Botschaft 102
Breadboard 40
break 72, 93
Built-in function 30, 131
Bytestring 100

C

Callback-Funktion 159
Canvas (Klasse) 285
 Methoden 288
Canvas (Objekt) 285
Casting 90
CGI-Skript 400, 406
 ausführbar machen 408
 Debugging 410
 Probleme 409
cgitb 410
Channel 118
chdir () 254
check 413
Checkbutton 185
Clock 298
close() 207
Colorimetrie 380
Comprehension 109
CREATE TABLE 421
crop() 257
CSI-Verbinder 356

D

Dämon 400
Datei 207
Datenbank 420
 relationale 420
def 134
Dialogbox 219

Dictionary 88, 120
 Operationen 122
 Digitaler Bilderrahmen (Projekt) 253
 Digitaler Türgong (Projekt) 160
 Digitales Thermometer (Projekt) 320
 Digitaluhr (Projekt) 246
 dir() 274
 Display 298
 Docstring 136
 Drumscore 112
 DS1820 316
 dump() 211
 Dynamic Domain Name System 416

E

Editor 213
 Editor mit Pulldown-Menüs (Projekt) 196
 Einschalten 439
 Einzelbild 358
 Elektrostatik 355
 elif 68
 else 67
 Entry 182
 Epoche 245
 EPROM 208
 Erweiterte Zuweisung 39
 Escape-Sequenz 99
 EVA-Prinzip 57
 Event 252
 Eventhandler 252
 Event-Sequenz 252
 except 212
 Exponentialschreibweise 28

F

Fallende Flanke 76
 Farbmixer (Projekt) 194
 FieldStorage 414
 File 207
 Modus 207
 File-Manager 55
 fill 177
 findall() 235
 Flachbandkabel 446
 Flanke
 fallende 76
 steigende 76
 Flankensteuerung 76
 Flash-Technik 208
 float 28
 for 92

Format 58
 Formatierungsoperator % 101
 Formular 412
 Fotos per E-Mail verschicken (Projekt) 383
 Freie Literatur 232
 Freies Obst (Projekt) 423
 frozenset 115
 Fünf Sekunden stoppen und gewinnen (Projekt) 243
 Funktion 131
 Aufruf per Knopfdruck 159
 Definition 134
 Kopf 134
 Körper 134
 mathematische 29
 Parameter 131
 Parameterübergabe 137
 voreingestellter Parameterwert 139
 Funktionskopf 134
 Funktionskörper 134
 Fußschalter 283

G

Gerät
 steuern 40
 Gerichtete Bewegungen (Projekt) 366
 getcwd() 254
 Get-Methode 412
 getPixel() 257
 Gleitkommazahl 28
 Glückskeks (Projekt) 188
 gmtime() 245
 Goethe oder Schiller (Projekt) 227
 GPIO 41, 72, 443
 Grafische Benutzungsoberfläche 171
 Grafische Oberfläche
 Raspbian 440
 Graph 124
 Grid-Layout 176, 190
 Optionen 191
 GUI 171

H

Hangman 200
 Hardware
 Ausstattung 437
 Häufigkeit von Buchstaben (Projekt) 116
 Heiße Stelle im Raum 343
 Hexadezimalsystem 27
 hidden 413
 Hilfe 34

Holzklammer 73
 Hotkey siehe Tastenkombination
 HTML 401

I

IDLE 21
 if 65
 ifconfig 397
 Indent Region 66
 Index 88, 95
 Indizierung 95
 input() 59
 INSERT INTO 421
 Instanz 104
 int 26
 Interaktive Webseite 411
 IP-Nummer 397
 Item 89, 121, 285
 Koordinaten 285
 Iteration 92
 Iterierbar 90

J

Jumperkabel 40

K

Kamera 355
 Kameramodul 355
 Kameraoptionen testen (Projekt) 359
 Karteikasten (Projekt) 212
 kill 406
 Klasse 104, 273, 275
 Klassendefinition 273, 275
 Klopfszeichen (Projekt) 111
 Knoten 125
 Koch-Schneeflocke (Projekt) 155
 Kollektion
 Operationen 89
 Kommentar 59
 Komplexe Zahl 29
 Konkatenation 94
 Kontrollvariable 185, 186
 Kopie 104
 Krimiautomat (Projekt) 182
 Küchentablett 330
 Kundenberatung (Projekt) 91
 Kürzester Weg (Projekt) 124

L

Label 174
 Laden 208

Lauf-Simulation (Projekt) 293
 Layout-Management 175
 LED (Projekt) 40
 Lesbare Zufallspasswörter (Projekt) 96
 listdir () 254
 Liste 88, 102
 Operationen 105
 Literal 25, 87
 Literatur
 freie 232
 load() 211
 localtime() 245
 Lostrommel (Projekt) 172
 LXTerminal. 442

M

MAC-Adresse 397
 Magic line 57, 172
 Magische Methode 274, 374
 Mathematische Funktion 29
 Memory-Stick 331
 Menge 88, 115
 Menu
 Optionen der Choices 198
 Messagebox 220
 Messwertaufzeichnung 332
 Messwernerfassung 321
 Methode 103
 magische 274, 374
 MIME 325
 Mobile Datenerfassung (Projekt) 330
 Mobile Stromversorgung 402
 modprobe 317
 Modul 31, 302
 anlegen 302
 Speicherort 304
 Modulo 24
 Morsen (Projekt) 123
 MPC3008 335

N

Name 35
 nobody 417
 nohup 405
 NOOBS 437

O

Oberfläche
 grafische 440
 Objekt 273
 änderbar/nicht-änderbar 275
 Wahrheitswert 69

- Objektorientierte Programmierung 273
- Oktalsystem 27
- omxplayer 383
- Online-Datenbank 423
- open() 207
- Operator 24
 - Plus 275
 - Vergleichsoperator 62
- os 254
- os.path 254

- P**
- Packer 176
- Parameter 30, 131
 - aktueller 131
 - beliebige Anzahl 140
- Parameterliste 134
- Parameterübergabe 137, 139
- Parameterwert
 - voreingestellter 139
- password 413
- Passwort 96
- pcmanfm 449
- pendown() 151
- Pfostenverbinder 446
- PhotoImage 177, 179
 - zoom() 180
- picamera (Modul) 380
- pickle 211
- PIL 256, 386
- PIL.Image
 - crop() 257
 - load() 257
 - resize() 258
 - save() 258
 - size 258
- Pillow 255
- pip (Installationswerkzeug) 256
- Placer 176
- Plotter 327
- Plotter (Projekt) 326
- Plusoperator 275
- Polymorphie 374
- Pong (Projekt) 282
- Port 399
- Positionsargument 132
- Potenz 25
- Potenzimeter 335
- PPM 178
- print() 60
 - Zeilenwechsel unterdrücken 61
- Privilegiertes Recht 417

- Programm
 - starten 53
- Programmierung
 - objektorientierte 273
- Programmverzweigung 61
- Projekt
 - Alarmanlage 77
 - Alphabet 276
 - Bewegung erfassen 363
 - Birnen oder Tomaten 373
 - Blinken 82
 - Digitaler Bilderrahmen 253
 - Digitaler Türgong 160
 - Digitales Thermometer 320
 - Digitaluhr 246
 - Editor mit Pulldown-Menüs 196
 - Farbmixer 194
 - Fotos per E-Mail verschicken 383
 - Freies Obst 423
 - Fünf Sekunden stoppen und gewinnen
 - 243
 - Gerichtete Bewegungen 366
 - Glücksseks 188
 - Goethe oder Schiller 227
 - Häufigkeit von Buchstaben 116
 - Kameraoptionen testen 359
 - Karteikasten 212
 - Klopfzeichen 111
 - Koch-Schneeflocke 155
 - Krimiautomat 182
 - Kundenberatung 91
 - Kürzester Weg 124
 - Lauf-Simulation 293
 - LED 40
 - Lesbare Zufallspasswörter 96
 - Lostrommel 172
 - Mobile Datenerfassung 330
 - Morsen 123
 - Plotter 326
 - Pong 282
 - Pythagorasbaum 153
 - Quicksort 148
 - Quiz 81
 - Rechenquiz 191
 - Rekursive Spirale aus Quadraten 151
 - Rekursive Summe 147
 - Renn, Lola renn 293
 - Schiller 227
 - Schwarzweißmalerei 181
 - Sichere Kommunikation 91
 - Sierpinski-Teppich 157
 - Staumelder 235

Stoppuhr 264
 Taschenlampe an- und ausschalten 44
 Telefonliste 108
 Überwachungskamera 361
 Urlaubsgrüße 186
 Verkehrsdichte auf der Autobahn 388
 Verkehrszählungen 161
 Visueller Zufallsgenerator 178
 Wahrnehmungstest 260
 Webcam 402
 Zahlenraten 71
 Zufallsnamen 107
 Zufallssounds 117
 Prozedur 142
 Prozess 247
 beenden 406
 PSF 19
 Pulldown-Menü 197
 Pull-down-Widerstand 76
 put() 180
 PuTTY 399
 pygame 118, 312
 pygame.mixer 118
 Pythagorasbaum (Projekt) 153
 Python 19
 Shell 21
 Versionen 20
 Python Imaging Library 386
 Python Imaging Library (PIL) 255
 Python Software Foundation 19

Q

Querystring 412
 Quicksort (Projekt) 148
 Quiz (Projekt) 81

R

Radiobutton 185
 range() 110
 Raspbian 440
 raspistill 357
 raspivid 357
 Rasterbild 201
 Rechenquiz (Projekt) 191
 Recht
 privilegiertes 417
 Reed-Schalter 78
 Regulärer Ausdruck 233
 Rekursive Spirale aus Quadraten (Projekt) 151
 Rekursive Summe (Projekt) 147
 Relais 418
 Relationale Datenbank 420

rename () 255
 Renn, Lola renn (Projekt) 293
 Rossum, Guido van 19
 RPI.GPIO 42
 RTC 247
 run module 52

S

Scale 195
 Schalter 73
 Schieberegler 195
 Schiller oder Goethe (Projekt) 227
 Schleife 70
 Schlüssel 122
 Schlüsselwort-Argument 61, 132
 Schwarzweißmalerei (Projekt) 181
 Schwellwertverfahren 181
 SDHC 208
 SD-Karte 208
 formatieren 438
 Seiteneffekt 146
 self 278
 Sensor 315
 Sequenz 88, 94
 Index 95
 Indizierung 95
 Slicing 95
 Vervielfältigung 94
 set 115
 Shebang 56
 Shell 21
 Shell-Fenster 52
 Shiftoperation 339
 Shortcuts siehe Tastenkombination
 showturtle() 151
 Sichere Kommunikation (Projekt) 91
 side 177
 Sierpinski-Teppich (Projekt) 157
 Skript 51
 sleep() 246
 Slicing 95
 Software
 Download 437
 Sound 118
 Spannung messen 334
 speed() 151
 Speichern 207
 SPI-Protokoll 336
 SQL 421
 sqlite3 420
 SSH 399
 start_new_thread() 249

Starten 439
 Staumelder (Projekt) 235
 Steckplatine 40
 Steigende Flanke 76
 Stoppuhr (Projekt) 264
 String 99
 Methoden 228
 StringIO 367
 Stromversorgung
 mobile 402
 submit 413
 subprocess 367
 suid-Bit 418
 Switch 296
 Syntax-Highlighting 52

T

Taschenlampe an- und ausschalten (Projekt)
 44
 Taschenrechner 24
 Tastenkombination 23, 53
 Telefonliste (Projekt) 108
 Temperaturdaten
 E-Mail 324
 Temperatursensor 315
 Temperatur-Zeitdiagramm 326
 Term 24
 Text-Widget 183
 Thread 247
 _thread 248
 time 245
 time() 246
 tkinter 171
 tkinter.filedialog 220
 try 212
 Tupel 89, 98
 Turtle-Grafik 149
 Typ 104
 type() 273
 Typhierarchie 87

U

Überladen 374
 Übersetzen 428
 Überwachungskamera (Projekt) 361
 UML-Klassendiagramm 278
 URL 226
 Urlaubsgrüße (Projekt) 186
 urllib.request 226
 urlopen() 226

V

VARCHAR 421
 Verbose mode 356
 Vererbung 275, 280
 Vergleichsoperator 62
 Verkehrsdichte auf der Autobahn (Projekt)
 388
 Verkehrszählungen (Projekt) 161
 Verzweigung 65
 einseitig 65
 zweiseitig 67
 Visueller Zufallsgenerator (Projekt) 178
 Vollbildmodus 249
 Voreingestellter Parameterwert 139

W

Wahrnehmungstest (Projekt) 260
 Wear leveling 209
 Webcam
 Internet anzapfen 385
 Webcam (Projekt) 402
 Webseite
 interaktive 411
 Webserver 397
 while 70
 Widerstand 40
 Widget 171, 174
 Aussehen 175
 Wiederholung 70
 WiFi-Adapter 397
 WLAN 397
 write() 207

Z

Zahl 25
 komplexe 29
 Zahlenraten (Projekt) 71
 Zahlenschloss 267
 Zähler 75
 Zeichenkette 99
 Zeit 243
 Zeiteinstellung 267
 Zeit-Objekt 245
 Zufallsfunktion 106
 Zufallsnamen (Projekt) 107
 Zufallssounds (Projekt) 117
 Zusicherung 144
 Zuweisung
 erweiterte 39
 Zuweisungsoperator 35