



Michael Weigend

8., erweiterte Auflage

```
'#010b')[2:] for i in data]
.startswith("0001"): i +=1

d):
number representing the numerical value
the multimeter display, if they represent a
herwise -1 is returned. ""

[2][7] + d[2][5] + d[2][4] + \
[1][6] + d[2][6]
[4][7] + d[4][5] + d[4][4] + \
[3][6] + d[4][6]
[6][7] + d[6][5] + d[6][4] + \
[5][6] + d[6][6]
[8][7] + d[8][5] + d[8][4] + \
[7][6] + d[8][6]

] + DIGIT[B] + DIGIT[C] + DIGIT[D])
t position into account
1": n/=10
"1": n/=100
"1": n/= 1000
, M, etc. into account
1": n /= 10**6
"1": n /= 10**6
"1": n *= 1000
= "1" * 1000
= "1" * 10**6
to act
1": -1
```

Python 3

Lernen und professionell anwenden

Das umfassende Praxisbuch

```
t of the me.
Hz, °C) ""
return "F"
": return "Ohms"
": return "A"
```



Hinweis des Verlages zum Urheberrecht und Digitalen Rechtemanagement (DRM)

Der Verlag räumt Ihnen mit dem Kauf des ebooks das Recht ein, die Inhalte im Rahmen des geltenden Urheberrechts zu nutzen. Dieses Werk, einschließlich aller seiner Teile, ist urheberrechtlich geschützt. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung des Verlages unzulässig und strafbar. Dies gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen und Einspeicherung und Verarbeitung in elektronischen Systemen.

Der Verlag schützt seine ebooks vor Missbrauch des Urheberrechts durch ein digitales Rechtemanagement. Bei Kauf im Webshop des Verlages werden die ebooks mit einem nicht sichtbaren digitalen Wasserzeichen individuell pro Nutzer signiert.

Bei Kauf in anderen ebook-Webshops erfolgt die Signatur durch die Shopbetreiber. Angaben zu diesem DRM finden Sie auf den Seiten der jeweiligen Anbieter.

Neuerscheinungen, Praxistipps, Gratiskapitel,
Einblicke in den Verlagsalltag –
gibt es alles bei uns auf Instagram und Facebook



[instagram.com/mitp_verlag](https://www.instagram.com/mitp_verlag)



[facebook.com/mitp.verlag](https://www.facebook.com/mitp.verlag)

Michael Weigend

Python 3

Lernen und professionell anwenden

Das umfassende Praxisbuch

8. Auflage



mitp

Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

ISBN 978-3-7475-0052-1

8. Auflage 2019

www.mitp.de

E-Mail: mitp-verlag@sigloch.de

Telefon: +49 7953 / 7189 - 079

Telefax: +49 7953 / 7189 - 082

© 2019 mitp Verlags GmbH & Co. KG, Frechen

Dieses Werk, einschließlich aller seiner Teile, ist urheberrechtlich geschützt. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung des Verlages unzulässig und strafbar. Dies gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen und die Einspeicherung und Verarbeitung in elektronischen Systemen.

Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.

Lektorat: Sabine Schulz, Lisa Kresse, Johanna Schlösser

Sprachkorrektur: Petra Heubach-Erdmann

Covergestaltung: Christian Kalkert

Coverbild: © Marc AZEMA / stock.adobe.com

Satz: III-satz, Husby, www.drei-satz.de

Inhaltsverzeichnis

Einleitung	21
Warum Python?	21
Python 3	21
An wen wendet sich dieses Buch?	21
Inhalt und Aufbau	22
Hinweise zur Typographie	23
Programmbeispiele	24
1 Grundlagen	25
1.1 Was ist Programmieren?	25
1.2 Hardware und Software.	26
1.3 Programm als Algorithmus.	27
1.4 Syntax und Semantik.	28
1.5 Interpreter und Compiler	28
1.6 Programmierparadigmen	30
1.7 Objektorientierte Programmierung	31
1.7.1 Strukturelle Zerlegung	31
1.7.2 Die Welt als System von Objekten	32
1.7.3 Objekte besitzen Attribute und beherrschen Methoden	33
1.7.4 Objekte sind Instanzen von Klassen	34
1.8 Hintergrund: Geschichte der objektorientierten Programmierung	34
1.9 Aufgaben	35
1.10 Lösungen	36
2 Der Einstieg – Python im interaktiven Modus	37
2.1 Python installieren.	37
2.2 Python im interaktiven Modus	40
2.2.1 Start des Python-Interpreters in einem Konsolenfenster.	40
2.2.2 Die Python-Shell von IDLE.	41
2.2.3 Die ersten Python-Befehle ausprobieren	41
2.2.4 Hotkeys	42
2.3 Objekte	43
2.4 Namen	45
2.5 Hintergrund: Syntax-Regeln für Bezeichner.	45
2.6 Schlüsselwörter	46
2.7 Anweisungen	47
2.7.1 Ausdruckanweisungen	48
2.7.2 Import-Anweisungen	52
2.7.3 Zuweisungen.	53
2.7.4 Erweiterte Zuweisungen.	57

2.7.5	Hintergrund: Dynamische Typisierung	57
2.8	Aufgaben	58
2.9	Lösungen	60
3	Python-Skripte	63
3.1	Ausprobieren, nachmachen, besser machen!	63
3.2	Skripte editieren und ausführen mit IDLE	63
3.3	Ausführen eines Python-Skripts	65
3.4	Kommentare	67
3.5	Die Zeilenstruktur von Python-Programmen	68
3.6	Das EVA-Prinzip	71
3.7	Phasen der Programmentwicklung	73
3.8	Guter Programmierstil	74
3.9	Hintergrund: Die Kunst des Fehlerfindens.	76
3.10	Weitere Entwicklungsumgebungen für Python	78
3.11	Aufgaben	79
3.12	Lösungen	80
4	Standard-Datentypen	83
4.1	Daten als Objekte	83
4.2	Fundamentale Datentypen im Überblick	85
4.3	Typen und Klassen	86
4.4	NoneType	87
4.5	Wahrheitswerte – der Datentyp bool	87
4.6	Ganze Zahlen	88
4.7	Gleitkommazahlen	90
4.8	Komplexe Zahlen	91
4.9	Arithmetische Operatoren für Zahlen	92
4.10	Sequenzen	97
4.10.1	Zeichenketten (Strings)	98
4.10.2	Bytestrings	100
4.10.3	Tupel	101
4.10.4	Liste	102
4.10.5	Bytearray	103
4.10.6	Einige Grundoperationen für Sequenzen.	103
4.10.7	Veränderbare und unveränderbare Sequenzen	105
4.11	Mengen	106
4.12	Dictionaries	107
4.13	Typumwandlungen	107
4.13.1	int()	108
4.13.2	float()	109
4.13.3	complex()	110
4.13.4	bool()	110
4.13.5	str()	110
4.13.6	dict(), list() und tuple()	111

4.14	Aufgaben	111
4.15	Lösungen	114
5	Kontrollstrukturen	117
5.1	Einfache Bedingungen	117
5.1.1	Vergleiche	117
5.1.2	Zugehörigkeit zu einer Menge (in, not in)	121
5.1.3	Beliebige Ausdrücke als Bedingungen	121
5.2	Zusammengesetzte Bedingungen – logische Operatoren	122
5.2.1	Negation (not)	122
5.2.2	Konjunktion (and)	123
5.2.3	Disjunktion (or)	124
5.2.4	Formalisierung von Bedingungen	125
5.2.5	Hinweis zum Programmierstil	126
5.3	Programmverzweigungen (bedingte Anweisungen)	126
5.3.1	Einseitige Verzweigung (if)	127
5.3.2	Zweiseitige Verzweigung (if-else)	127
5.3.3	Mehrfache Fallunterscheidung (elif)	128
5.3.4	Bedingte Ausdrücke	130
5.4	Bedingte Wiederholung (while)	130
5.4.1	Endlosschleifen	131
5.5	Iteration über eine Kollektion (for)	133
5.5.1	Zählschleifen – Verwendung von range()	134
5.5.2	Verschachtelte Iterationen	135
5.5.3	Vertiefung: Iterative Berechnung rekursiver Folgen	137
5.6	Abbruch einer Schleife mit break	137
5.6.1	Abbruch eines Schleifendurchlaufs mit continue	138
5.7	Abfangen von Ausnahmen mit try	139
5.7.1	try...except	140
5.8	Aufgaben	142
5.9	Lösungen	146
6	Funktionen	151
6.1	Aufruf von Funktionen	151
6.2	Definition von Funktionen	154
6.3	Schrittweise Verfeinerung	156
6.4	Ausführung von Funktionen	160
6.4.1	Globale und lokale Namen	160
6.4.2	Seiteneffekte – die global-Anweisung	163
6.4.3	Parameterübergabe	164
6.5	Voreingestellte Parameterwerte	166
6.5.1	Schlüsselwort-Argumente	168
6.6	Funktionen mit beliebiger Anzahl von Parametern	170
6.7	Lokale Funktionen	171
6.8	Rekursive Funktionen	172

6.9	Experimente zur Rekursion mit der Turtle-Grafik	174
6.9.1	Turtle-Befehle im interaktiven Modus	174
6.9.2	Eine rekursive Spirale.	175
6.9.3	Baumstrukturen	177
6.9.4	Künstlicher Blumenkohl – selbstähnliche Bilder.	178
6.10	Rekursive Zahlenfunktionen	180
6.11	Hintergrund: Wie werden rekursive Funktionen ausgeführt?	181
6.11.1	Execution Frames	181
6.11.2	Rekursionstiefe	182
6.12	Funktionen als Objekte.	184
6.12.1	Hintergrund: Typen sind keine Funktionen	185
6.13	Lambda-Formen	185
6.14	Funktionsannotationen: Typen zuordnen	186
6.15	Hinweise zum Programmierstil.	187
6.15.1	Allgemeines.	187
6.15.2	Funktionsnamen.	187
6.15.3	Kommentierte Parameter.	187
6.15.4	Docstrings	188
6.16	Aufgaben	189
6.17	Lösungen	192
7	Sequenzen, Mengen und Generatoren	197
7.1	Gemeinsame Operationen für Sequenzen	197
7.1.1	Zugriff auf Elemente einer Sequenz.	198
7.1.2	Slicing von Sequenzen	199
7.1.3	Auspacken (unpacking)	200
7.2	Vertiefung: Rekursive Funktionen für Sequenzen.	201
7.2.1	Rekursives Summieren	201
7.2.2	Rekursive Suche	201
7.3	Tupel.	203
7.4	Listen	204
7.4.1	Eine Liste erzeugen.	204
7.4.2	Eine Liste verändern.	207
7.4.3	Flache und tiefe Kopien	209
7.4.4	Listen sortieren	210
7.4.5	Binäre Suche in einer sortierten Liste.	212
7.4.6	Zwei Sortierverfahren im Vergleich	213
7.4.7	Modellieren mit Listen – Beispiel: die Charts	217
7.5	Generatoren.	221
7.5.1	Generatorausdrücke	222
7.5.2	Generatorfunktionen	222
7.5.3	Iteratoren.	224
7.5.4	Verwendung von Generatoren.	225
7.6	Mengen	225
7.6.1	Operationen für Mengen	227

7.6.2	Modellieren mit Mengen – Beispiel: Graphen	228
7.7	Aufgaben	231
7.8	Lösungen	233
8	Dictionaries	235
8.1	Operationen für Dictionaries	235
8.2	Wie erstellt man ein Dictionary?	236
8.2.1	Definition mit einem Dictionary-Display	236
8.2.2	Schrittweiser Aufbau eines Dictionarys	238
8.2.3	Ein Dictionary aus anderen Dictionaries zusammensetzen – update()	239
8.3	Zugriff auf Daten in einem Dictionary	239
8.3.1	Vergebliche Zugriffsversuche	239
8.4	Praxisbeispiel: Vokabeltrainer	240
8.5	Typische Fehler	242
8.6	Aufgaben	242
8.7	Lösungen	245
9	Ein- und Ausgabe	249
9.1	Files	249
9.1.1	Die Rolle der Files bei E/A-Operationen	249
9.1.2	Was ist ein File?	250
9.1.3	Ein File-Objekt erzeugen	251
9.1.4	Speichern einer Zeichenkette	252
9.1.5	Laden einer Zeichenkette aus einer Datei	253
9.1.6	Absolute und relative Pfade	253
9.1.7	Zwischenspeichern, ohne zu schließen	255
9.1.8	Zugriff auf Files (lesen und schreiben)	256
9.1.9	Speichern beliebiger Daten auf Files	258
9.2	Mehr Zuverlässigkeit durch try- und with-Anweisungen	259
9.2.1	try...finally	260
9.2.2	with-Anweisungen	261
9.3	Objekte speichern mit pickle	262
9.3.1	Funktionen zum Speichern und Laden	262
9.4	Die Pseudofiles sys.stdin und sys.stdout	264
9.5	Ausgabe von Werten mit der print()-Funktion	265
9.5.1	Anwendung: Ausgabe von Tabellen	266
9.6	Kommandozeilen-Argumente (Optionen)	267
9.7	Aufgaben	270
9.8	Lösungen	272
10	Definition eigener Klassen	277
10.1	Klassen und Objekte	277
10.2	Definition von Klassen	279
10.3	Objekte (Instanzen)	281

10.4	Zugriff auf Attribute – Sichtbarkeit	284
10.4.1	Öffentliche Attribute	284
10.4.2	Private Attribute	285
10.4.3	Properties	287
10.4.4	Dynamische Erzeugung von Attributen	289
10.5	Methoden	289
10.5.1	Polymorphismus – Überladen von Operatoren	290
10.5.2	Vertiefung: Objekte ausführbar machen – die Methode <code>__call__()</code>	294
10.5.3	Statische Methoden	294
10.6	Abstraktion, Verkapselung und Geheimnisprinzip	296
10.7	Vererbung	297
10.7.1	Spezialisierungen	297
10.7.2	Beispiel: Die Klasse Konto – eine Spezialisierung der Klasse Geld	298
10.7.3	Vertiefung: Standardklassen als Basisklassen	301
10.8	Hinweise zum Programmierstil	303
10.8.1	Bezeichner	303
10.8.2	Sichtbarkeit	303
10.8.3	Dokumentation von Klassen	304
10.9	Typische Fehler	305
10.10	Aufgaben	306
10.11	Lösungen	310
11	Klassenbibliotheken in Modulen speichern	315
11.1	Testen einer Klasse in einem lauffähigen Stand-alone-Skript	315
11.2	Module speichern und importieren	317
11.3	Den Zugang zu einem Modul sicherstellen	319
11.4	Programmierstil: Verwendung und Dokumentation von Modulen	321
12	Objektorientiertes Modellieren	323
12.1	Phasen einer objektorientierten Software-Entwicklung	323
12.2	Fallstudie: Modell eines Wörterbuchs	324
12.2.1	OOA: Entwicklung einer Klassenstruktur	324
12.2.2	OOD: Entwurf einer Klassenstruktur für eine Implementierung in Python	328
12.2.3	OOP: Implementierung der Klassenstruktur	330
12.3	Assoziationen zwischen Klassen	334
12.3.1	Reflexive Assoziationen	334
12.3.2	Aggregation	336
12.4	Beispiel: Management eines Musicals	337
12.4.1	OOA	337
12.4.2	OOD	339
12.4.3	OOP	339
12.5	Aufgaben	349
12.6	Lösungen	350

13	Textverarbeitung	355
13.1	Standardmethoden zur Verarbeitung von Zeichenketten	355
	13.1.1 Formatieren	356
	13.1.2 Schreibweise	356
	13.1.3 Tests	357
	13.1.4 Entfernen und Aufspalten	358
	13.1.5 Suchen und Ersetzen	359
13.2	Codierung und Decodierung	359
	13.2.1 Platonische Zeichen und Unicode	359
	13.2.2 Vertiefung: Zeichenketten durch Bytefolgen darstellen	361
13.3	Automatische Textproduktion	363
	13.3.1 Texte mit variablen Teilen – Anwendung der String-Methode format()	363
	13.3.2 Vertiefung: Eine Tabelle erstellen	366
	13.3.3 Mahnbriefe	367
	13.3.4 Textuelle Repräsentation eines Objektes	368
	13.3.5 F-Strings	370
13.4	Analyse von Texten	371
	13.4.1 Chat Bots	371
	13.4.2 Textanalyse mit einfachen Vorkommenstests	372
13.5	Reguläre Ausdrücke	374
	13.5.1 Aufbau eines regulären Ausdrucks	375
	13.5.2 Objekte für reguläre Ausdrücke (RE-Objekte)	378
	13.5.3 Analyse von Strings mit match() und search()	379
	13.5.4 Textpassagen extrahieren mit findall()	379
	13.5.5 Zeichenketten zerlegen mit split()	381
	13.5.6 Teilstrings ersetzen mit sub()	382
	13.5.7 Match-Objekte	382
13.6	Den Computer zum Sprechen bringen – Sprachsynthese	385
	13.6.1 Buchstabieren	387
	13.6.2 Den Klang der Stimme verändern	388
13.7	Aufgaben	391
13.8	Lösungen	394
14	Systemfunktionen	403
14.1	Das Modul sys – die Schnittstelle zum Laufzeitsystem	403
	14.1.1 Informationen über die aktuelle Systemumgebung	404
	14.1.2 Standardeingabe und -ausgabe	405
	14.1.3 Die Objektverwaltung beobachten mit getrefcount()	406
	14.1.4 Ausführung eines Skripts beenden	407
14.2	Das Modul os – die Schnittstelle zum Betriebssystem	407
	14.2.1 Dateien und Verzeichnisse suchen	408
	14.2.2 Hintergrund: Zugriffsrechte abfragen und ändern (Windows und Unix)	409
	14.2.3 Dateien und Verzeichnisse anlegen und modifizieren	411

14.2.4	Merkmale von Dateien und Verzeichnissen abfragen	412
14.2.5	Pfade verarbeiten	413
14.2.6	Hintergrund: Umgebungsvariablen	415
14.2.7	Systematisches Durchlaufen eines Verzeichnisbaumes	416
14.3	Datum und Zeit	418
14.3.1	Funktionen des Moduls time	418
14.3.2	Sekundenformat	419
14.3.3	Zeit-Tupel	420
14.3.4	Zeitstrings	421
14.3.5	Einen Prozess unterbrechen mit sleep()	422
14.4	Zeitberechnungen mit dem Modul datetime	422
14.4.1	Die Klasse datetime	422
14.4.2	Die Zeitzone	424
14.4.3	Die Klasse timedelta	425
14.5	Aufgaben	425
14.6	Lösungen	426
15	Grafische Benutzungsoberflächen mit tkinter	431
15.1	Ein einführendes Beispiel	432
15.2	Einfache Widgets	435
15.3	Die Master-Slave-Hierarchie	436
15.4	Optionen der Widgets	437
15.4.1	Optionen bei der Instanziierung setzen	437
15.4.2	Widget-Optionen nachträglich konfigurieren	438
15.4.3	Fonts	439
15.4.4	Farben	440
15.4.5	Rahmen	440
15.4.6	Die Größe eines Widgets	441
15.4.7	Leerraum um Text	443
15.5	Gemeinsame Methoden der Widgets	444
15.6	Die Klasse Tk	444
15.7	Die Klasse Button	445
15.8	Die Klasse Label	445
15.8.1	Dynamische Konfiguration der Beschriftung	446
15.8.2	Verwendung von Kontrollvariablen	447
15.9	Die Klasse Entry	449
15.10	Die Klasse Radiobutton	451
15.11	Die Klasse Checkbutton	453
15.12	Die Klasse Scale	455
15.13	Die Klasse Frame	457
15.14	Aufgaben	457
15.15	Lösungen	458
16	Layout	463
16.1	Der Packer	463

16.2	Layout-Fehler	465
16.3	Raster-Layout	466
16.4	Vorgehensweise bei der GUI-Entwicklung	470
	16.4.1 Die Benutzungsoberfläche gestalten	473
	16.4.2 Funktionalität hinzufügen	476
16.5	Aufgaben	477
16.6	Lösungen	480
17	Grafik	491
17.1	Die tkinter-Klasse Canvas	491
	17.1.1 Generierung grafischer Elemente – ID, Positionierung und Display-Liste.	492
	17.1.2 Grafische Elemente gestalten.	494
	17.1.3 Visualisieren mit Kreisdiagrammen	496
17.2	Die Klasse PhotoImage	499
	17.2.1 Eine Pixelgrafik erzeugen.	500
	17.2.2 Fotos analysieren und verändern.	502
17.3	Bilder in eine Benutzungsoberfläche einbinden.	505
	17.3.1 Icons auf Schaltflächen.	505
	17.3.2 Hintergrundbilder.	506
	17.3.3 Hintergrund: Das PPM-Format	508
17.4	Die Python Imaging Library (PIL)	509
	17.4.1 Installation eines Moduls mit pip	509
	17.4.2 Mit PIL beliebige Bilddateien einbinden.	510
	17.4.3 Steganografie – Informationen in Bildern verstecken	511
17.5	Aufgaben	513
17.6	Lösungen	514
18	Event-Verarbeitung	519
18.1	Einführendes Beispiel	520
18.2	Event-Sequenzen	522
	18.2.1 Event-Typen.	522
	18.2.2 Qualifizierer für Maus- und Tastatur-Events	522
	18.2.3 Modifizierer	524
18.3	Beispiel: Tastaturereignisse verarbeiten.	524
18.4	Programmierung eines Eventhandlers	526
	18.4.1 Beispiel für eine Event-Auswertung	527
18.5	Bindemethoden	528
18.6	Aufgaben	528
18.7	Lösungen	530
19	Komplexe Benutzungsoberflächen	537
19.1	Text-Widgets.	537
	19.1.1 Methoden der Text-Widgets	538
19.2	Rollbalken (Scrollbars).	540

19.3	Menüs	542
19.3.1	Die Klasse Menu	542
19.3.2	Methoden der Klasse Menu	543
19.4	Texteditor mit Menüleiste und Pulldown-Menü	544
19.5	Dialogboxen	546
19.6	Applikationen mit mehreren Fenstern	550
19.7	Aufgaben	553
19.8	Lösungen	554
20	Threads.	559
20.1	Funktionen in einem Thread ausführen	560
20.2	Thread-Objekte erzeugen – die Klasse Thread	562
20.3	Aufgaben	565
20.4	Lösungen	566
21	Fehler finden und vermeiden.	571
21.1	Testen von Bedingungen	571
21.1.1	Ausnahmen (Exceptions)	571
21.1.2	Testen von Vor- und Nachbedingungen mit assert	572
21.1.3	Vertiefung: Programmabstürze ohne Fehlermeldung.	575
21.2	Debugging-Modus und optimierter Modus	577
21.3	Ausnahmen gezielt auslösen	578
21.4	Selbstdokumentation	579
21.5	Dokumentation eines Programmlaufs mit Log-Dateien	581
21.5.1	Grundfunktionen	581
21.5.2	Beispiel: Logging in der GUI-Programmierung.	582
21.6	Vertiefung: Professionelles Arbeiten mit Logging	583
21.6.1	Logging-Levels.	583
21.6.2	Logger-Objekte	588
21.6.3	Das Format der Logging-Meldungen konfigurieren	588
21.7	Debugging	590
21.8	Aufgabe	591
21.9	Lösung	592
22	Dynamische Webseiten – CGI und WSGI	593
22.1	Wie funktionieren dynamische Webseiten?	593
22.2	Wie spät ist es? Aufbau eines CGI-Skripts	595
22.2.1	Ein einfacher HTTP-Server	599
22.3	Kommunikation über interaktive Webseiten	599
22.3.1	Aufbau eines HTML-Formulars	600
22.3.2	Eingabekomponenten in einem HTML-Formular.	602
22.4	Verarbeitung von Eingabedaten mit FieldStorage	604
22.5	Sonderzeichen handhaben	606
22.6	CGI-Skripte debuggen	607
22.7	Der Apache-Webserver	608
22.7.1	Den Apache-Server installieren	608

22.7.2	CGI-Skripte auf dem Apache-Server	610
22.8	Dynamische Webseiten mit WSGI	610
22.8.1	Einfacher geht's nicht: Ein Stand-alone-WSGI-Webserver mit wsgiref	610
22.9	mod_wsgi	611
22.9.1	Installation	611
22.9.2	Vorbereitung	612
22.9.3	Den Apache-Server konfigurieren	612
22.9.4	Ein WSGI-Skript für den Apache-Server	614
22.9.5	Tipps zum Debuggen	615
22.9.6	Zugriff von einem entfernten Rechner im WLAN	616
22.10	Verarbeitung von Eingabedaten aus Formularen	616
22.11	Objektorientierte WSGI-Skripte – Beispiel: ein Chatroom	619
22.11.1	Die HTML-Seiten	621
22.11.2	Die Klassen für den Chatroom	622
22.11.3	Skript (Teil 2)	623
22.12	WSGI-Skripte mit Cookies	626
22.12.1	Besuche zählen	627
22.13	Aufgabe	629
22.14	Lösung	630
23	Internet-Programmierung	635
23.1	Was ist ein Protokoll?	635
23.2	Übertragung von Dateien mit FTP	636
23.2.1	Das Modul ftplib	636
23.2.2	Navigieren und Downloaden	637
23.2.3	Ein Suchroboter für FTP-Server	639
23.3	Zugriff auf Webseiten mit HTTP und HTTPS	644
23.3.1	Automatische Auswertung von Webseiten	645
23.4	Zugriff auf Ressourcen im Internet über deren URL	647
23.4.1	Webseite herunterladen und verarbeiten	647
23.4.2	Projekt: Wie warm wird es heute?	648
23.4.3	Datei herunterladen und speichern	649
23.4.4	Projekt: Filme herunterladen	649
23.5	E-Mails senden mit SMTP	651
23.6	Aufgaben	653
23.7	Lösungen	655
24	Datenbanken	663
24.1	Was ist ein Datenbanksystem?	663
24.2	Entity-Relationship-Diagramme (ER-Diagramme)	664
24.3	Relationale Datenbanken	665
24.4	Darstellung von Relationen als Listen oder Dictionaries	666
24.5	Das Modul sqlite3	667
24.5.1	Eine Tabelle anlegen	667

24.5.2	Anfragen an eine Datenbank	669
24.5.3	SQL-Anweisungen mit variablen Teilen	670
24.5.4	SQL-Injections	671
24.6	Online-Redaktionssystem mit Datenbankbindung	672
24.6.1	Objektorientierte Analyse (OOA).	674
24.6.2	Objektorientierter Entwurf des Systems (OOD).	674
24.6.3	Hintergrund: Authentifizieren mit MD5-Fingerprints	676
24.6.4	Implementierung des Redaktionssystems mit Python (OOP)	677
24.7	Aufgaben	687
24.8	Lösungen	688
25	Testen und Tuning	691
25.1	Automatisiertes Testen	691
25.2	Testen mit Docstrings – das Modul doctest	691
25.3	Praxisbeispiel: Suche nach dem Wort des Jahres	694
25.4	Klassen testen mit doctest.	701
25.4.1	Wie testet man eine Klasse?.	701
25.4.2	Normalisierte Whitespaces – doctest-Direktiven	702
25.4.3	Ellipsen verwenden.	702
25.4.4	Dictionaries testen	703
25.5	Gestaltung von Testreihen mit unittest.	703
25.5.1	Einführendes Beispiel mit einem Testfall	704
25.5.2	Klassen des Moduls unittest	705
25.5.3	Weiterführendes Beispiel	708
25.6	Tuning	711
25.6.1	Performance-Analyse mit dem Profiler	711
25.6.2	Praxisbeispiel: Auswertung astronomischer Fotografien	713
25.6.3	Performance-Analyse und Tuning	719
25.7	Aufgaben	720
25.8	Lösungen	722
26	XML und JSON	729
26.1	Was ist XML?.	729
26.2	XML-Dokumente.	730
26.3	Ein XML-Dokument als Baum	732
26.4	DOM.	733
26.5	Das Modul xml.dom.minidom.	736
26.5.1	XML-Dokumente und DOM-Objekte	736
26.5.2	Die Basisklasse Node	738
26.5.3	Die Klassen Document, Element und Text.	740
26.6	Attribute von XML-Elementen	742
26.7	Anwendungsbeispiel 1: Eine XML-basierte Klasse	742
26.8	Anwendungsbeispiel 2: Datenkommunikation mit XML	745
26.8.1	Überblick.	746
26.8.2	Das Client-Programm.	747

26.8.3	Das Server-Programm	750
26.9	JSON	754
26.9.1	JSON-Texte decodieren	755
26.9.2	Decodierungsfehler	756
26.9.3	Ein Dictionary als JSON-Objekt speichern: Kompakt oder gut lesbar?	756
26.9.4	Projekt: Verarbeitung von Wetterdaten	759
26.10	Aufgaben	762
26.11	Lösungen	763
27	Modellieren mit Kellern, Schlangen und Graphen	765
27.1	Stack (Keller, Stapel)	765
27.2	Queue (Schlange)	768
27.3	Graphen	769
27.4	Aufgaben	779
27.5	Lösungen	781
28	Benutzungsoberflächen mit Qt	785
28.1	Was bietet PyQt5?	785
28.1.1	PyQt5 erkunden	786
28.2	Wie arbeitet PyQt? Applikation und Fenster	786
28.3	Eine objektorientierte Anwendung mit PyQt5	787
28.4	Ein Webbrowser	788
28.5	Interaktive Widgets	792
28.6	Label – Ausgabe von Text und Bild	793
28.7	Signale	794
28.8	Checkboxen und Radiobuttons	795
28.9	Auswahlliste (ComboBox)	798
28.10	Gemeinsame Operationen der Widgets	800
28.11	Spezielle Methoden eines Fensters	801
28.12	Events	803
28.13	Fonts	804
28.14	Stylesheets	806
28.15	Icons	809
28.16	Messageboxen	809
28.17	Timer	810
28.18	Das Qt-Layout unter der Lupe	812
28.18.1	Absolute Positionierung und Größe	812
28.18.2	Raster-Layout	814
28.18.3	Form-Layout	815
28.19	Browser für jeden Zweck	817
28.19.1	Die Klasse QWebEngineView	817
28.20	Ein Webbrowser mit Filter	818
28.21	Surfen mit Geschichte – der Verlauf einer Sitzung	820
28.22	Aufgaben	822
28.23	Lösungen	823

29	Multimediaanwendungen mit Qt	827
29.1	Kalender und Textfeld – ein digitales Tagebuch	827
	29.1.1 Programmierung	828
29.2	Kamerabilder	833
29.3	Dialoge	835
	29.3.1 Projekt Ansichtskarte	837
29.4	Videoplayer	841
	29.4.1 Ein einfacher Videoplayer	841
	29.4.2 Videoplayer mit Playlist	845
	29.4.3 Regeln zur Änderung der Größe (Size Policy)	848
	29.4.4 Das Dashboard bei Mausbewegungen einblenden	849
29.5	Aufgaben	852
29.6	Lösungen	856
30	Rechnen mit NumPy	865
30.1	NumPy installieren	865
30.2	Arrays erzeugen	865
	30.2.1 Arrays	866
	30.2.2 Matrizen und Vektoren	868
	30.2.3 Zahlenfolgen	868
	30.2.4 Zufallsarrays	869
	30.2.5 Spezielle Arrays	870
30.3	Indizieren	871
30.4	Slicing	872
30.5	Arrays verändern	873
30.6	Arithmetische Operationen	875
30.7	Funktionen, die elementweise ausgeführt werden	876
30.8	Einfache Visualisierung	877
30.9	Matrizenmultiplikation mit dot()	878
30.10	Array-Funktionen und Achsen	879
30.11	Projekt: Diffusion	881
30.12	Vergleiche	884
30.13	Projekt: Wolken am Himmel	884
30.14	Projekt: Wie versteckt man ein Buch in einem Bild?	887
30.15	Datenanalyse mit Histogrammen	890
30.16	Wie funktioniert ein Medianfilter?	893
30.17	Rechnen mit SciPy	896
	30.17.1 Lineare Gleichungssysteme lösen	896
	30.17.2 Integration	898
30.18	Aufgaben	899
30.19	Lösungen	902
31	Messdaten verarbeiten	907
31.1	Messwerte in einem Diagramm darstellen – Matplotlib und tkinter	907
	31.1.1 Basisprojekt	907

31.1.2	Erweiterung: Den letzten Wert löschen	911
31.1.3	Das Aussehen eines Diagramms gestalten	913
31.2	Messwerte aus einem Multimeter lesen und darstellen	916
31.2.1	Vorbereitung	916
31.2.2	Werte auslesen	917
31.2.3	Welche Ziffern zeigt das Display des Multimeters?	920
31.3	Anzeige der Temperatur	924
31.4	Messreihen aufzeichnen	926
31.5	Aufgabe	929
31.6	Lösung	929
32	Parallele Datenverarbeitung	933
32.1	Was sind parallele Programme?	933
32.2	Prozesse starten und abbrechen	934
32.3	Funktionen in eigenen Prozessen starten	935
32.4	Prozesse zusammenführen – join()	937
32.5	Wie können Prozesse Objekte austauschen?	938
32.5.1	Objekte als Argumente übergeben	938
32.5.2	Objekte über eine Pipe senden und empfangen	938
32.5.3	Objekte über eine Queue austauschen	939
32.6	Daten im Pool bearbeiten	940
32.6.1	Mit dem Pool geht's schneller – ein Zeitexperiment	941
32.6.2	Forschen mit Big Data aus dem Internet	942
32.7	Synchronisation	945
32.8	Produzenten und Konsumenten	948
32.8.1	Sprücheklopfer	949
32.9	Aufgaben	951
32.10	Lösungen	952
33	Django	955
33.1	Django aus der Vogelperspektive	955
33.2	Start eines Projekts	956
33.2.1	Den Server starten	958
33.2.2	Startseite und View einrichten	959
33.3	Datenbankanbindung	961
33.4	Modelle erstellen	962
33.5	Modelle aktivieren	963
33.6	In der Python-Shell die Datenbank bearbeiten	967
33.6.1	Objekte durch Aufruf der Klasse erzeugen	967
33.6.2	Auf Attribute eines Objektes zugreifen	968
33.6.3	Objekte finden	969
33.6.4	Objekte erzeugen und Beziehungen herstellen	970
33.6.5	Den Beziehungsmanager nutzen	970
33.6.6	Objekte löschen	971
33.7	Django-Modelle unter der Lupe	971

33.8	Der Manager unter der Lupe – Objekte erzeugen und suchen	973
33.9	Administration	976
33.9.1	Eine Applikation der Website-Verwaltung zugänglich machen	978
33.10	Views einrichten – die Grundstruktur	982
33.10.1	Was sind Views?	982
33.10.2	Funktionen für Views	983
33.10.3	URL-Patterns	984
33.11	View-Funktionen erweitern	985
33.11.1	Startseite	985
33.11.2	Auflistung der Ideen zu einer Frage – question_index	988
33.11.3	Die Templates verbessern: Namen statt expliziter URLs	990
33.12	Interaktive Webseiten – Views mit Formularen	991
33.12.1	Eingabe einer neuen Frage	991
33.12.2	Eingabe einer neuen Idee	996
33.12.3	View-Funktion für das Speichern einer neuen Idee	997
33.12.4	Fertig!	998
33.13	Die nächsten Schritte	998
33.14	Aufgabe	999
33.15	Lösung	1000
A	Anhang	1003
A.1	Zeichencodierung	1003
A.1.1	Codierung von Sonderzeichen in HTML	1003
A.2	Quellen im WWW	1003
A.3	Standardfunktionen und Standardklassen	1004
A.4	Mathematische Funktionen	1006
A.4.1	Das Modul math	1006
A.4.2	Das Modul random	1007
A.5	EBNF-Grammatik	1008
B	Glossar	1013
C	Download der Programmbeispiele	1025
D	Ein Python-Modul veröffentlichen: PyPI	1027
D.1	Bei PyPI und TestPyPI registrieren	1028
D.2	Ein Paket für die Veröffentlichung vorbereiten	1029
D.2.1	Die Programmdatei setup.py	1029
D.2.2	Die Lizenz	1030
D.2.3	Die Datei README.txt	1031
D.2.4	Die Datei __init__.py	1032
D.3	Das Paket auf PyPI veröffentlichen	1032
D.3.1	Das Paket aktualisieren	1033
	Stichwortverzeichnis	1035



Einleitung

Warum Python?

Es gibt triftige Argumente für die Verwendung der Programmiersprache Python.

- Python ist einfach. Man könnte auch sagen minimalistisch. Auf Sprachelemente, die nicht unbedingt notwendig sind, wurde verzichtet. Mit Python kann man kurze Programme schreiben, die viel leisten.
- Python besitzt einen interaktiven Modus. Sie können einzelne Befehle direkt eingeben und ihre Wirkung beobachten. Python unterstützt das Experimentieren und Ausprobieren. Das erleichtert das Erlernen neuer Programmierkonzepte und hilft vor allem Anfängern bei den ersten »Gehversuchen«.
- Dennoch ist Python kein Spielzeug. Zusammen mit vielen Zusatzkomponenten, so genannten Modulen, ist es eine sehr mächtige Programmiersprache.
- Python ist nichtkommerziell. Alle Software, die Sie benötigen, ist kostenlos und für jede Plattform verfügbar.
- Hinter Python steht eine wachsende internationale Community aus Wissenschaftlern und Praktikern, die die Sprache pflegen und weiterentwickeln.

Python 3

Im Jahre 2008 fand in der Python-Welt eine kleine Revolution statt. Python 3 wurde veröffentlicht. Eine neue Version, die mit den Vorgängerversionen 2.X nicht mehr kompatibel ist. Ein Programm, das z.B. in Python 2.5 geschrieben worden ist, läuft (in der Regel) nicht mehr mit einem Python-3-Interpreter. Das ist natürlich schade, war aber notwendig, weil es einige sehr tief gehende Änderungen gab. Doch das neue Python 3 ist noch konsistenter und führt zu schönerem Programmtext als die früheren Versionen.

An wen wendet sich dieses Buch?

Dieses Buch ist für jeden, der die Programmierung mit Python lernen möchte. Besondere Vorkenntnisse werden nicht erwartet. Für die hinteren Kapitel ist es allerdings hilfreich, wenn man sich mit HTML auskennt. Das Buch wendet sich sowohl an Anfänger als auch an Leserinnen und Leser, die bereits mit einer höheren Programmiersprache vertraut sind, und ihr Wissen erweitern und vertiefen wollen. Für Neulinge gibt es zahlreiche Passagen, in denen grundlegende Konzepte anschaulich erklärt werden. Insbesondere das erste Kapitel ist zum überwiegenden Teil eine allgemeine Einführung für diejenigen, die sich bisher noch nie ausführlicher mit der Computertechnik beschäftigt haben. Wenn Sie sich eher zu

den Fortgeschrittenen zählen, dürfen Sie getrost diese Textabschnitte überspringen und sich dem zuwenden, das Sie interessiert.

Auf der anderen Seite enthält das Buch auch Stellen, die eine Herausforderung darstellen. Einige Abschnitte tragen Überschriften, die mit *Hintergrund:* oder *Vertiefung:* beginnen. Sie enthalten Ausblicke und Hintergrundinformationen oder gehen vertiefend auf speziellere Aspekte der jeweiligen Thematik ein, die nicht jeden interessieren.

Generell ist der Theorieanteil dieses Buches gering. Die praktische Arbeit steht im Vordergrund. In der Regel ist es möglich, theoretische Passagen (wie die über formale Grammatiken) zu überspringen, wenn man nun gar nicht damit zurechtkommt. Alle wichtigen Dinge werden zusätzlich auch auf anschauliche Weise erklärt. Und Sie werden erleben, dass beim Nachvollziehen und praktischen Ausprobieren der Programmbeispiele auch zunächst schwierig erscheinende Konzepte verständlich werden. Lassen Sie sich also nicht abschrecken.

Inhalt und Aufbau

Im Zentrum steht die Kunst der Programmentwicklung nach dem objektorientierten Paradigma. Dabei machen wir einen Rundgang durch verschiedene Gebiete der Informatik. Wir werfen einen Blick hinter die Kulissen von Software-Systemen, die Sie als Anwender aus dem Alltag kennen. Wie gestaltet man eine grafische Benutzeroberfläche? Wie funktioniert E-Mail? Wie programmiert man einen Chatroom? Darüber hinaus werden eine Reihe fundamentaler Ideen der Informatik angesprochen. Das Buch orientiert sich an den üblichen Curricula von Universitätskursen zur Einführung in die Programmierung. In vielen Fällen dürfte es deshalb eine sinnvolle Ergänzung zu einem Vorlesungsskript sein.

Dieses Buch ist so angelegt, dass man es von vorne nach hinten lesen kann. Wir fangen mit einfachen Dingen an und nachfolgende Kapitel knüpfen an den vorhergehenden Inhalt an. Idealerweise sollte jeder Begriff bei seiner ersten Verwendung erklärt werden. Doch lässt sich dieses Prinzip nur schwer in Perfektion umsetzen. Manchmal gehen wir von einem intuitiven Vorverständnis aus und erläutern die Begrifflichkeit erst kurz darauf ausführlich.

Im vorderen Teil des Buches finden Sie an verschiedenen Stellen Hinweise zum Programmierstil und zu typischen Fehlern. Am Ende jedes Kapitels gibt es Übungsaufgaben, die in der Regel nach Schwierigkeitsgrad sortiert sind. Einige Programmieraufgaben sind so komplex, dass man sie (insbesondere als Anfänger) eigentlich gar nicht eigenständig lösen kann. Sie sind dann eher als Erweiterung gedacht und es wurde ins Kalkül gezogen, dass Sie »mogeln« und während der Bearbeitung in die Lösung gucken.

Unterkapitel, deren Überschriften mit dem Wort »Vertiefung« beginnen, wenden sich an besonders interessierte Leser und können in der Regel übersprungen werden.

Der vordere Teil des Buches befasst sich mit den grundlegenden Konzepten der Programmierung mit Python. Herausgestellt werden die syntaktischen Besonderheiten gegenüber anderen Programmiersprachen. Sie finden an verschiedenen Stellen Hinweise zum Programmierstil und zu typischen Fehlern. Angesprochen werden unter anderem folgende Punkte:

- Aufbau von Anweisungen in einem Python Programm
- Umgang mit der Standard-Entwicklungsumgebung IDLE

- Standard-Datentypen
- Modellieren mit Datenstrukturen: Tupel, Listen, Dictionaries, Mengen
- Kontrollstrukturen: Wiederholungen, Verzweigungen, Abfangen von Ausnahmen (try ... except)
- Funktionen: Arten von Parametern, Voreinstellungen, Lambda-Ausdrücke, Rekursion, Docstrings
- Ein- und Ausgabe: Dateien, pickle
- Konzepte der Objektorientierung: Klassen, Objekte, Vererbung, statische Methoden, Polymorphie, Properties
- Techniken der objektorientierten Modellierung: Analyse (OOA) und Design (OOD), UML, Objekt- und Klassendiagramme, Assoziationen
- Modularisieren
- Verarbeitung von Zeichenketten: String-Methoden, Codierung und Decodierung, Formatierung, reguläre Ausdrücke, Sprachsynthese, Chat-Bots
- Systemfunktionen: Schnittstelle zum Betriebssystem, Datum und Zeit
- Grundprinzipien der Gestaltung von grafischen Benutzeroberflächen mit tkinter: Widgets, Event-Verarbeitung, Layout, Threads
- Debugging-Techniken

Im hinteren Teil des Buches werden die Kapitel immer spezieller. Hier kommen dann gelegentlich auch Module von Drittanbietern ins Spiel, die nicht zur Standardinstallation von Python gehören (z.B. PIL, PyQt, NumPy). Sie müssen erst heruntergeladen und installiert werden. Zu diesen spezielleren Themen gehören:

- Internet-Programmierung: CGI-Skripte, WSGI, Webserver, E-Mail-Clients
- Datenbanken und XML
- Testen und Performance-Analyse: doctest, unittest
- Benutzeroberflächen für Multimedia-Anwendungen mit PyQt: Video-Player, Webbrowser, Kalender
- Wissenschaftliches Rechnen mit NumPy und SciPy: Arrays, Vektoren und Matrizen, digitale Bildbearbeitung, Datenvisualisierung, lineare Gleichungssysteme, Integralrechnung
- Parallele Datenverarbeitung: Prozesse und Synchronisation, Queues, Pipes, Pools
- Messdaten eines externen digitalen Multimeters erfassen und verarbeiten
- Webentwicklung mit Django.

Hinweise zur Typographie

Achten Sie beim Lesen auf den Schrifttyp. Formale Texte, wie Python-Programmtext, Funktions- und Variablenamen, Operatoren, Grammatik. Regeln, Zahlen und mathematische Ausdrücke, werden in einem Zeichenformat mit fester Breite gesetzt. Beispiele:

```
x = y + 1
print()
```

In solchen formalen Texten tauchen gelegentlich Wörter auf, die kursiv gesetzt sind. Hierbei handelt es sich um Platzhalter, die man nicht Buchstabe für Buchstabe aufschreibt, sondern z.B. durch Zahlen oder andere Zeichenfolgen ersetzt. Beispiel:

```
range(zahl)
```

Hier bezeichnet *zahl* eine (ganze) Zahl. Ein korrekter Aufruf der Funktion `range()` lautet z.B. `range(10)`, während `range(zahl)` zu Problemen führen kann.

In Programmtexten sind wichtige Passagen fett gedruckt, damit man sie schneller finden kann.

Programmbeispiele

Das Buch enthält zahlreiche Programmbeispiele, die zum Ausprobieren, Nachmachen und Weiterentwickeln ermuntern sollen. Sie können alle Skripte und einige zusätzliche Dateien als ZIP-Archiv von der Website des mitp-Verlages herunterladen. Der URL ist:

<http://www.mitp.de/0051>

Klicken Sie im Kasten **DOWNLOADS** auf den Link **PROGRAMMBEISPIELE**.

Beim Design der Beispiele wurde darauf geachtet, dass sie möglichst kurz und übersichtlich sind. Häufig sind die Skripte Spielzeugversionen richtiger Software, die man im Alltag zu sinnvollen Dingen nutzen kann. Sie sind Modelle – etwa so wie Häuser aus Legosteinen Modelle richtiger Häuser sind. Sie sind auf das Wesentliche reduziert und sollen nur bestimmte Aspekte verdeutlichen. Sie genügen deshalb nicht den Qualitätsanforderungen, die man üblicherweise an professionelle Software stellt, aber sie dienen vielleicht als Anregung und Inspiration für eigene Projekte.

Grundlagen

Bitte noch etwas Geduld! Im ersten Kapitel bleibt der Computer noch ausgeschaltet. Hier wird zunächst eine anschauliche Vorstellung von einigen Grundideen der Programmierung vermittelt. Sie helfen, den Rest des Buches besser zu verstehen. Im Mittelpunkt stehen folgende Fragen:

- Was sind Programme und Algorithmen?
- Worin unterscheiden sich Programmierparadigmen?
- Was ist die Philosophie der objektorientierten Programmierung?

1.1 Was ist Programmieren?

Es ist eigentlich ganz einfach: Programmieren ist das Schreiben eines Programms. Nun gibt es den Begriff »Programm« auch in unserer Alltagssprache – fernab von jeder Computertechnik. Sie kennen Fernseh- und Kinoprogramme, planen ein Programm für Ihre Geburtstagsparty, genießen im Urlaub vielleicht Animationsprogramme (sofern Sie nichts Besseres zu tun haben) und lesen als gewissenhafter Staatsbürger vor den Bundestagswahlen Parteiprogramme. In diesen Zusammenhängen versteht man unter einem Programm eigentlich recht unterschiedliche Dinge: Ein Parteiprogramm ist so etwas wie ein strukturiertes Konzept politischer Ziele, ein Kinoprogramm ein Zeitplan für Filmvorstellungen und ein Animationsprogramm ein Ablauf von Unterhaltungsveranstaltungen.

In der Informatik – der Wissenschaft, die hinter der Programmieretechnik steht – ist der Begriff Programm natürlich enger und präziser gefasst. Allerdings gibt es auch hier unterschiedliche Sichtweisen.

Die älteste und bekannteste Definition basiert auf dem Begriff *Algorithmus*. Grob gesprochen ist ein Algorithmus eine Folge von Anweisungen (oder militärisch formuliert: Befehlen), die man ausführen muss, um ein Problem zu lösen. Unter einem Programm versteht man in dieser Sichtweise einen Algorithmus,

- der in einer Sprache geschrieben ist, die auch Maschinen verstehen können (Programmiersprache), und
- der das Verhalten von Maschinen steuert.

Daraus folgt: Wer ein Computerprogramm schreibt, muss zumindest zwei Dinge tun:

- Er oder sie muss einen Algorithmus erfinden, der in irgendeiner Weise nützlich ist und zum Beispiel bei der Lösung eines Problems helfen kann.
- Der Algorithmus muss fehlerfrei in einer Programmiersprache formuliert werden. Man spricht dann von einem Programmtext.

Ziel einer Programmentwicklung ist korrekter Programmtext.

1.2 Hardware und Software

Ein Computer ist eine universelle Maschine, deren Verhalten durch ein Programm bestimmt wird. Ein Computersystem besteht aus Hardware und Software. Ersteres ist das englische Wort für »Eisenwaren« und meint alle Komponenten des Computers, die man anfassen kann – Arbeitsspeicherbausteine, Prozessor, Peripheriespeicher (Festplatte, Diskette, CD), Monitor, Tastatur usw. Software dagegen ist ein Kunstwort, das als Pendant zu Hardware gebildet wurde. Mit Software bezeichnet man die Summe aller Programme, die die Hardware steuern.

Man kann die gesamte Software eines Computers grob in zwei Gruppen aufteilen:

Das *Betriebssystem* regelt den Zugriff auf die Hardware des Computers und verwaltet Daten, die im Rechner gespeichert sind. Es stellt eine Umgebung bereit, in der Benutzer Programme ausführen können. Bekannte Betriebssysteme sind Unix, MS Windows oder Mac OS. Python-Programme laufen unter allen drei genannten Betriebssystemen. Man nennt sie deshalb portabel.

Anwendungs- und Systemsoftware dient dazu, spezifische Probleme zu lösen. Ein Textverarbeitungsprogramm z.B. unterstützt das Erstellen, Verändern und Speichern von Textdokumenten. Anwendungssoftware ist also auf Bedürfnisse des Benutzers ausgerichtet, während das Betriebssystem nur für ein möglichst störungsfreies und effizientes Zusammenspiel der verschiedenen Komponenten des Computersystems sorgt.

Ein Computersystem wird häufig durch ein Schichtenmodell wie in Abbildung 1.1 beschrieben. Die unterste Schicht ist die Computer-Hardware, darüber liegt das Betriebssystem und zuoberst befinden sich schließlich die Anwendungs- und Systemprogramme, die eine Benutzungsschnittstelle enthalten. Nur über diese oberste Software-Schicht kommunizieren Menschen mit einem Computersystem.

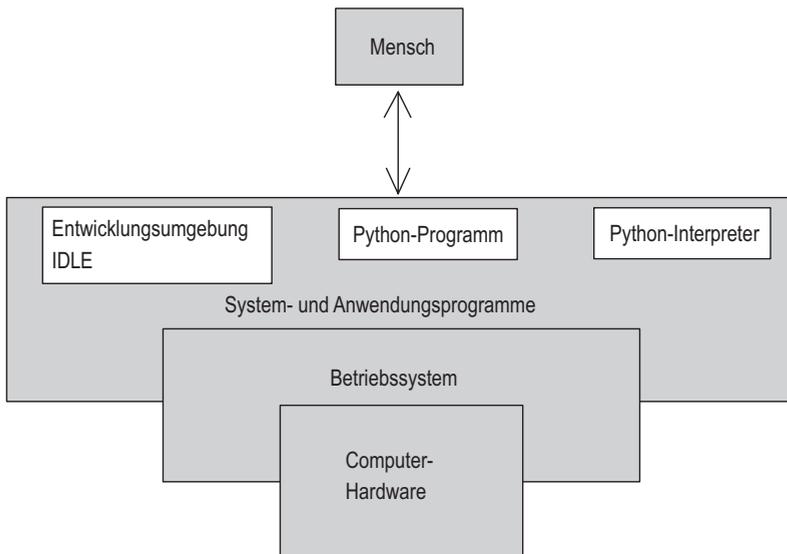


Abb. 1.1: Komponenten eines Computer-Systems

Wenn Sie ein Python-Programm schreiben, entwickeln Sie vor allem Anwendungssoftware. Dabei verwenden Sie eine Systemsoftware, zum Beispiel die integrierte Entwicklungsumgebung IDLE. Ausgeführt wird das Programm mithilfe einer weiteren Systemsoftware, nämlich dem Python-Interpreter. Dieser »liest« den Python-Programmtext Zeile für Zeile und beauftragt das Betriebssystem (eine Schicht tiefer), bestimmte Dinge zu tun – etwa eine Zahl auf den Bildschirm zu schreiben.

1.3 Programm als Algorithmus

Ein Algorithmus ist eine Anleitung zur Lösung einer Aufgabe. Es besteht aus einer Folge von Anweisungen, die so präzise formuliert sind, dass sie auch von einem völlig Unkundigen rein mechanisch ausgeführt werden können. Sie kennen Algorithmen aus dem Alltag:

- Kochrezept
- Anleitung zur Mund-zu-Mund-Beatmung in einer Erste-Hilfe-Fibel
- Gebrauchsanweisung für die Benutzung einer Bohrmaschine

Algorithmus Brathähnchen
nach Martha Pötsch (1901 -1994)

Schalten Sie Ihren Backofen ein und stellen Sie den Temperaturregler auf 200 °C (bei einem Umluftherd nur 180 °C).

Schälen Sie eine Zwiebel und zerschneiden Sie sie in Viertel.

Schneiden Sie eine Tomate ebenfalls in Viertel.

Reiben Sie ein frisches ausgenommenes Hähnchen innen und außen mit insgesamt zwei gestrichenen Teelöffeln Salz ein.

Legen Sie das gesalzene Hähnchen, Zwiebel und Tomate in eine Casserole oder ofenfeste Porzellanschale. Geben Sie eine Tassenfüllung Wasser hinzu.

Schieben Sie das Gefäß mit den Zutaten in den Backofen auf eine mittlere Schiene.

Nach vierzig Minuten wenden Sie das Hähnchen und ergänzen das verdampfte Wasser. Nach weiteren zwanzig Minuten prüfen Sie, ob das Hähnchen goldbraun ist. Ist das nicht der Fall, erhöhen Sie die Temperatur um 20 °C.

Nach weiteren zehn Minuten schalten Sie den Backofen ab und nehmen das Gefäß mit dem köstlich duftenden Hähnchen heraus. Fertig.

Abb. 1.2: Natürlichsprachlich formulierter Algorithmus zur Zubereitung eines Brathähnchens, entwickelt von Martha Pötsch aus Essen

Abbildung 1.2 zeigt einen äußerst effizienten Algorithmus zur Zubereitung eines Brathähnchens (Vorbereitungszeit: eine Minute). Wenn auch das Rezept wirklich sehr gut ist (es stammt von meiner Großmutter), so erkennt man dennoch an diesem Beispiel zwei Schwächen umgangssprachlich formulierter Alltags-Algorithmen:

- Sie beschreiben die Problemlösung meist nicht wirklich vollständig, sondern setzen voraus, dass der Leser, d.h. die den Algorithmus ausführende Instanz, über ein gewisses Allgemeinwissen verfügt und in der Lage ist, »Beschreibungslücken« selbstständig zu füllen. So steht in dem Kochrezept nichts davon, dass man die Backofentür öffnen und schließen muss. Das versteht sich von selbst und wird deshalb weggelassen.
- Sie enthalten ungenaue Formulierungen, die man unterschiedlich interpretieren kann. Was heißt z.B. »goldbraun«?

Auch ein Computerprogramm kann man als Algorithmus auffassen. Denn es »sagt« dem Computer, was er zu tun hat. Damit ein Algorithmus von einem Computer ausgeführt werden kann, muss er in einer Sprache formuliert sein, die der Computer »versteh« – einer Programmiersprache. Im Unterschied zu »natürlichen« Sprachen, wie Deutsch oder Englisch, die sich in einer Art evolutionärem Prozess im Laufe von Jahrhunderten entwickelt haben, sind Programmiersprachen »künstliche« Sprachen. Sie wurden von Fachleuten entwickelt und sind speziell auf die Formulierung von Algorithmen zugeschnitten.

1.4 Syntax und Semantik

Eine Programmiersprache ist – wie jede Sprache – durch Syntax und Semantik definiert. Die *Syntax* legt fest, welche Folgen von Zeichen ein Programmtext in der jeweiligen Sprache ist. Zum Beispiel ist

```
a = 1 ! 2
```

kein gültiger Python-Programmtext, weil die Python-Syntax vorschreibt, dass in einem arithmetischen Ausdruck zwischen zwei Zahlen ein Operator (z.B. +, -, *, /) stehen muss. Das Ausrufungszeichen ! ist aber nach der Python-Syntax kein Operator.

Dagegen ist die Zeichenfolge

```
print("Schweinebraten mit Klößen")
```

ein syntaktisch korrektes Python-Programm. Die Syntax sagt aber nichts darüber aus, welche Wirkung dieses Mini-Programm hat. Die Bedeutung eines Python-Programmtextes wird in der *Semantik* definiert. Bei diesem Beispiel besagt die Semantik, dass auf dem Bildschirm die Zeichenkette Schweinebraten mit Klößen ausgegeben wird.

1.5 Interpreter und Compiler

Python ist eine höhere Programmiersprache. Es ist eine künstliche Sprache für Menschen, die Algorithmen formulieren wollen. Mit einer höheren Programmiersprache lässt sich auf bequeme Weise Programmtext notieren, der leicht durchschaubar und gut verständlich ist. Syntax und Semantik einer höheren Programmiersprache sind auf die Bedürfnisse von Menschen zugeschnitten und nicht auf die technischen Spezifika der Maschine, die das Programm ausführen soll.