

```
d1 = [format(i, '#010b')[2:] for i in data]
```



mitp

```
    if not d1[i].startswith("0001"): i += 1  
    return d1[i:i+14]
```

```
__get_number(self, d):  
    """Returns a float number representing the number  
    of the digits on the multimeter display, if  
    valid number. Otherwise -1 is returned. """  
    try:
```

```
        A = d[1][7] + d[2][7] + d[2][5] + d[2][4] +  
            d[1][5] + d[1][6] + d[2][6]  
        B = d[3][7] + d[4][7] + d[4][5] + d[4][4] +  
            d[3][5] + d[3][6] + d[4][6]  
        C = d[5][7] + d[6][7] + d[6][5] + d[6][4] +  
            d[5][5] + d[5][6] + d[6][6]  
        D = d[7][7] + d[8][7] + d[8][5] + d[8][4] +  
            d[7][5] + d[7][6] + d[8][6]
```

```
        n = int(DIGIT[A] + DIGIT[B] + DIGIT[C] + DIGIT[D])  
        # take the point position into account
```

```
        if d[7][4] == "1": n/=10  
        elif d[5][4] == "1": n/=100  
        elif d[3][4] == "1": n/= 1000  
        # take prefix k, M, etc. into account  
        if d[9][4] == "1": n /= 10**6  
        elif d[9][5] == "1": n /= 10**9
```

```
        elif d[9][6] == "1": n /= 1000  
        elif d[10][4] == "1": n /= 1000  
        elif d[10][5] == "1": n /= 1000  
        # in the case of an error  
        if d[11][4] == "1":  
            return n
```

```
    except:  
        return "Error"
```

```
__get_unit(self, d):
```

```
    """ Returns  
    (A, V, C)  
    if d[11][4] == "1": return "F"  
    elif d[11][5] == "1": return "Ohms"  
    elif d[12][4] == "1": return "A"
```

Michael  
Weigend

9., erweiterte  
Auflage

# Python 3

Lernen und professionell anwenden

**Das umfassende Praxisbuch**

# Inhaltsverzeichnis

	<b>Einleitung</b> .....	23
	Warum Python? .....	23
	Python 3 .....	23
	An wen wendet sich dieses Buch? .....	23
	Inhalt und Aufbau .....	24
	Hinweise zur Typographie .....	25
	Programmbeispiele .....	26
<b>1</b>	<b>Grundlagen</b> .....	27
1.1	Was ist Programmieren? .....	27
1.2	Hardware und Software. ....	28
1.3	Programm als Algorithmus. ....	29
1.4	Syntax und Semantik. ....	30
1.5	Interpreter und Compiler .....	30
1.6	Programmierparadigmen .....	32
1.7	Objektorientierte Programmierung .....	33
	1.7.1 Strukturelle Zerlegung .....	33
	1.7.2 Die Welt als System von Objekten .....	34
	1.7.3 Objekte besitzen Attribute und beherrschen Methoden .....	35
	1.7.4 Objekte sind Instanzen von Klassen .....	36
1.8	Hintergrund: Geschichte der objektorientierten Programmierung .....	36
1.9	Aufgaben .....	37
1.10	Lösungen .....	38
<b>2</b>	<b>Der Einstieg – Python im interaktiven Modus</b> .....	39
2.1	Python installieren. ....	39
2.2	Python im interaktiven Modus .....	42
	2.2.1 Start des Python-Interpreters in einem Konsolenfenster .....	42
	2.2.2 Die IDLE-Shell. ....	43
	2.2.3 Die ersten Python-Befehle ausprobieren .....	43
	2.2.4 Hotkeys .....	44
2.3	Objekte .....	45
2.4	Namen .....	47
2.5	Hintergrund: Syntax-Regeln für Bezeichner .....	47
2.6	Schlüsselwörter .....	48
2.7	Anweisungen .....	49
	2.7.1 Ausdruckanweisungen .....	50
	2.7.2 Import-Anweisungen .....	54
	2.7.3 Zuweisungen .....	55
	2.7.4 Erweiterte Zuweisungen .....	59

2.7.5	Hintergrund: Dynamische Typisierung .....	59
2.8	Aufgaben .....	60
2.9	Lösungen .....	62
<b>3</b>	<b>Python-Skripte</b> .....	<b>65</b>
3.1	Ausprobieren, nachmachen, besser machen! .....	65
3.2	Skripte editieren und ausführen mit IDLE .....	65
3.3	Ausführen eines Python-Skripts .....	67
3.4	Kommentare .....	69
3.5	Die Zeilenstruktur von Python-Programmen .....	70
3.6	Das EVA-Prinzip .....	73
3.7	Phasen der Programmentwicklung .....	75
3.8	Guter Programmierstil .....	76
3.9	Hintergrund: Die Kunst des Fehlerfindens. ....	78
3.10	Weitere Entwicklungsumgebungen für Python .....	80
3.10.1	Thonny – eine Entwicklungsumgebung für Python-Einsteiger ...	80
3.10.2	Python in der Cloud .....	81
3.10.3	Jupyter Notebook und Google Colab. ....	82
3.10.4	Entwicklungsumgebungen für Profis .....	82
3.11	Aufgaben .....	83
3.12	Lösungen .....	84
<b>4</b>	<b>Standard-Datentypen</b> .....	<b>87</b>
4.1	Daten als Objekte .....	87
4.2	Fundamentale Datentypen im Überblick .....	89
4.3	Typen und Klassen .....	90
4.4	NoneType .....	91
4.5	Wahrheitswerte – der Datentyp bool .....	91
4.6	Ganze Zahlen .....	92
4.7	Gleitkommazahlen .....	94
4.8	Komplexe Zahlen .....	95
4.9	Arithmetische Operatoren für Zahlen .....	96
4.10	Sequenzen .....	101
4.10.1	Zeichenketten (Strings) .....	102
4.10.2	Bytestrings .....	104
4.10.3	Tupel .....	105
4.10.4	Liste .....	106
4.10.5	Bytearray .....	107
4.10.6	Einige Grundoperationen für Sequenzen. ....	107
4.10.7	Veränderbare und unveränderbare Sequenzen .....	109
4.11	Mengen .....	110
4.12	Dictionaries .....	111
4.13	Typumwandlungen .....	111
4.13.1	int() .....	113
4.13.2	float() .....	113

4.13.3	complex()	114
4.13.4	bool()	114
4.13.5	str()	114
4.13.6	dict(), list() und tuple()	115
4.14	Aufgaben	115
4.15	Lösungen	118
<b>5</b>	<b>Kontrollstrukturen</b>	<b>123</b>
5.1	Einfache Bedingungen	123
5.1.1	Vergleiche	123
5.1.2	Zugehörigkeit zu einer Menge (in, not in)	127
5.1.3	Beliebige Ausdrücke als Bedingungen	127
5.2	Zusammengesetzte Bedingungen – logische Operatoren	128
5.2.1	Negation (not)	128
5.2.2	Konjunktion (and)	129
5.2.3	Disjunktion (or)	130
5.2.4	Formalisierung von Bedingungen	131
5.2.5	Hinweis zum Programmierstil	132
5.3	Programmverzweigungen (bedingte Anweisungen)	132
5.3.1	Einseitige Verzweigung (if)	133
5.3.2	Zweiseitige Verzweigung (if-else)	133
5.3.3	Mehrfache Fallunterscheidung (elif)	134
5.3.4	Bedingte Ausdrücke	136
5.4	Bedingte Wiederholung (while)	136
5.4.1	Endlosschleifen	137
5.5	Iteration über eine Kollektion (for)	139
5.5.1	Zählschleifen – Verwendung von range()	140
5.5.2	Verschachtelte Iterationen	141
5.5.3	Vertiefung: Iterative Berechnung rekursiver Folgen	143
5.6	Abbruch einer Schleife mit break	143
5.6.1	Abbruch eines Schleifendurchlaufs mit continue	144
5.7	Abfangen von Ausnahmen mit try	145
5.7.1	try...except	146
5.8	Aufgaben	148
5.9	Lösungen	152
<b>6</b>	<b>Funktionen</b>	<b>157</b>
6.1	Aufruf von Funktionen	157
6.2	Definition von Funktionen	160
6.3	Schrittweise Verfeinerung	162
6.4	Ausführung von Funktionen	166
6.4.1	Globale und lokale Namen	166
6.4.2	Seiteneffekte – die global-Anweisung	169
6.4.3	Parameterübergabe	170



6.5	Voreingestellte Parameterwerte . . . . .	172
6.5.1	Schlüsselwort-Argumente . . . . .	174
6.6	Funktionen mit beliebiger Anzahl von Parametern . . . . .	176
6.7	Lokale Funktionen. . . . .	177
6.8	Rekursive Funktionen. . . . .	178
6.9	Experimente zur Rekursion mit der Turtle-Grafik . . . . .	180
6.9.1	Turtle-Befehle im interaktiven Modus . . . . .	180
6.9.2	Eine rekursive Spirale. . . . .	181
6.9.3	Baumstrukturen . . . . .	183
6.9.4	Künstlicher Blumenkohl – selbstähnliche Bilder. . . . .	184
6.10	Rekursive Zahlenfunktionen . . . . .	186
6.11	Hintergrund: Wie werden rekursive Funktionen ausgeführt? . . . . .	187
6.11.1	Execution Frames . . . . .	187
6.11.2	Rekursionstiefe . . . . .	188
6.12	Funktionen als Objekte. . . . .	190
6.12.1	Hintergrund: Typen sind keine Funktionen . . . . .	191
6.13	Lambda-Formen . . . . .	191
6.14	Funktionsannotationen: Typen zuordnen . . . . .	192
6.15	Hinweise zum Programmierstil. . . . .	193
6.15.1	Allgemeines. . . . .	193
6.15.2	Funktionsnamen. . . . .	193
6.15.3	Kommentierte Parameter. . . . .	194
6.15.4	Docstrings . . . . .	194
6.16	Aufgaben . . . . .	196
6.17	Lösungen . . . . .	199
7	<b>Sequenzen, Mengen und Generatoren</b> . . . . .	203
7.1	Gemeinsame Operationen für Sequenzen . . . . .	203
7.1.1	Zugriff auf Elemente einer Sequenz. . . . .	204
7.1.2	Slicing von Sequenzen . . . . .	205
7.1.3	Auspacken (unpacking) . . . . .	206
7.2	Vertiefung: Rekursive Funktionen für Sequenzen . . . . .	207
7.2.1	Rekursives Summieren . . . . .	207
7.2.2	Rekursive Suche . . . . .	207
7.3	Tupel. . . . .	209
7.4	Listen . . . . .	210
7.4.1	Eine Liste erzeugen. . . . .	210
7.4.2	Eine Liste verändern. . . . .	213
7.4.3	Flache und tiefe Kopien . . . . .	215
7.4.4	Listen sortieren . . . . .	216
7.4.5	Binäre Suche in einer sortierten Liste. . . . .	218
7.4.6	Zwei Sortierverfahren im Vergleich . . . . .	219
7.4.7	Modellieren mit Listen – Beispiel: die Charts . . . . .	223
7.5	Generatoren. . . . .	227
7.5.1	Generatorausdrücke . . . . .	228

7.5.2	Generatorfunktionen .....	228
7.5.3	Iteratoren .....	230
7.5.4	Verwendung von Generatoren .....	231
7.6	Mengen .....	231
7.6.1	Operationen für Mengen .....	233
7.6.2	Modellieren mit Mengen – Beispiel: Graphen .....	234
7.7	Aufgaben .....	237
7.8	Lösungen .....	239
<b>8</b>	<b>Dictionaries</b> .....	<b>241</b>
8.1	Operationen für Dictionaries .....	241
8.2	Wie erstellt man ein Dictionary? .....	242
8.2.1	Definition mit einem Dictionary-Display .....	242
8.2.2	Schrittweiser Aufbau eines Dictionarys. ....	244
8.2.3	Ein Dictionary aus anderen Dictionaries zusammensetzen – update() .....	245
8.3	Zugriff auf Daten in einem Dictionary .....	245
8.3.1	Vergebliche Zugriffsversuche .....	245
8.4	Praxisbeispiel: Vokabeltrainer .....	246
8.5	Typische Fehler .....	248
8.6	Aufgaben .....	248
8.7	Lösungen .....	251
<b>9</b>	<b>Ein- und Ausgabe</b> .....	<b>255</b>
9.1	Streams .....	255
9.1.1	Die Rolle der Streams bei E/A-Operationen .....	255
9.1.2	Was ist ein Stream? .....	256
9.1.3	Eine Datei öffnen .....	257
9.1.4	Speichern einer Zeichenkette .....	258
9.1.5	Laden einer Zeichenkette aus einer Datei .....	260
9.1.6	Absolute und relative Pfade .....	260
9.1.7	Zwischenspeichern, ohne zu schließen .....	262
9.1.8	Zugriff auf Streams (lesen und schreiben) .....	263
9.2	Mehr Zuverlässigkeit durch try- und with-Anweisungen .....	265
9.2.1	try...finally .....	266
9.2.2	with-Anweisungen .....	267
9.3	Objekte speichern mit pickle .....	268
9.3.1	Funktionen zum Speichern und Laden .....	268
9.4	Die Streams sys.stdin und sys.stdout .....	270
9.5	Ausgabe von Werten mit der print()-Funktion .....	271
9.5.1	Anwendung: Ausgabe von Tabellen .....	272
9.6	Kommandozeilen-Argumente (Optionen) .....	273
9.6.1	Zugriff auf Optionen .....	274
9.6.2	Beispiel .....	274
9.6.3	Skripte mit Optionen testen .....	275

9.7	Aufgaben .....	275
9.8	Lösungen .....	278
<b>10</b>	<b>Definition eigener Klassen .....</b>	<b>283</b>
10.1	Klassen und Objekte .....	283
10.2	Definition von Klassen .....	285
10.3	Objekte (Instanzen) .....	287
10.4	Zugriff auf Attribute – Sichtbarkeit .....	290
	10.4.1 Öffentliche Attribute .....	290
	10.4.2 Private Attribute .....	291
	10.4.3 Properties .....	293
	10.4.4 Dynamische Erzeugung von Attributen .....	295
10.5	Methoden .....	295
	10.5.1 Polymorphismus – überladen von Operatoren .....	296
	10.5.2 Vertiefung: Objekte ausführbar machen – die Methode __call__() .....	299
	10.5.3 Statische Methoden .....	300
10.6	Abstraktion, Verkapselung und Geheimnisprinzip .....	302
	10.6.1 Abstraktion .....	302
	10.6.2 Verkapselung .....	302
	10.6.3 Geheimnisprinzip .....	302
10.7	Vererbung .....	303
	10.7.1 Spezialisierungen .....	303
	10.7.2 Beispiel: Die Klasse Konto – eine Spezialisierung der Klasse Geld .....	304
	10.7.3 Vertiefung: Standardklassen als Basisklassen .....	307
10.8	Hinweise zum Programmierstil .....	309
	10.8.1 Schreibweise .....	309
	10.8.2 Sichtbarkeit .....	309
	10.8.3 Dokumentation von Klassen .....	310
10.9	Typische Fehler .....	311
	10.9.1 Versehentliches Erzeugen neuer Attribute .....	311
	10.9.2 Verwechseln von Methoden und Attributen .....	311
10.10	Aufgaben .....	312
10.11	Lösungen .....	315
<b>11</b>	<b>Klassen wiederverwenden – Module .....</b>	<b>321</b>
11.1	Testen einer Klasse in einem lauffähigen Stand-alone-Skript .....	321
11.2	Module speichern und importieren .....	323
11.3	Den Zugang zu einem Modul sicherstellen .....	325
	11.3.1 Erweitern der Verzeichnisliste sys.path .....	325
	11.3.2 Anwendungsbeispiel: Eine interaktive Testumgebung .....	325
	11.3.3 Kompilieren von Modulen .....	326
11.4	Programmierstil: Verwendung und Dokumentation von Modulen .....	327

<b>12</b>	<b>Objektorientiertes Modellieren</b>	<b>329</b>
12.1	Phasen einer objektorientierten Software-Entwicklung	329
12.1.1	Objektorientierte Analyse (OOA)	329
12.1.2	Objektorientierter Entwurf (OOD)	330
12.1.3	Objektorientierte Programmierung (OOP)	330
12.2	Beispiel: Modell eines Wörterbuchs	330
12.2.1	OOA: Entwicklung einer Klassenstruktur	330
12.2.2	OOD: Entwurf einer Klassenstruktur zur Implementierung in Python	334
12.2.3	OOP: Implementierung der Klassenstruktur	336
12.3	Assoziationen zwischen Klassen	340
12.3.1	Reflexive Assoziationen	340
12.3.2	Aggregation	342
12.4	Beispiel: Management eines Musicals	343
12.4.1	OOA	343
12.4.2	OOD	345
12.4.3	OOP	345
12.5	Aufgaben	355
12.6	Lösungen	356
<b>13</b>	<b>Textverarbeitung</b>	<b>361</b>
13.1	Standardmethoden zur Verarbeitung von Zeichenketten	361
13.1.1	Formatieren	362
13.1.2	Schreibweise	362
13.1.3	Tests	363
13.1.4	Entfernen und Aufspalten	364
13.1.5	Suchen und Ersetzen	365
13.2	Codierung und Decodierung	365
13.2.1	Platonische Zeichen und Unicode	365
13.2.2	Vertiefung: Zeichenketten durch Bytefolgen darstellen	367
13.3	Automatische Textproduktion	369
13.3.1	Texte mit variablen Teilen – Anwendung der String-Methode format()	369
13.3.2	Vertiefung: Eine Tabelle erstellen	372
13.3.3	Mahnbriefe	373
13.3.4	Textuelle Repräsentation eines Objekts	374
13.3.5	F-Strings	376
13.4	Analyse von Texten	377
13.4.1	Chat-Bots	377
13.4.2	Textanalyse mit einfachen Vorkommenstests	378
13.5	Reguläre Ausdrücke	380
13.5.1	Die Funktion findall() aus dem Modul re	381
13.5.2	Aufbau eines regulären Ausdrucks	381
13.5.3	Objekte für reguläre Ausdrücke	384
13.5.4	Strings untersuchen mit search()	385

13.5.5	Textpassagen extrahieren mit findall() .....	386
13.5.6	Zeichenketten zerlegen mit split() .....	387
13.5.7	Teilstrings ersetzen mit sub() .....	388
13.5.8	Match-Objekte .....	389
13.6	Den Computer zum Sprechen bringen – Sprachsynthese .....	391
13.6.1	Buchstabieren .....	393
13.6.2	Den Klang der Stimme verändern .....	395
13.7	Aufgaben .....	398
13.8	Lösungen .....	401
<b>14</b>	<b>Systemfunktionen .....</b>	<b>409</b>
14.1	Das Modul sys – die Schnittstelle zum Laufzeitsystem .....	409
14.1.1	Informationen über die aktuelle Systemumgebung .....	410
14.1.2	Standardeingabe und -ausgabe .....	411
14.1.3	Die Objektverwaltung beobachten mit getrefcount() .....	412
14.1.4	Ausführung eines Skripts beenden .....	413
14.2	Das Modul os – die Schnittstelle zum Betriebssystem .....	413
14.2.1	Dateien und Verzeichnisse suchen .....	414
14.2.2	Hintergrund: Zugriffsrechte abfragen und ändern (Windows und Unix) .....	415
14.2.3	Dateien und Verzeichnisse anlegen und modifizieren .....	417
14.2.4	Merkmale von Dateien und Verzeichnissen abfragen .....	418
14.2.5	Pfade verarbeiten .....	419
14.2.6	Hintergrund: Umgebungsvariablen .....	421
14.2.7	Systematisches Durchlaufen eines Verzeichnisbaumes .....	422
14.3	Datum und Zeit .....	424
14.3.1	Funktionen des Moduls time .....	425
14.3.2	Sekundenformat .....	425
14.3.3	Zeit-Tupel .....	426
14.3.4	Zeitstrings .....	427
14.3.5	Einen Prozess unterbrechen mit sleep() .....	428
14.4	Zeitberechnungen mit dem Modul datetime .....	428
14.4.1	Die Klasse datetime .....	428
14.4.2	Die Zeitzone .....	430
14.4.3	Die Klasse timedelta .....	431
14.5	Aufgaben .....	431
14.6	Lösungen .....	432
<b>15</b>	<b>Grafische Benutzungsoberflächen mit tkinter .....</b>	<b>437</b>
15.1	Ein einführendes Beispiel .....	438
15.2	Einfache Widgets .....	441
15.3	Die Master-Slave-Hierarchie .....	442
15.4	Optionen der Widgets .....	443
15.4.1	Optionen bei der Instanziierung setzen .....	443
15.4.2	Widget-Optionen nachträglich konfigurieren .....	444



15.4.3	Fonts .....	445
15.4.4	Farben .....	446
15.4.5	Rahmen .....	446
15.4.6	Die Größe eines Widgets .....	447
15.4.7	Leerraum um Text .....	449
15.5	Gemeinsame Methoden der Widgets .....	450
15.6	Die Klasse Tk .....	450
15.7	Die Klasse Button .....	451
15.8	Die Klasse Label .....	451
15.8.1	Dynamische Konfiguration der Beschriftung .....	452
15.8.2	Verwendung von Kontrollvariablen .....	453
15.9	Die Klasse Entry .....	455
15.10	Die Klasse Radiobutton .....	457
15.11	Die Klasse Checkbutton .....	459
15.12	Die Klasse Scale .....	461
15.13	Die Klasse Frame .....	463
15.14	Aufgaben .....	463
15.15	Lösungen .....	464
<b>16</b>	<b>Layout .....</b>	<b>469</b>
16.1	Der Packer .....	469
16.2	Layout-Fehler .....	471
16.3	Raster-Layout .....	472
16.4	Vorgehensweise bei der GUI-Entwicklung .....	476
16.4.1	Die Benutzungsoberfläche gestalten .....	479
16.4.2	Funktionalität hinzufügen .....	482
16.5	Aufgaben .....	483
16.6	Lösungen .....	486
<b>17</b>	<b>Grafik .....</b>	<b>497</b>
17.1	Die tkinter-Klasse Canvas .....	497
17.1.1	Generierung grafischer Elemente – ID, Positionierung und Display-Liste .....	498
17.1.2	Grafische Elemente gestalten .....	500
17.1.3	Visualisieren mit Kreisdiagrammen .....	502
17.2	Die Klasse PhotoImage .....	505
17.2.1	Eine Pixelgrafik erzeugen .....	506
17.2.2	Fotos analysieren und verändern .....	508
17.3	Bilder in eine Benutzungsoberfläche einbinden .....	511
17.3.1	Icons auf Schaltflächen .....	511
17.3.2	Hintergrundbilder .....	512
17.3.3	Hintergrund: Das PPM-Format .....	514
17.4	Die Python Imaging Library (PIL) .....	515
17.4.1	Installation eines Moduls mit pip .....	515
17.4.2	Mit PIL beliebige Bilddateien einbinden .....	516

17.4.3	Steganografie – Informationen in Bildern verstecken . . . . .	517
17.5	Aufgaben . . . . .	519
17.6	Lösungen . . . . .	520
<b>18</b>	<b>Event-Verarbeitung . . . . .</b>	<b>525</b>
18.1	Einführendes Beispiel . . . . .	526
18.2	Event-Sequenzen . . . . .	528
18.2.1	Event-Typen . . . . .	528
18.2.2	Qualifizierer für Maus- und Tastatur-Events . . . . .	528
18.2.3	Modifizierer . . . . .	530
18.3	Beispiel: Tastaturereignisse verarbeiten . . . . .	530
18.4	Programmierung eines Eventhandlers . . . . .	532
18.4.1	Beispiel für eine Event-Auswertung . . . . .	533
18.5	Bindemethoden . . . . .	534
18.6	Aufgaben . . . . .	534
18.7	Lösungen . . . . .	537
<b>19</b>	<b>Komplexe Benutzungsoberflächen . . . . .</b>	<b>543</b>
19.1	Text-Widgets . . . . .	543
19.1.1	Methoden der Text-Widgets . . . . .	544
19.2	Rollbalken (Scrollbars) . . . . .	546
19.3	Menüs . . . . .	547
19.3.1	Die Klasse Menu . . . . .	548
19.3.2	Methoden der Klasse Menu . . . . .	548
19.4	Texteditor mit Menüleiste und Pulldown-Menü . . . . .	550
19.5	Dialogboxen . . . . .	552
19.6	Applikationen mit mehreren Fenstern . . . . .	556
19.7	Aufgaben . . . . .	559
19.8	Lösungen . . . . .	560
<b>20</b>	<b>Threads . . . . .</b>	<b>565</b>
20.1	Funktionen in einem Thread ausführen . . . . .	566
20.2	Thread-Objekte erzeugen – die Klasse Thread . . . . .	568
20.3	Aufgaben . . . . .	571
20.4	Lösungen . . . . .	572
<b>21</b>	<b>Fehler finden und vermeiden . . . . .</b>	<b>577</b>
21.1	Testen von Bedingungen . . . . .	577
21.1.1	Ausnahmen (Exceptions) . . . . .	577
21.1.2	Testen von Vor- und Nachbedingungen mit assert . . . . .	578
21.1.3	Vertiefung: Programmabstürze ohne Fehlermeldung . . . . .	581
21.2	Debugging-Modus und optimierter Modus . . . . .	583
21.3	Ausnahmen gezielt auslösen . . . . .	584
21.4	Selbstdokumentation . . . . .	585
21.5	Dokumentation eines Programmlaufs mit Log-Dateien . . . . .	587
21.5.1	Grundfunktionen . . . . .	587

21.5.2	Beispiel: Logging in der GUI-Programmierung .....	588
21.6	Vertiefung: Professionelles Arbeiten mit Logging .....	589
21.6.1	Logging-Levels .....	589
21.6.2	Logger-Objekte .....	594
21.6.3	Das Format der Logging-Meldungen konfigurieren .....	594
21.7	Debugging .....	596
21.7.1	Schaltflächen des Debug-Control-Fensters .....	597
21.7.2	Breakpoints .....	597
<b>22</b>	<b>Dynamische Webseiten – CGI und WSGI .....</b>	<b>599</b>
22.1	Wie funktionieren dynamische Webseiten? .....	599
22.2	Wie spät ist es? Aufbau eines CGI-Skripts. ....	601
22.2.1	Die Ausgabe eines CGI-Skripts .....	601
22.2.2	Wie ist ein CGI-Skript aufgebaut? .....	602
22.2.3	Verwendung von Schablonen. ....	603
22.2.4	Aufruf mit dem Webbrowser .....	604
22.2.5	Ein einfacher HTTP-Server .....	605
22.3	Kommunikation über interaktive Webseiten. ....	605
22.3.1	Aufbau eines HTML-Formulars. ....	606
22.3.2	Eingabekomponenten in einem HTML-Formular .....	608
22.4	Verarbeitung von Eingabedaten mit FieldStorage. ....	610
22.5	Sonderzeichen handhaben .....	612
22.6	CGI-Skripte debuggen. ....	613
22.7	Der Apache-Webserver .....	614
22.7.1	Den Apache-Server installieren .....	615
22.7.2	CGI-Skripte auf dem Apache-Server .....	616
22.8	Dynamische Webseiten mit WSGI. ....	616
22.8.1	Einfacher geht's nicht: Ein Stand-alone-WSGI-Webserver mit wsgiref .....	617
22.9	mod_wsgi .....	618
22.9.1	Installation .....	618
22.9.2	Vorbereitung .....	619
22.9.3	Den Apache-Server konfigurieren .....	619
22.9.4	Ein WSGI-Skript für den Apache-Server. ....	621
22.9.5	Tipps zum Debuggen .....	621
22.9.6	Zugriff von einem entfernten Rechner im WLAN .....	622
22.10	Verarbeitung von Eingabedaten aus Formularen .....	623
22.11	Objektorientierte WSGI-Skripte – Beispiel: ein Chatroom. ....	625
22.11.1	Die HTML-Seiten .....	627
22.11.2	Die Klassen für den Chatroom. ....	629
22.11.3	Skript (Teil 2): .....	629
22.12	WSGI-Skripte mit Cookies .....	632
22.12.1	Besuche zählen .....	633

<b>23</b>	<b>Internet-Programmierung</b>	641
23.1	Was ist ein Protokoll?	641
23.2	Übertragung von Dateien mit FTP	642
23.2.1	Das Modul ftplib	642
23.2.2	Navigieren und Downloaden	643
23.2.3	Ein Suchroboter für FTP-Server	645
23.3	Zugriff auf Webseiten mit HTTP und HTTPS	649
23.3.1	Automatische Auswertung von Webseiten	651
23.4	Zugriff auf Ressourcen im Internet über deren URL	653
23.4.1	Webseite herunterladen und verarbeiten	653
23.4.2	Projekt: Wie warm wird es heute?	654
23.4.3	Datei herunterladen und speichern	655
23.4.4	Projekt: Filme herunterladen	655
23.5	E-Mails senden mit SMTP	657
23.6	Aufgaben	660
23.7	Lösungen	661
<b>24</b>	<b>Datenbanken</b>	669
24.1	Was ist ein Datenbanksystem?	669
24.2	Entity-Relationship-Diagramme (ER-Diagramme)	670
24.3	Relationale Datenbanken	671
24.4	Darstellung von Relationen als Mengen oder Dictionaries	672
24.5	Das Modul sqlite3	673
24.5.1	Beispiel: Telefonbuch	673
24.5.2	Eine Tabelle anlegen	674
24.5.3	Anfragen an eine Datenbank	675
24.5.4	Datensuche im interaktiven Modus	676
24.5.5	SQL-Anweisungen mit variablen Teilen	678
24.5.6	Vertiefung: SQL-Injection	680
24.6	Online-Redaktionssystem mit Datenbankbindung	681
24.6.1	Objektorientierte Analyse (OOA)	683
24.6.2	Objektorientierter Entwurf des Systems (OOD)	684
24.6.3	Hintergrund: Authentifizieren mit SHA-256	686
24.6.4	Implementierung des Redaktionssystems mit Python (OOP)	687
24.7	Aufgaben	697
24.8	Lösungen	698
<b>25</b>	<b>Testen und Tuning</b>	701
25.1	Automatisiertes Testen	701
25.2	Testen mit Docstrings – das Modul doctest	701
25.3	Praxisbeispiel: Suche nach dem Wort des Jahres	704
25.4	Klassen testen mit doctest	711
25.4.1	Wie testet man eine Klasse?	711
25.4.2	Normalisierte Whitespaces – doctest-Direktiven	712
25.4.3	Ellipsen verwenden	712

25.4.4	Dictionaries testen. . . . .	713
25.5	Gestaltung von Testreihen mit unittest . . . . .	713
25.5.1	Einführendes Beispiel mit einem Testfall. . . . .	714
25.5.2	Klassen des Moduls unittest. . . . .	715
25.5.3	Weiterführendes Beispiel . . . . .	718
25.6	Tuning . . . . .	721
25.6.1	Performance-Analyse mit dem Profiler. . . . .	721
25.6.2	Praxisbeispiel: Auswertung astronomischer Fotografien. . . . .	723
25.6.3	Performance-Analyse und Tuning. . . . .	729
25.7	Aufgaben . . . . .	730
25.8	Lösungen . . . . .	732
<b>26</b>	<b>XML und JSON . . . . .</b>	<b>739</b>
26.1	Was ist XML? . . . . .	739
26.2	XML-Dokumente . . . . .	740
26.3	Ein XML-Dokument als Baum . . . . .	742
26.4	DOM . . . . .	743
26.5	Das Modul xml.dom.minidom . . . . .	746
26.5.1	XML-Dokumente und DOM-Objekte . . . . .	746
26.5.2	Die Basisklasse Node . . . . .	748
26.5.3	Die Klassen Document, Element und Text . . . . .	750
26.6	Attribute von XML-Elementen . . . . .	752
26.7	Anwendungsbeispiel 1: Eine XML-basierte Klasse . . . . .	752
26.8	Anwendungsbeispiel 2: Datenkommunikation mit XML. . . . .	755
26.8.1	Überblick . . . . .	756
26.8.2	Das Client-Programm. . . . .	757
26.8.3	Das Server-Programm. . . . .	760
26.9	JSON . . . . .	764
26.9.1	JSON-Texte decodieren. . . . .	765
26.9.2	Decodierungsfehler. . . . .	766
26.9.3	Ein Dictionary als JSON-Objekt speichern: Kompakt oder gut lesbar? . . . . .	766
26.9.4	Projekt: Verarbeitung von Wetterdaten. . . . .	769
26.10	Aufgaben . . . . .	772
26.11	Lösungen . . . . .	773
<b>27</b>	<b>Modellieren mit Kellern, Schlangen und Graphen . . . . .</b>	<b>775</b>
27.1	Stack (Keller, Stapel) . . . . .	775
27.2	Queue (Schlange). . . . .	778
27.3	Graphen . . . . .	779
27.4	Aufgaben . . . . .	789
27.5	Lösungen . . . . .	791
<b>28</b>	<b>Benutzungsoberflächen mit Qt. . . . .</b>	<b>795</b>
28.1	Was bietet PyQt5? . . . . .	795
28.2	PyQt5 erkunden . . . . .	796



28.3	Wie arbeitet PyQt? Applikation und Fenster.....	796
28.4	Eine objektorientierte Anwendung mit PyQt5 .....	797
28.5	Ein Webbrowser .....	798
28.6	Interaktive Widgets .....	802
28.7	Label – Ausgabe von Text und Bild .....	803
28.8	Signale .....	804
28.9	Checkboxes und Radiobuttons .....	805
28.10	Auswahlliste (ComboBox).....	808
28.11	Gemeinsame Operationen der Widgets .....	810
28.12	Spezielle Methoden eines Fensters .....	811
28.13	Events.....	813
28.14	Fonts.....	814
28.15	Stylesheets .....	816
28.16	Icons.....	819
28.17	Messageboxen .....	819
28.18	Timer .....	820
28.19	Das Qt-Layout unter der Lupe .....	822
	28.19.1 Absolute Positionierung und Größe .....	822
	28.19.2 Raster-Layout.....	824
	28.19.3 Form-Layout .....	825
28.20	Browser für jeden Zweck .....	827
	28.20.1 Die Klasse QWebEngineView .....	827
28.21	Ein Webbrowser mit Filter .....	828
28.22	Surfen mit Geschichte – der Verlauf einer Sitzung .....	830
28.23	Aufgaben .....	832
28.24	Lösungen .....	833
<b>29</b>	<b>Multimediaanwendungen mit Qt .....</b>	<b>837</b>
29.1	Kalender und Textfeld – ein digitales Tagebuch .....	837
	29.1.1 Programmierung .....	838
29.2	Kamerabilder .....	843
29.3	Dialoge .....	845
	29.3.1 Projekt: Ansichtskarte .....	847
29.4	Videoplayer .....	851
	29.4.1 Ein einfacher Videoplayer .....	851
	29.4.2 Videoplayer mit Playlist .....	855
	29.4.3 Regeln zur Änderung der Größe (Size Policy).....	858
	29.4.4 Das Dashboard bei Mausbewegungen einblenden .....	859
29.5	Aufgaben .....	862
29.6	Lösungen .....	867
<b>30</b>	<b>Rechnen mit NumPy.....</b>	<b>875</b>
30.1	NumPy installieren .....	875
30.2	Arrays erzeugen.....	875
	30.2.1 Arrays.....	875

30.2.2	Matrizen und Vektoren . . . . .	878
30.2.3	Zahlenfolgen . . . . .	878
30.2.4	Zufallsarrays . . . . .	879
30.2.5	Spezielle Arrays . . . . .	880
30.3	Indizieren . . . . .	881
30.4	Slicing . . . . .	882
30.5	Arrays verändern . . . . .	883
30.6	Arithmetische Operationen . . . . .	885
30.7	Funktionen, die elementweise ausgeführt werden . . . . .	886
30.8	Einfache Visualisierung . . . . .	887
30.9	Matrizenmultiplikation mit dot() . . . . .	888
30.10	Array-Funktionen und Achsen . . . . .	889
30.11	Projekt: Diffusion. . . . .	891
30.12	Vergleiche. . . . .	894
30.13	Projekt: Wolken am Himmel . . . . .	894
30.14	Projekt: Wie versteckt man ein Buch in einem Bild? . . . . .	897
30.15	Datenanalyse mit Histogrammen. . . . .	900
30.16	Wie funktioniert ein Medianfilter? . . . . .	903
30.17	Rechnen mit SciPy. . . . .	906
30.17.1	Lineare Gleichungssysteme lösen . . . . .	906
30.17.2	Integration . . . . .	908
30.18	Aufgaben . . . . .	909
30.19	Lösungen . . . . .	912
<b>31</b>	<b>Messdaten verarbeiten. . . . .</b>	<b>917</b>
31.1	Messwerte in einem Diagramm darstellen – Matplotlib und tkinter . . . . .	917
31.1.1	Basisprojekt . . . . .	917
31.1.2	Erweiterung: Den letzten Wert löschen. . . . .	921
31.1.3	Das Aussehen eines Diagramms gestalten . . . . .	923
31.2	Messwerte aus einem Multimeter lesen und darstellen . . . . .	926
31.2.1	Vorbereitung . . . . .	926
31.2.2	Werte auslesen . . . . .	927
31.2.3	Welche Ziffern zeigt das Display des Multimeters? . . . . .	930
31.3	Anzeige der Temperatur . . . . .	934
31.4	Messreihen aufzeichnen . . . . .	936
31.5	Aufgabe. . . . .	939
31.6	Lösung . . . . .	939
<b>32</b>	<b>Parallele Datenverarbeitung . . . . .</b>	<b>943</b>
32.1	Was sind parallele Programme? . . . . .	943
32.2	Prozesse starten und abbrechen . . . . .	944
32.3	Funktionen in eigenen Prozessen starten . . . . .	945
32.4	Prozesse zusammenführen – join() . . . . .	947
32.5	Wie können Prozesse Objekte austauschen? . . . . .	948
32.5.1	Objekte als Argumente übergeben . . . . .	948

32.5.2	Objekte über eine Pipe senden und empfangen .....	949
32.5.3	Objekte über eine Queue austauschen .....	950
32.6	Daten im Pool bearbeiten .....	951
32.6.1	Mit dem Pool geht's schneller – ein Zeitexperiment .....	951
32.6.2	Forschen mit Big Data aus dem Internet .....	953
32.7	Synchronisation .....	956
32.8	Produzenten und Konsumenten .....	958
32.8.1	Sprücheklopfer .....	959
32.9	Aufgaben .....	961
32.10	Lösungen .....	962
<b>33</b>	<b>Django</b> .....	<b>967</b>
33.1	Django aus der Vogelperspektive .....	967
33.2	Ein Projekt anlegen .....	968
33.2.1	Den Server starten .....	970
33.2.2	Eine neue Applikation anlegen .....	970
33.2.3	Startseite und View einrichten .....	971
33.3	Datenbankanbindung .....	974
33.4	Modelle erstellen .....	974
33.5	Modelle aktivieren .....	976
33.6	In der Python-Shell die Datenbank bearbeiten .....	979
33.6.1	Objekte durch Aufruf der Klasse erzeugen .....	980
33.6.2	Auf Attribute eines Objekts zugreifen .....	981
33.6.3	Objekte finden .....	981
33.6.4	Objekte erzeugen und Beziehungen herstellen .....	982
33.6.5	Den Beziehungsmanager nutzen .....	983
33.6.6	Objekte löschen .....	983
33.7	Django-Modelle unter der Lupe .....	984
33.7.1	Grenzwerte .....	984
33.7.2	Leere Felder .....	985
33.7.3	Voreingestellte Werte .....	985
33.7.4	Einmaligkeit .....	985
33.7.5	Auswahlmöglichkeiten .....	985
33.8	Der Manager unter der Lupe – Objekte erzeugen und suchen .....	986
33.8.1	Objekte erzeugen .....	986
33.8.2	Objekte finden .....	986
33.8.3	Mehrere Bedingungen .....	987
33.8.4	Suchen über Beziehungen .....	988
33.8.5	Weitere Suchmethoden .....	988
33.9	Administration .....	989
33.9.1	Eine Applikation der Website-Verwaltung zugänglich machen ...	991
33.10	Views einrichten – die Grundstruktur .....	995
33.10.1	Was sind Views? .....	995
33.10.2	Funktionen für Views .....	995
33.10.3	URL-Patterns .....	996

33.11	View-Funktionen erweitern. ....	997
33.11.1	Startseite ....	998
33.11.2	Auflistung der Ideen zu einer Frage – question_index ....	1001
33.11.3	Die Templates verbessern: Namen statt expliziter URLs ....	1003
33.12	Interaktive Webseiten – Views mit Formularen ....	1004
33.12.1	Eingabe einer neuen Frage. ....	1004
33.12.2	Eingabe einer neuen Idee. ....	1010
33.12.3	View-Funktion für das Speichern einer neuen Idee. ....	1012
33.12.4	Fertig! ....	1013
33.13	Die nächsten Schritte. ....	1013
33.14	Aufgabe. ....	1014
33.15	Lösung ....	1015
<b>A</b>	<b>Anhang</b> ....	1019
A.1	Codierung von Sonderzeichen in HTML. ....	1019
A.2	Quellen im WWW ....	1019
A.3	Standardfunktionen und Standardklassen. ....	1020
A.4	Mathematische Funktionen ....	1022
A.4.1	Das Modul math ....	1022
A.4.2	Das Modul random ....	1023
A.5	EBNF-Grammatik ....	1024
<b>B</b>	<b>Glossar</b> ....	1029
<b>C</b>	<b>Download der Programmbeispiele</b> ....	1043
<b>D</b>	<b>Ein Python-Modul veröffentlichen: PyPI</b> ....	1045
D.1	Bei PyPI und TestPyPI registrieren ....	1046
D.2	Ein Paket für die Veröffentlichung vorbereiten. ....	1047
D.2.1	Die Programmdatei setup.py ....	1047
D.2.2	Die Lizenz ....	1048
D.2.3	Die Datei README.txt ....	1049
D.2.4	Die Datei __init__.py. ....	1050
D.3	Das Paket auf PyPI veröffentlichen ....	1050
D.3.1	Das Paket aktualisieren. ....	1051
	<b>Stichwortverzeichnis</b> ....	1053

# 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, sogenannten 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 Variablennamen, 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/0544>

Klicken Sie im Kasten DOWNLOADS auf den Link PROGRAMMBEISPIELE.

Außerdem sind die Programmbeispiele in einem GitHub-Repository veröffentlicht. URL:

<https://github.com/mweigend/python3/>

Weitere Hinweise zum Download finden Sie im Anhang C.

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 Programmiertechnik 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 macOS. 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.

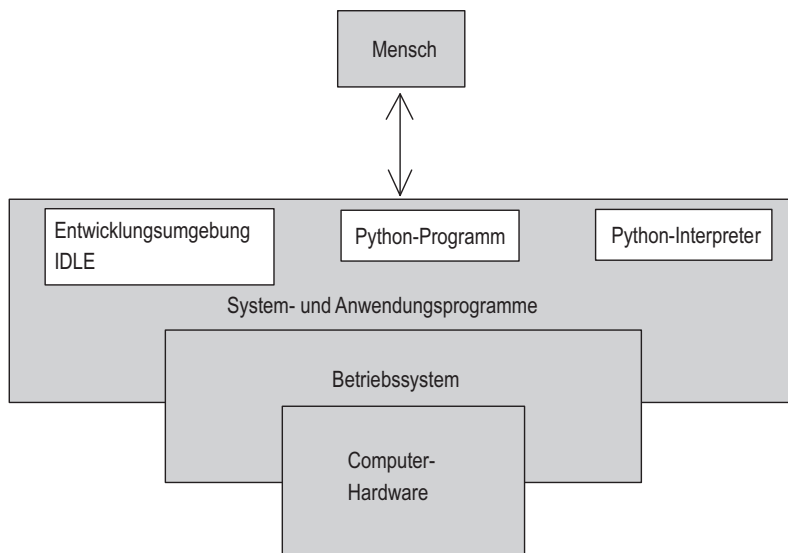


Abb. 1.1: Komponenten eines Computer-Systems



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 ausgenommenenes 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 »versteht« – 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.

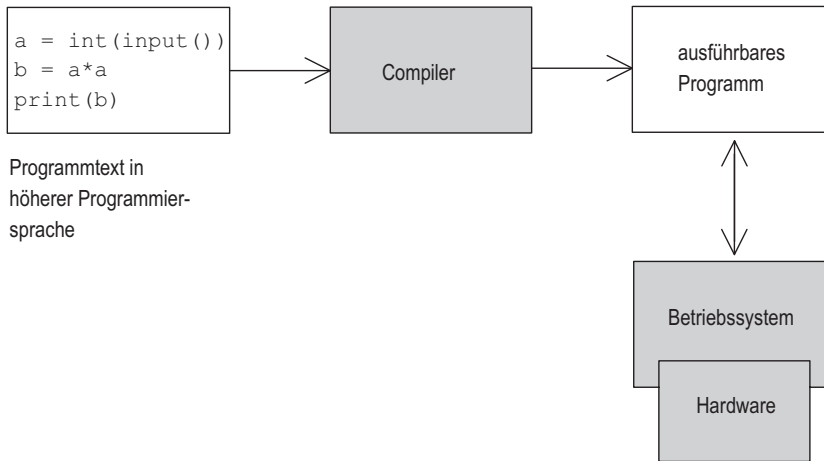
## 1.5 Interpreter und Compiler

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.

Damit ein Programmtext – man spricht auch von Quelltext (*source code*) – vom Computer »verstanden« wird und abgearbeitet werden kann, muss er in ein ausführbares Programm übersetzt werden.

Dazu gibt es zwei unterschiedliche Methoden.

Ein *Compiler* übersetzt einen kompletten Programmtext und erzeugt ein direkt ausführbares Programm, das vom Betriebssystem geladen und gestartet werden kann. Bei der Übersetzung müssen natürlich die Besonderheiten des Rechners, auf dem das Programm laufen soll, berücksichtigt werden. Es gibt dann z.B. unterschiedliche Fassungen für MS-Windows- und Unix-Systeme. Programmiersprachen, bei denen kompiliert wird, sind z.B. Pascal, C, C++.



**Abb. 1.3:** Arbeitsweise eines Compilers

Ein *Interpreter* liest einen Programmtext Zeile für Zeile und führt (über das Betriebssystem) jede Anweisung direkt aus. Wenn ein Programm gestartet werden soll, muss zuerst der Interpreter aufgerufen werden. Für jedes Betriebssystem gibt es zu der Programmiersprache einen eigenen Interpreter. Wer ein Programm in einer interpretativen Sprache verwenden möchte, benötigt also zusätzlich zu dem Anwendungsprogramm noch einen Interpreter.

Python ist eine interpretative Programmiersprache. Dies hat den Vorteil, dass ein und dasselbe Programm auf allen Rechnerplattformen läuft. Als nachteilig könnte man aus Entwicklersicht empfinden, dass der Quelltext einer Software, die man verkaufen möchte, immer offen gelegt ist (*open source*). Damit besteht das Risiko, dass jemand illegalerweise den Programmtext leicht verändert und ihn unter seinem Namen weiterverkauft. Das geistige Eigentum des Programmentwicklers ist also schlecht geschützt. Auf der anderen Seite gibt es einen gewissen Trend, nur solche Software einzusetzen, deren Quelltext bekannt ist. Denn nur dann ist es möglich, etwaige Fehler, die erst im Lauf des Betriebes sichtbar werden, zu finden und zu beseitigen. Wer Software verwendet, deren Quelltext geheim gehalten ist, macht sich vom Software-Hersteller abhängig, und ist im Störfall »auf Gedeih und Verderb« auf ihn angewiesen.

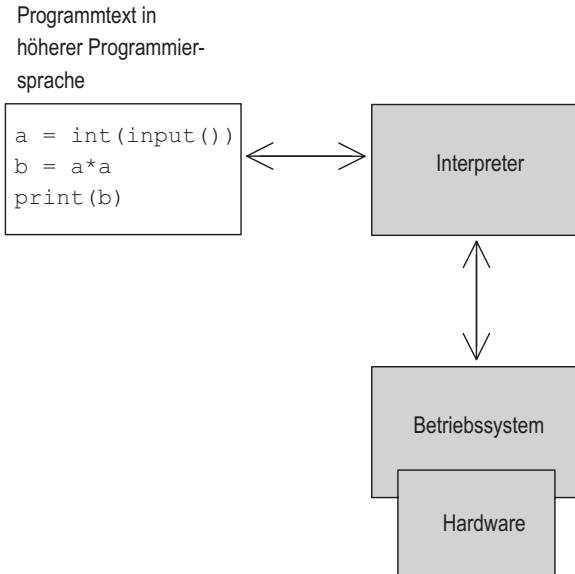


Abb. 1.4: Arbeitsweise eines Interpreters

## 1.6 Programmierparadigmen

Ein Paradigma ist allgemein ein Denk- oder Handlungsmuster, an dem man sich z.B. bei der Formulierung einer Problemlösung orientiert. Wenn man ein Programm als Algorithmus betrachtet, also als System von Befehlen, folgt man dem *imperativen* Programmierparadigma (*imperare*: lat. befehlen).

Zur Abgrenzung sei kurz darauf hingewiesen, dass es auch andere Programmierparadigmen gibt. *Prolog* z.B. ist eine *deklarative* Programmiersprache. Ein deklaratives Programm beschreibt Eigenschaften der Lösung des Problems. Der Programmierer legt sein Augenmerk auf die Frage, *was* berechnet werden soll, und nicht, *wie* man es berechnet. Dagegen stellt ein imperatives Programm eine Anleitung dar. Sie beschreibt, *wie* – Schritt für Schritt – die Aufgabe gelöst werden soll.

Das folgende kleine Experiment veranschaulicht den Unterschied. Wenn Sie es selbst durchspielen wollen, benötigen Sie sieben Streichhölzer. Die beiden folgenden Texte beschreiben auf deklarative und auf imperative Weise, wie die Streichhölzer angeordnet werden sollen. Probieren Sie aus, mit welchem Paradigma Sie besser zurechtkommen.

### Deklaratives Paradigma:

- Insgesamt gibt es sieben Streichhölzer.
- Genau ein Streichholz berührt an beiden Enden jeweils zwei weitere Streichhölzer.
- Wenigstens ein Streichholz bildet mit zwei benachbarten Streichhölzern jeweils einen rechten Winkel.

- Drei Streichhölzer liegen zueinander parallel, berühren sich aber nicht.
- Es gibt kein Streichholz, das nicht an jedem Ende wenigstens ein anderes Streichholz berührt.

#### Imperatives Paradigma:

- Legen Sie zwei Streichhölzer (A und B) in einer geraden Linie nebeneinander auf den Tisch, so dass sie sich an einer Stelle berühren.
- Legen Sie ein Streichholz C mit einem Ende an der Stelle an, wo sich A und B berühren. Das Streichholz soll einen rechten Winkel zu A und B bilden.
- Legen Sie an die äußeren Enden von A und B jeweils ein weiteres Streichholz mit einem Ende an (D und E), so dass diese neuen Streichhölzer jeweils einen rechten Winkel zu A und B bilden und in die gleiche Richtung gehen wie das mittlere Streichholz.
- Verbinden Sie die noch freien Enden von C, D und E mit den verbleibenden zwei Streichhölzern.

Eine Abbildung der korrekten Anordnung finden Sie am Ende des Kapitels. Vermutlich haben Sie die zweite Aufgabe schneller lösen können. Tatsächlich benötigen auch in der Computertechnik imperative Programme weniger Rechenzeit als deklarative.

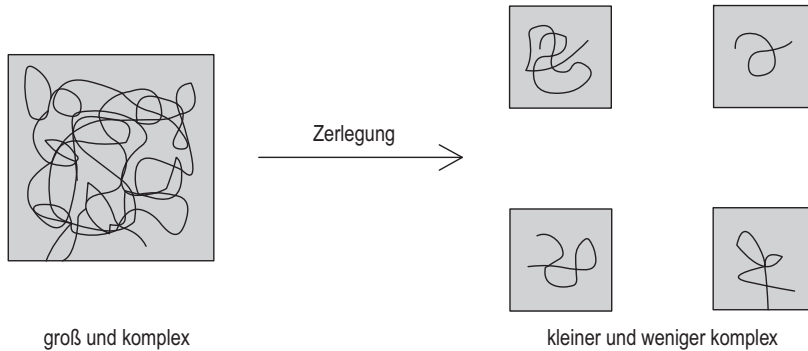
Zum Schluss sei noch das Paradigma der *funktionalen* Programmierung erwähnt. Mit funktionalen Programmiersprachen wie z.B. *Haskell* oder *Scheme* kann man ein Programm als (mathematische) Funktion definieren. Einfache vorgegebene Funktionen werden zu einer komplexen Funktion verknüpft, die das Gewünschte leistet. Mathematisch geschulten Menschen fällt diese Art der Programmentwicklung bei bestimmten Problemen leichter.

Man kann mit Fug und Recht sagen, dass unter diesen drei Paradigmen der imperative Ansatz am verbreitetsten ist. Funktionale und deklarative Sprachen spielen heute in der Praxis der Software-Entwicklung eher eine untergeordnete Rolle. Auch die *objektorientierte Programmierung* (OOP) wird als eigenes Programmierparadigma beschrieben. Das hört sich so an, als wäre die objektorientierte Programmierung etwas ganz anderes als das imperative oder funktionale Paradigma. Aber ganz so ist es eigentlich nicht. Vielmehr betrifft das Paradigma der Objektorientierung einen Aspekt der Programmentwicklung, den ich bisher noch nicht erwähnt habe. Es geht um die Beherrschung von Komplexität.

## 1.7 Objektorientierte Programmierung

### 1.7.1 Strukturelle Zerlegung

Die ersten Computerprogramme waren einfach und dienten der Lösung eines relativ kleinen, gut umgrenzten Problems. Die Situation wird ganz anders, wenn man umfangreiche Software erstellen möchte, etwa ein Textverarbeitungsprogramm oder ein Verwaltungsprogramm für eine Bibliothek. Solche großen Systeme lassen sich nur beherrschen, wenn man sie zunächst in kleinere überschaubare Teile aufbricht. Abbildung 1.5 soll diesen Gedanken veranschaulichen.



**Abb. 1.5:** Zerlegung eines komplexen Systems

Die Vorteile liegen auf der Hand:

Die kleineren Teile des Ganzen lassen sich einfacher programmieren. Die Wahrscheinlichkeit, dass sie Fehler enthalten, ist geringer. Mehrere Personen können zeitgleich und unabhängig voneinander die Einzelteile erstellen. Das spart Zeit. Und es kann sein, dass man später einen Baustein, den man früher einmal programmiert hat, wieder verwenden kann. Das spart Kosten.

Das objektorientierte Paradigma bietet ein Verfahren, nach dem große Systeme in kleinere Teile zerlegt werden können.

## 1.7.2 Die Welt als System von Objekten

In der objektorientierten Sichtweise stellt man sich die Welt als System von Objekten vor, die untereinander Botschaften austauschen. Zur Veranschaulichung betrachten wir ein Beispiel aus dem Alltag, das in Abbildung 1.6 illustriert wird.

Leonie in Bonn möchte ihrer Freundin Elena in Berlin einen Blumenstrauß schicken. Sie geht deshalb zu Mark, einem Blumenhändler, und erteilt ihm einen entsprechenden Auftrag. Betrachten wir Mark als Objekt. In der Sprache der objektorientierten Programmierung sagt man: Leonie sendet an das Objekt Mark eine Botschaft, nämlich: »Sende sieben gelbe Rosen an Elena, Markgrafenstr. 10 in Berlin.«. Damit hat sie getan, was sie tun konnte. Es liegt nun in Marks Verantwortung, den Auftrag zu bearbeiten. Mark versteht die Botschaft und weiß, was zu tun ist. Das heißt, er kennt einen Algorithmus für das Verschicken von Blumen. Der erste Schritt ist, einen Blumenhändler in Berlin zu finden, der die Rosen an Elena liefern kann. In seinem Adressverzeichnis findet er den Floristen Sascha. Ihm sendet er eine leicht veränderte Botschaft, die nun zusätzlich noch den Absender enthält. Damit ist Mark fertig und hat die Verantwortung für den Prozess weitergegeben. Auch Sascha hat einen zur Botschaft passenden Algorithmus parat. Er stellt den gewünschten Blumenstrauß zusammen und beauftragt seinen Boten Daniel, die Rosen auszuliefern. Daniel muss nun den Weg zur Zieladresse finden und befragt seine Straßenkarte. Sie antwortet ihm mit einer Wegbeschreibung. Nachdem Daniel den Weg zu Elenas Wohnung gefunden hat, überreicht er die Blumen und teilt ihr in einer Botschaft mit, von wem sie stammen. Damit ist der gesamte Vorgang, den Leonie angestoßen hat und an dem mehrere Objekte beteiligt waren, beendet.

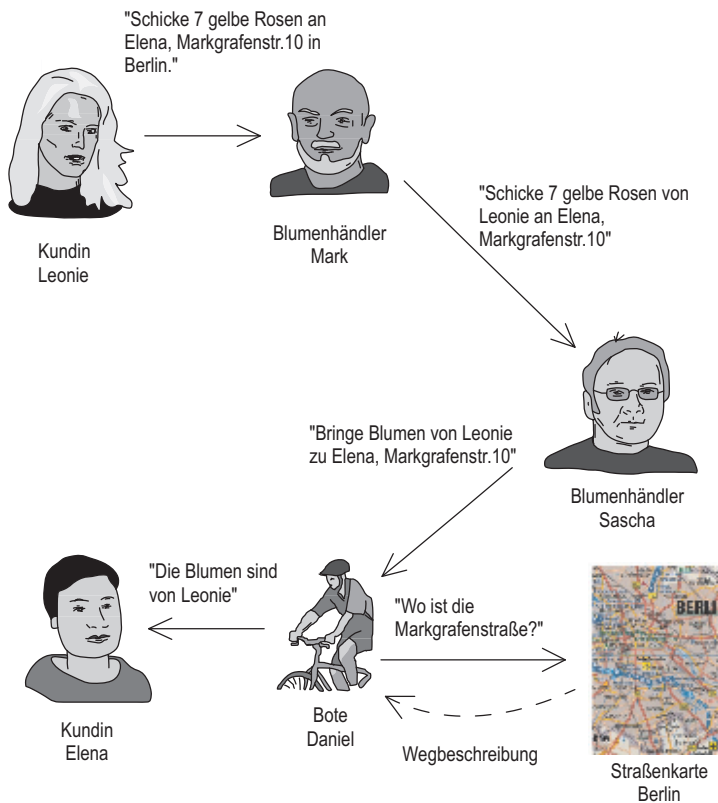


Abb. 1.6: Objektorientiertes Modell eines Blumenversandsystems

### 1.7.3 Objekte besitzen Attribute und beherrschen Methoden

Jedes Objekt besitzt Eigenschaften oder *Attribute*. Ein Attribut eines Blumenhändlers ist z.B. die Stadt, in der er sein Geschäft hat. Dieses Attribut ist auch für die Umwelt wichtig. So musste Mark einen Blumenhändler mit dem Attribut »wohnhaft in Berlin« suchen. Weitere typische Attribute von Blumenhändlern sind Name, Telefonnummer, Warenbestand oder Öffnungszeiten.

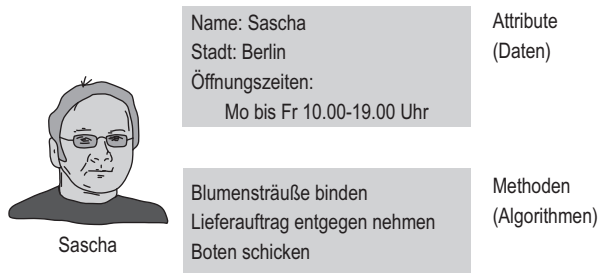


Abb. 1.7: Objekte besitzen Attribute und beherrschen Methoden.

Objekte sind in der Lage, bestimmte Operationen auszuführen, die man *Methoden* nennt. Ein Blumenhändler z.B. kann einen Lieferauftrag für Blumen entgegennehmen, Sträuße binden, einen Boten schicken, beim Großhandel neue Blumen einkaufen usw. Wenn ein Objekt eine geeignete Botschaft empfängt, wird eine zur Botschaft passende Operation gestartet. Man sagt: Die Methode wird aufgerufen. Der Umwelt, das heißt den anderen Objekten, ist bekannt, welche Methoden ein Objekt beherrscht. Die Umwelt weiß von den Methoden nur,

- was sie bewirken
- welche Daten sie als Eingabe benötigen

Die Umwelt weiß aber nicht, *wie* das Objekt funktioniert, das heißt, nach welchen Algorithmen die Botschaften verarbeitet werden. Dieses bleibt ein privates Geheimnis des Objektes.

Leonie hat keine Ahnung, wie Mark den Blumentransport bewerkstelligt. Es interessiert sie auch gar nicht. Ihre Aufgabe bestand allein darin, für ihr Problem ein geeignetes Objekt zu finden und ihm eine geeignete Botschaft zu senden. Ein ungeeignetes Objekt wäre zum Beispiel Tom, der Zahnarzt, oder Katrin, die Leiterin des Wasserwerks gewesen. Diese Objekte hätten Leonis Nachricht gar nicht verstanden und zurückgewiesen. Außerdem ist für Leonie wichtig, wie sie die Botschaft an Mark formuliert. Sie muss ihm ihren Namen mitteilen (damit der Empfänger weiß, von wem die Blumen sind), die Adresse des Empfängers sowie Anzahl und Sorte der Blumen, die gesendet werden sollen.

Eine Methode ist die Implementierung (technische Realisierung) eines Algorithmus. Bei der Programmierung einer Methode mit Python (oder einer anderen objektorientierten Sprache) wird also wieder das imperative Paradigma wichtig.

### 1.7.4 Objekte sind Instanzen von Klassen

Die Objekte des Beispiels kann man in Gruppen einteilen. Sascha und Mark sind beide Blumenhändler. Sie beherrschen beide dieselben Methoden und besitzen dieselben Attribute (z.B. die Stadt), allerdings mit unterschiedlichen Werten. Man sagt: Sascha und Mark sind *Instanzen* der Klasse »Blumenhändler«. In der objektorientierten Programmierung ist eine Klasse die Definition eines bestimmten Typs von Objekten. Sie ist so etwas wie ein Bauplan, in dem die Methoden und Attribute beschrieben werden. Nach diesem Schema können Objekte (Instanzen) einer Klasse erzeugt werden. Ein Objekt ist eine Konkretisierung, eine Inkarnation einer Klasse. Alle Instanzen einer Klasse sind von der Struktur her gleich. Sie unterscheiden sich allein in der Belegung ihrer Attribute mit Werten. Die Objekte Sascha und Mark besitzen dasselbe Attribut »Stadt«, aber bei Sascha trägt es den Wert »Berlin« und bei Mark »Bonn«.

## 1.8 Hintergrund: Geschichte der objektorientierten Programmierung

Die Grundideen der Objektorientierung (wie z.B. die Begriffe Klasse und Objekt) tauchen zum ersten Mal in der Simulationssprache SIMULA auf. Sie wurde von Ole-Johan Dahl and Kristen Nygaard am Norwegian Computing Centre (NCC) in Oslo zwischen 1962 und 1967 entwickelt und diente zur Simulation komplexer Systeme der realen Welt. Die erste universell verwendbare objektorientierte Programmiersprache wurde in den Jahren 1970 bis 1980



am Palo Alto Research Center der Firma Xerox von Alan Key und seinem Team entwickelt und unter dem Namen SmallTalk-80 in die Öffentlichkeit gebracht. Wenig später entstand in den Bell Laboratories (AT&T, USA) unter der Leitung von Bjarne Stroustrup die Sprache C++ als objektorientierte Erweiterung von C. Sie wurde zu Beginn der Neunzigerjahre zur dominierenden objektorientierten Sprache. Mitte der Neunzigerjahre etablierte sich Java (Sun Microsystems Inc.) auf dem Markt. Die Entwicklung von Python wurde 1989 von Guido van Rossum am Centrum voor Wiskunde en Informatica (CWI) in Amsterdam begonnen und wird nun durch die nichtkommerzielle Organisation Python Software Foundation (PSF) koordiniert. Gegenwärtig gibt es eine rasch wachsende Community von Python-Programmierern.

Etwa parallel zur Entwicklung von objektorientierten Programmiersprachen wurden Konzepte der objektorientierten Analyse (OOA) und des objektorientierten Entwurfs (OOD) veröffentlicht. Im Prozess einer objektorientierten Software-Entwicklung sind OOA und OOD der Implementierung in einer Programmiersprache vorgelagert. Im Gegensatz zur rein textuellen Notation der Programmiersprachen verwenden objektorientierte Analyse- und Entwurfsmethoden auch visuelle Darstellungen. Besonders zu erwähnen ist die Unified Modeling Language (UML), die in der Version 1.1 im September 1997 publiziert wurde und heute so etwas wie einen Industriestandard zur grafischen Beschreibung objektorientierter Software-Systeme darstellt.

## 1.9 Aufgaben

### Aufgabe 1

Welche der folgenden Texte sind Algorithmen?

1. Liebesbrief
2. Formular zur Beantragung eines Personalausweises
3. Märchen
4. Musterlösung einer Mathematikaufgabe
5. Die christlichen Zehn Gebote

### Aufgabe 2

Ordnen Sie den folgenden Beschreibungen einer Problemlösung passende Programmierparadigmen zu (imperativ, objektorientiert, deklarativ).

1. Um ein Zündholz zu entzünden, reiben Sie den Kopf des Zündholzes über die Reibfläche.
2. Um eine Menge von Blumenvasen der Größe nach zu sortieren, sorgen Sie davor, dass jede Blumenvase entweder am Anfang der Reihe steht oder größer als ihr linker Nachbar ist.
3. Der Betrieb in einem Restaurant funktioniert so: Es gibt einen Koch und einen Kellner. Der Kellner kümmert sich um die Gäste, säubert die Tische, bringt das Essen und kassiert. Der Koch bereitet das Essen zu, wenn er vom Kellner einen Auftragszettel mit den Nummern der bestellten Gerichte erhält.

## 1.10 Lösungen

### Lösung 1

1. Liebesbriefe können natürlich sehr unterschiedlich aussehen, manche sind leidenschaftlich, andere poetisch und sensibel. Wenn auch nach Auffassung des Kommunikationstheoretikers Schulz von Thun jede sprachliche Botschaft (unter anderem) auch eine appellative Dimension hat, dürfte ein Liebesbrief insgesamt wohl kaum als Anweisung zur Lösung eines Problems zu sehen sein und ist damit kein Algorithmus.
2. Ein solches Formular besitzt die entscheidenden Merkmale eines Algorithmus. Es beschreibt (einigermaßen unmissverständlich) alle Aktionen, die ausgeführt werden müssen, um das Problem »Wie komme ich an einen Personalausweis?« zu lösen.
3. Märchen erzählen, was vor langer Zeit passiert ist. Sie sind keine Algorithmen.
4. Eine gut formulierte Musterlösung beschreibt in der Regel einen Lösungsweg, führt also die mathematischen Operationen (in der richtigen Reihenfolge) auf, die man ausführen muss, um die Aufgabe zu lösen. Sie kann somit als Algorithmus (mit Kommentaren zum besseren Verständnis) betrachtet werden.
5. Die Zehn Gebote sind zwar allgemeine Verhaltensvorschriften (soziale Normen), definieren aber kein konkretes Verhalten, das zu einer Problemlösung führt.

### Lösung 2

1. Imperativ. Es handelt sich um eine Folge von Anweisungen.
2. Deklarativ. Es wird beschrieben, welche Eigenschaften die Lösung (nach Größe sortierte Blumenvasen) haben muss, aber nicht, *wie* man dieses Ziel erreicht.
3. Objektorientiert. Das Restaurant wird als System interagierender Objekte beschrieben.

### Lösung des Experimentes »Programmierparadigmen« (Abschnitt 1.6)

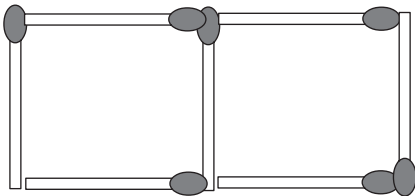


Abb. 1.8: Eine mögliche Lösung des Streichholz-Experiments

# Stichwortverzeichnis

\_\_abs\_\_() 297  
\_\_add\_\_() 296  
\_\_call\_\_() 299  
\_\_contains\_\_() 297  
\_\_debug\_\_ 584  
\_\_del\_\_() 297  
\_\_delitem\_\_() 297  
\_\_eq\_\_() 297  
\_\_float\_\_() 297  
\_\_ge\_\_() 297  
\_\_getitem\_\_() 297  
\_\_gt\_\_() 297  
\_\_init\_\_() 285  
\_\_le\_\_() 297  
\_\_len\_\_() 297  
\_\_lt\_\_() 297  
\_\_main\_\_ 586  
\_\_mod\_\_() 297  
\_\_mul\_\_() 297  
\_\_name\_\_ 586  
\_\_ne\_\_() 297  
\_\_neg\_\_() 297  
\_\_str\_\_() 297, 374  
\_thread 566

## A

Abbruch  
    Schleife 143  
Abfangen  
    Laufzeitfehler 145  
Abstrakter Datentyp 775  
Abstraktion 302  
access() 415  
add\_cascade() 549  
add\_checkbutton() 549  
add\_choice() 548  
add\_command() 549  
add\_radiobutton() 549  
add\_separator() 549  
Additive Farbmischung 460  
Adjazenzliste 781  
Aggregat 342  
Aggregation 342  
Aktueller Parameter 51, 157  
Algorithmus 27, 29  
Analyse 377  
    objektorientierte 683  
anchor 470

and 129  
Animation 569  
Anonymes Objekt 289  
Ansichtskarte 847  
Anweisung 49  
    bedingte 132  
    globale 169  
Anweisungsblock 72, 160  
Anwendungssoftware 28  
Apache-Webserver 614  
Apfelmännchen 507  
appendChild 749  
Arbeitsverzeichnis 414  
Argument 51, 157  
argv 409  
Arithmetischer Ausdruck 100  
Array 875  
    ändern 883  
Array-Funktion 889  
asctime() 425  
askokcancel() 553  
askopenfile() 552  
askopenfilename() 552  
asksaveasfile() 552  
asksaveasfilename() 552  
askyesno() 553  
assert 578  
AssertionError 578  
Assignment siehe Zuweisung  
Assoziation 334, 340  
    Aggregat 342  
    Kardinalität 344  
    reflexive 340  
Asynchrone Kommunikation 437  
Attribut 284, 289  
    dynamische Erzeugung 295  
    Klassenattribut 290  
    Objektattribut 290  
    öffentlich 290  
    privat 291  
    Zugriff 290  
AttributError 578  
Ausdruck 50, 127  
    arithmetischer 100  
    bedingter 136  
    regulärer 380  
Ausdruckanweisung 50  
Ausgabe 255

Ausnahme 145, 577  
Authentifizieren 686

## B

background 444  
backward() 181  
Basisklasse 303  
Baud 928  
Baum 183, 742  
bd 444, 446  
Bedingte Anweisung 132  
Bedingter Ausdruck 136  
Bedingung 123  
Befragung 709  
Begrenzungskasten 499  
Beliebige Anzahl von Parametern 176  
Benutzungsoberfläche 330, 479  
    grafische 437  
Betriebssystem 28, 413  
Bezeichner 47  
Beziehung 334  
bg 444  
Bildergalerie 519  
Binäre Suche 218  
binden 534  
Block 72  
BNF 1024  
bool 90, 91  
bool() 114  
borderwidth 444, 446  
Botschaft 52  
Bounding Box 499  
Box-Layout 801  
break 143  
Breakpoint 597  
Breitensuche 785  
Bubblesort 220  
Bug 596  
Built in function 54  
Button 451  
    Checkbox 459  
    Radiobutton 457  
    Submit-Button 609  
Bytestring 104

## C

Caesars Algorithmus 274  
Callable object 50  
Canvas 497  
    Display List 499  
    ID 499  
    Item 498  
    Koordinatensystem 499  
    Optionen der Items 500  
capitalize 362  
center 362  
cgi 610  
cgi.FieldStorage() 610

CGI-Skript 616  
    Aufbau 601  
    debuggen 613  
    erste Zeile 602  
    interaktive Webseite 605  
    Querystring 607  
    Verarbeitung von Eingabedaten 610  
cgitb 613  
cgitb.enable() 613  
Charts 223  
Chatroom 625  
chdir() 414  
Checkbox 459  
    Erscheinungsformen 459  
    Werte 459  
childNodes 748  
chmod() 415  
choice() 490  
clear() 242  
Client-Server-System 642  
close() 258, 263  
closed 263  
column 473  
columnspan 473  
ComboBox 808  
command 451, 461  
Compiler 30  
complex 90  
complex() 114  
COM-Port 926  
Container 90  
continue 144  
Cookie 632  
Coordinated Universal Time 424  
coords() 497  
count() 365  
CREATE TABLE 674  
create\_arc() 497  
create\_image() 497  
create\_line() 497  
create\_oval() 498  
create\_polygon() 498  
create\_rectangle() 498  
create\_text() 498  
create\_window() 498  
createsuperuser 989  
crop() 516  
CSRF-Token 1007  
CSV-Datei 936  
ctime() 425  
Current working directory 414  
Cursor 674  
cwd() 643

## D

Datei  
    anlegen 417  
    Merkmale abfragen 418

- suchen 414
- Datenbank 669
  - relationale 671
- Datenbank-Management-System 669
- Datenbanksystem 669
- Datenkommunikation 755
- Datentyp 87
  - abstrakter 775
- Datenverarbeitung
  - parallele 943
- datetime 428
- Datum 424
- DBMS siehe Datenbank-Management-System
- Debugger 596
- Debugging-Modus 583
- def 160
- Deklarativ 32
- delete() 455, 498, 545
- deselect() 458
- Dezimalbruch 94
- Dialog 845
- Dialogbox 552
- Dialog-Widget 846
- dict() 115
- Dictionary 111, 241, 672
  - Display 242
  - Operationen 241
  - Schlüssel 111
  - schrittweiser Aufbau 244
  - Zugriff auf Daten 245
- Digitales Multimeter 917
- Digitaluhr 571, 832
- Disjunktion 130
- Display List 499
- Divide and conquer 221
- Division 98
- Django 967
  - Administration 989
  - Datenbankanbindung 974
  - Klassenattribute eines Modells 984
  - Manager 981, 986
  - Modell erstellen 974
  - Modelle aktivieren 976
  - Route 972
  - Server starten 970
  - View 971, 995
- DMM 917
- Docstring 160, 194, 701
- doctest 701, 711
- Document Object Model siehe DOM
- DOM 743
  - createTextNode() 751
  - Document 744, 750
  - documentElement 751
  - Element 750
  - getElementsByName() 751
  - tagName 751
  - Text 750

- Download 643
- Drag&Drop 536
- Duck-Typing 59
- dump() 269
- Dynamische Typisierung 59

## E

- EBNF-Grammatik 1024
- Editieren 65
- Eingabe 255
- Eingabefeld 608
- Einrückung 72
- Einwegfunktion 686
- elif 134
- Eliza 377
- Ellipse 712
- E-Mail 657
- E-Mail-Client 657
- end() 389
- Endlosschleife 137
- Endrekursion 183
- endswith() 363
- Entity-Relationship-Diagramm 670
- Entry 455
  - delete() 455
  - get() 455
  - Passworteingabe 455
  - show 455
- Entwicklungsumgebung 556
- environ 421
- ER-Diagramm siehe Entity-Relationship-Diagramm
- Erweiterte Zuweisung 59
- Escape-Sequenz 103
- EVA-Prinzip 73
- Event 813
  - binden 534
  - Taste 530
- Eventhandler 532
- Event-Modifizierer 530
- Event-Sequenz 528
- exc\_info() 410
- except 146
- Exception siehe Ausnahme
- exec\_prefix 410
- executable 410
- execute() 674
- exists() 418, 419
- exit() 410
- exitfunc 410
- expand 470
- Exponentialschreibweise 94
- Externes Modell 670
- Extreme Programming 701

## F

- Fallunterscheidung 134
- False 90, 91

- Farbe 446
  - Farbmischung
    - additive 460
  - Farbverlauf 818
  - Farbwechselleuchte 813
  - Fehler 78, 248, 311, 577
    - logischer 79, 577
    - Syntaxfehler 79, 577
  - Fenster
    - mehrere Fenster 556
  - fg 444
  - FieldStorage 610
  - FIFO 778
  - File
    - laden 260
    - speichern 258
  - File Transfer Protocol 642
  - fill 470
  - finally 266
  - find\_all() 498
  - find\_closest() 498
  - find\_overlapping() 498
  - find() 365
  - findall() 384, 386
  - Finden
    - gieriges 387
    - nicht gieriges 387
  - firstChild 748
  - Flake8 195
  - Flash 755
  - Flesch-Analyse 400
  - FLOAT 674
  - float 90
  - float() 113
  - flush() 263
  - Folge
    - rekursive 143
  - Font 445, 814
  - font 444
  - for 139
  - foreground 444
  - Formaler Parameter 160
  - Formatierung
    - Tabelle 272
  - Formatierungsoperator % 369
  - Form-Layout 825
  - Foto 508
  - Frame 463
  - Fremdschlüssel 672
  - from 54
  - from\_ 461
  - FTP 642
  - ftplib 642
  - FTP-Server 643, 645
  - Funktion 50, 157
    - als Objekt 190
    - Aufruf 157
    - Ausführung 166
    - beliebige Anzahl von Parametern 176
    - Definition 160
    - Kopf 160
    - Körper 160
    - Lambda-Form 191
    - lokale Funktion 177
    - Parameter 157
    - Parameterübergabe 170
    - rekursive 178
    - Schlüsselwort-Argument 174
    - Seiteneffekt 169
    - voreingestellter Parameterwert 172
  - Funktionskopf 160
  - Funktionskörper 160
  - Funktionsplotter 520
- ## G
- Ganze Zahl 92
  - Geheimnisprinzip 302
  - Generator 227
  - Generatordruck 228
  - Generatorfunktion 228
  - geopy 866
  - Geräte-Manager 927
  - Geschäftsprozess 331
  - Geschäftsprozessdiagramm 331
  - Geschwister 742
  - Gesprächsroboter 378
  - get() 455, 545
  - getatime() 418, 419
  - getcwd() 414
  - getenv() 421
  - getfirst() 612
  - getlist() 612
  - getmtime() 418, 419
  - getPixel() 516
  - getrefcount() 412
  - getrefcount(object) 410
  - getsize() 418
  - getvalue() 612
  - Gieriges Finden 387
  - Gleich 46
  - Gleitkommazahl 94
  - global 169
  - Global Interpreter Lock (GIL) 944
  - Globaler Name 166
  - Globals 597
  - gmtime() 425
  - Go 597
  - Grafik 497
  - Grammatik 48, 1024
  - Graph 779
  - grid() 472
  - GUI 437, 476
- ## H
- Hardware 28

- hasAttributes() 749
- hasChildNodes() 749
- height 444, 448
- Hexadezimalzahl 93
- hidden 609
- Hintergrundbild 512
- Hotkey 44
- HTML
  - Checkbox 609
  - Eingabefeld 608
  - Formular 606
  - Passworтеingabe 608
  - Radiobutton 608
  - Submit-Button 609
  - versteckte Variablen 609
- HTML-Formular 606
  - Checkbox 609
  - Eingabefeld 608
  - Radiobutton 608
- HTTP 649
- HTTP-Paket 601
- HTTP-Server 600, 605
- Hypertext Transfer Protocol 649

## I

- Icon 511
- id() 127
- Identifizier 47
- Identisch 46
- Identität 46
- IDLE 43, 65
- if 133
- if-else 133
- IGNORECASE 384
- image 444
- Imperativ 33
- import 54
- in 127, 203
- Index 544
- IndexError 578
- indicatoron 458, 459
- Informatik 27
- information hiding 302
- insert() 545
- insertBefore() 749
- Installation 40
- Instanz 36, 287
- INT 674
- int 90, 92
- int() 113
- Interaktive Webseite 605
- Interaktiver Modus 39, 42
- Internes Modell 669
- Internet-Programmierung 641
- Interpreter 30
- IOError 578
- isalnum() 363

- isalpha() 363
- isdigit() 363
- isdir() 414
- isfile() 414
- islower() 363
- isupper() 363
- Item 498
- itemcget() 498
- itemconfigure() 498
- items() 242
- Iteration 139
- Iterator 230
- Iterierbar 90

## J

- justify 444, 543

## K

- Kalender 837
- Kalenderdatum 428
- Kamerabild 843
- Kante 779
- Kardinalität 344
- Keller 775
- Key siehe Schlüssel
- KeyError 578
- keys() 242, 612
- Keyword siehe Schlüsselwort
- Kind 742
- Klasse 36, 283, 285
  - Beziehung zwischen Klassen 334
  - Definition 285
  - Dokumentation 310
  - Fehler 311
  - Konstruktor 285
  - Kopf 285
  - Oberklasse 285
  - Programmierstil 309
  - Spezialisierung 304
- Klassenattribut 284
- Klassenstruktur 330
- Knoten 742, 779
- Kollektion 90
- Kommandozeilen-Argument 273
- Kommentar 69, 77
- Kommunikation 437, 641
  - asynchrone 437
- Komplexe Zahl 95
- Konjunktion 129
- Konkatenation 108
- Konstruktor 285
- Kontrollstruktur 123
  - Endlosschleife 137
  - try 145
- Kontrollvariable 453
- Konzeptuelles Modell 669
- Kopie

flache 215  
 tiefe 215  
 Kreisdiagramm 502  
 Kunststoff 464  
 Kurze Zeichenkette 102

## L

Label 451  
 label 461  
 Lambda-Form 191  
 Landesumweltamt 660  
 Lange Zeichenkette 103  
 lastChild 748  
 Laufzeitfehler  
     abfangen 145  
 Laufzeitsystem 409  
 Layout 77, 469  
 Layout-Fehler 471  
 Leerraum 449  
 Leichtgewichtprozess 566  
 Lichtschalter 463  
 LIFO 775  
 List comprehension 211  
 list() 115  
 listdir () 414  
 Liste 106, 210  
     erzeugen 210  
     list comprehension 211  
     Modellierung 223  
     Operationen 210  
     sortieren 216  
     verändern 213  
 Literal 45, 87  
 ljust 362  
 Locals 597  
 localtime() 425  
 Lock-Mechanismus 957  
 Log 587  
 Log-Datei 587  
 Logger-Objekt 594  
 logging 587  
 Logging-Level 589  
 login() 657  
 Logischer Fehler 79, 577  
 Lokaler Name 166  
 long 90  
 lower() 363  
 lstrip([chars]) 364

## M

Mandelbrotmenge 507  
 map() 192  
 Maskieren 383  
 Master-Slave-Hierarchie 442  
 match() 384  
 Match-Objekt 389  
 Matrix 875

Matrizenmultiplikation 888  
 Medianfilter 903  
 Mehrere Fenster 556  
 Memory 535  
 Menge 110, 127  
 Menu 548  
     Methoden 548  
     Optionen der Choices 549  
 Menü 547  
 Mergesort 590  
 MessageBox 552  
 Metasprache 739  
 Methode 35, 52, 284, 295  
 Migration 977  
 MIT-License 1048  
 mkdir() 417  
 mktime() 425  
 mod\_wsgi 618  
 mode 263  
 Modell  
     externes 670  
     internes 669  
     konzeptuelles 669  
 Modellieren 329  
 Model-View-Template 967  
 Modul 321  
     importieren 323  
     kompilieren 326  
     Programmierstil 327  
     speichern 323  
     Zugang sicherstellen 325  
 modules 410  
 Modulo 99  
 Modus  
     Debugging 583  
     interaktiver 39, 42  
     optimierter 583  
 move() 498  
 Multimedial 437  
 Multimeter  
     digitales 917  
 Multiplikation 97  
 Musical 343

## N

Nachbedingung 578  
 Name 47  
     globaler 166  
     lokaler 166  
 NameError 578  
 Navigieren 643  
 Negation 128  
 Netiquette 645  
 next() 230  
 nextSibling 749  
 Nichtterminalsymbol 1024  
 Node 748



nodeType 749

None 91

NoneType 91

not 128

not in 127

NumPy 875

**O**

Oberklasse 285

Object Relational Mapping (ORM) 978

Objekt 45, 87, 283

Abstraktion 302

anonymes 289

Attribut 284

Botschaft 52

callable object 50

für reguläre Ausdrücke 384

Geheimnisprinzip 302

Identität 46

Instanz 287

laden 269

Match-Objekte 389

Methode 52, 295

Name 47

speichern 268

textuelle Repräsentation 374

Typ 45

Verkapselung 302

Wert 45

Zustand 289

Objektattribut 290

Objektdiagramm 289

Objektorientierte Analyse 329, 683

Objektorientierte Programmierung 33

Objektorientierte Software-Entwicklung 329

Objektorientierter Entwurf 330

Objektorientiertes Modellieren 329

Objektsymbol (UML) 289

offvalue 459

Online-Abstimmung 636

Online-Redaktionssystem 681

Online-Shop 661

onvalue 459

OOA siehe Objektorientierte Analyse

OOD siehe Objektorientierter Entwurf

OOP siehe Objektorientierte Programmierung

open() 257

OpenStreetView 866

OpenWeatherMap 769

Operator

- 97

% 369

+ 97

in 127, 203

logischer Operator 128

Priorität 101

überladen 296

Vergleichsoperatoren 124, 894

Vorzeichenoperator 97

Optimierter Modus 583

Option 273

or 130

orient 461, 546

os 413

Over 597

Ozonkonzentration 660

**P**

pack() 469

Packer 469

padx 444, 470, 473

pady 444, 470, 473

Paradigma 32

Parallele Datenverarbeitung 943

Parallele Programmierung 943

Parameter 157

aktueller 51, 157

formaler 160

Parameterliste 160

Parameterübergabe 170, 171

Parameterwert

voreingestellter 172

parentNode 749

parse() 747

parseString() 747

Passende Zeichenkette 380

Passwort 455

Passworтеingabefeld 608

path 410

pendown() 181

Performance-Analyse 721, 729

Pfad 260, 261

absolut 260

relativ 261

Pfadbezeichnung 259

PhotoImage 505

copy() 505

height() 505

put 506

width() 506

write() 506

pickle 268

PIL.Image

crop() 516

resize() 517

size 517

pip 1045

Pixelgrafik 506

platform 410

Platonisches Schriftzeichen 365

Playlist 855

Polymorphie 296

Polymorphismus 296

pop() 776

Portable Pixmap 514  
 Positionsargument 175  
 Potenz 96  
 PPM 514  
 previousSibling 749  
 Primfaktor 581  
 print 53  
 Priorität 101  
 Problem 76  
 Problemspezifikation 76  
 Profiler 721  
 Programm 27  
 Programmieren  
     objektorientiertes 33  
 Programmierparadigma 32  
 Programmierstil 76, 132, 193, 309, 327  
 Programmierung  
     parallele 943  
 Programmverzweigung 132  
 Prompt 43  
 Protokoll 641  
 Prozess 565  
     unterbrechen 428  
 Pulldown-Menü 548  
 push() 776  
 put() 506  
 putenv() 421  
 PyPI 1046  
 PyQt5 795  
 PySerial 926  
 Python Package Index 1046  
 Python-Homepage 39  
 Python-Interpreter 42

## Q

QCalendarWidget 837, 842  
 QCamera 845  
 QCheckbox 805  
 QFileDialog 847  
 QIcon 819  
 QInputDialog 847  
 QLabel 803  
 QMessageBox 819  
 QPixmap 803  
 QPlaylist 858  
 QRadiobutton 805  
 Qt 795  
 QTextEdit 837  
 Qt-Fenster 811  
 QTimer 820  
 Qt-Layout 822  
 Qt-Widgets 802  
 Qualifizierer 528  
 Querystring 607  
 Queue 778  
 Quicksort 221, 586  
 Quit 597

quit() 657  
 QMediaPlayer 857  
 QVideoWidget 862  
 QViewFinder 843  
 QWebView 827

## R

Radiobutton 457  
     command 457  
     Erscheinungsform 457  
     Selektion 458  
     variable 458  
 Rahmen 446  
 raise 584  
 range() 140  
 Raster-Layout 472  
 Raumplan 781  
 re 389  
 read() 260, 263  
 readline() 263  
 Regel 48, 1025  
 Regulärer Ausdruck 380  
 Rekursionstiefe 188  
 Rekursive Folge 143  
 Rekursive Funktion 178  
 Relation 671  
 Relationale Datenbank 671  
 relief 444  
 removeChild(oldChild) 749  
 Rendern 1000  
 replace() 365  
 requests 649  
 requests.Response 650  
 resolution 461  
 Response 650  
 reST 1049  
 reStructuredText 1049  
 retrbinary() 643  
 retrlines() 643  
 rjust 362  
 Rollbalken 546  
 row 473  
 rowspan 473  
 rstrip() 364  
 run module 66  
 run() 569

## S

Scale 461  
 Schiffe versenken 484  
 Schlange 778  
 Schleife  
     Abbruch 143  
 Schlüssel 111  
 Schlüsselwort 48  
 Schlüsselwort-Argument 174  
 Schriftzeichen

- platonische 365
- Scrollbar 546
- sdist 1051
- search() 385
- see() 545
- seek() 263
- Seiteneffekt 169
- Sekundenformat 425
- Selbstständig 184
- Selbstdokumentation 585
- select() 458
- SELECT-Anweisung 675
- Semantik 30
- sendmail() 657
- Sequenz 101
  - gemeinsame Operationen 203
  - in 203
  - Konkatenation 108, 203
  - Länge 109, 204
  - not in 203
  - Slicing 205
  - veränderbar und unveränderbar 109
  - Vervielfältigung 108
  - Zugriff 107
- serial 928
- set\_debuglevel() 657
- setup.py 1047
- Shell 42
- Shell-Fenster 66
- Shortcut 44
- show 455
- showerror() 552
- showinfo() 552
- showturtle() 181
- showvalue 462
- showwarning() 553
- Sicht 670
- Sichtbarkeit 290
- side 470
- Sierpinski-Dreieck 184
- Signal 804
- Simple Mail Transfer Protocol 657
- SimpleCookie 632
- Size Policy 858
- Skript 65
  - Ausführung beenden 413
- sleep() 425
- Slicing 205, 207, 882
- slider 546
- Slot 805
- SMTP 657
- Software 28
- Sommerzeit 426
- Sortieren 216
- Sortierverfahren 219
- Soziogramm 791
- Speech SDK 391
- speed() 181
- Speichern
  - Objekte 268
- Spezialisierung 304
- Spirale 181
- split() 364, 385, 387
- splitlines() 364
- Sprachsynthese 391
- SQL 673
- SQL-Injection 680
- sqlite3 673
- Stack 775
- Stand-alone-Skript 321
- Standardausgabe 411
- Standardeingabe 411
- Stapel 775
- start\_new\_thread() 566, 567
- start() 389, 569
- Startsymbol 1025
- Statement siehe Anweisung
- staticmethod() 300
- Statische Methode 300
- stderr 410
- stdin 410
- stdout 410
- Steganografie 517, 897
- Step 597
- Sternenhimmel 725
- sticky 473
- StopIteration 230
- str() 114
- Stream 255
  - lesen und schreiben 263
- String 102
  - kurze Zeichenkette 102
  - lange Zeichenkette 103
  - siehe auch Zeichenkette
- strip() 364
- Stylesheet 816
- sub() 385, 388
- Subklasse 303
- Submit-Button 609
- Suche
  - binäre 218
- Suchroboter 645
- Synchronisation 956
- Syntax 30
- Syntaxfehler 79
- sys 409
- sys.argv 274
- sys.path 325
- sys.stdin 270
- sys.stdout 270
- Systemfunktion 409
- Systemsoftware 28
- Systemumgebung 410

**T**

Tabelle 272  
 tabs 543  
 tag\_bind() 498  
 Taschenrechner 473  
 Tastenkombination 44  
 Tastenname 530  
 TCP/IP-Modell 641  
 tell() 263  
 Terminalsymbol 1024  
 Test  
     Turing 377  
     Vorkommenstest 378  
 Test Driven Development 701  
 Testen 321, 701  
 TestPyPI 1046  
 Testreihe 713  
 Text  
     Index-Formate 545  
     Methoden 544  
     Optionen 543  
     Rollbalken 546  
 text 444  
 Textanalyse 378  
 Texteditor 544, 550  
 textvariable 444  
 Thread 565, 568, 943  
 threading 566, 568  
 time 54  
 time() 425  
 timedelta 431  
 title() 451  
 Tk 450  
 tkFileDialog 552  
 Tkinter 437, 497  
 tkMessageBox 552  
 top() 776  
 toprettyxml() 747  
 toxml() 747  
 Trennstring 387  
 trough 546  
 True 90, 91  
 try 145  
 Tuning 721, 729  
 Tupel 105, 209  
 tuple() 115  
 Turing-Test 377  
 Türme von Hanoi 198  
 Turtle-Grafik 180  
 Typ 45  
     None 91  
 TypeError 578  
 Typumwandlung 111

**U**

Überladen 296

Umgebungsvariable 421  
 UML 289  
 UML-Klassendiagramm 336  
 underline 444  
 Unicode 365  
     utf-8 740  
 unicode() 114  
 unittest 713  
 Unix  
     Programmausführung 68  
 unlink() 749  
 Unterklasse 303  
 update() 245  
 upper() 363  
 URL 600  
 URL-Pattern 996  
 USB-to-Serial 927  
 use case 331  
 User-Agent 652  
 UTC 424  
 utf-8 740

**V**

ValueError 578  
 values() 242  
 VARCHAR 674  
 Variablenname 56  
 Vektor 875  
 Verarbeitungsschicht 642  
 Verbinden  
     Zeilen 71  
 Vererbung 303  
 Verfeinerung 162  
 Vergleich 123  
 Vergleichsoperator 124, 894  
 Verkapselung 302  
 Verzeichnis 257  
     anlegen 417  
     Merkmale abfragen 418  
     suchen 414  
 Verzeichnisbaum 422  
 Verzweigung 133  
 Videoplayer 851  
 View 971, 995  
 View-Funktion 967, 995  
 Vokabeltrainer 246  
 Vorbedingung 578  
 Voreingestellter Parameterwert 172  
 Vorkommenstest 378

**W**

Wahrheitswert 91  
 Währungsumrechner 463  
 walk() 422  
 Webbrowser 798  
 Webseite 605

Wegenetz 782  
 Wegsuche 785  
 Weltkarte 864  
 Wert 45  
 Wetterdaten 770  
 while 136  
 Widerstandsthermometer 934  
 Widget 441, 450
 

- Button 451
- einfach 441
- Farbe 446
- Font 445
- Größe 447
- konfigurieren 444
- Layout 469
- Leerraum 449
- Master-Slave-Hierarchie 442
- Methoden 450
- Option 443, 444
- Rahmen 446
- Text 543

 width 444, 448  
 Wiederholung 136  
 Windows 67  
 with 267  
 Wort des Jahres 704  
 Wörterbuch 111, 330  
 Wörterraten 485  
 wrap 543  
 write() 258, 263  
 WSGI 616  
 WSGI-Skript 690

## X

XML 739
 

- Attribute 752

Datenkommunikation 755
 

- Tags 741

 xml.dom.minidom 746  
 xscrollcommand 543

## Y

yield 229  
 yscrollcommand 543

## Z

Zahl
 

- ganze 92
- Gleitkommazahl 94
- Hexadezimalzahl 93
- komplexe 95

 Zeichenkette 102, 361
 

- formatieren 362
- kurze 102
- lange 103
- passende 380
- zerlegen 387

 Zeile
 

- Einrückung 72
- verbinden 71

 Zeilenstruktur 70  
 Zeit 424  
 Zeitkomplexität 219  
 Zeitstring 427  
 Zeittupel 426  
 Zeitzone 430  
 ZeroDivisionError 578  
 Zugriffsrecht 415  
 Zuweisung 55
 

- erweitert 59

 Zuweisungsoperator 56