



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"
```

# Inhaltsverzeichnis

|   |           |
|---|-----------|
| <b>Einleitung</b> .....   | 21        |
| Warum Python? .....   | 21        |
| Python 3 .....  | 21        |
| An wen wendet sich dieses Buch? .....                                   | 21        |
| Inhalt und Aufbau .....   | 22        |
| Hinweise zur Typographie .....  | 23        |
| Programmbeispiele .....   | 24        |
| <b>1 Grundlagen</b> .....   | <b>25</b> |
| 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        |
| <b>2 Der Einstieg – Python im interaktiven Modus</b> .....              | <b>37</b> |
| 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 .....   | 44        |
| 2.5 Hintergrund: Syntax-Regeln für Bezeichner .....                     | 45        |
| 2.6 Schlüsselwörter .....   | 46        |
| 2.7 Anweisungen .....   | 46        |
| 2.7.1 Ausdruckanweisungen .....   | 47        |
| 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        |
| <b>3</b> | <b>Python-Skripte</b> .....                     | <b>63</b> |
| 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        |
| <b>4</b> | <b>Standard-Datentypen</b> .....                | <b>83</b> |
| 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        |
| <b>5</b> | <b>Kontrollstrukturen</b> .....                          | <b>117</b> |
| 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        |
| <b>6</b> | <b>Funktionen</b> .....                                  | <b>151</b> |
| 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. . . . .                                    | 188        |
| 6.15.4   | Docstrings . . . . .   | 188        |
| 6.16     | Aufgaben . . . . .   | 189        |
| 6.17     | Lösungen . . . . .   | 192        |
| <b>7</b> | <b>Sequenzen, Mengen und Generatoren . . . . .</b>                 | <b>197</b> |
| 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        |
| <b>8</b>  | <b>Dictionaries</b> .....  | <b>235</b> |
| 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 –<br>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        |
| <b>9</b>  | <b>Ein- und Ausgabe</b> .....  | <b>249</b> |
| 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        |
| <b>10</b> | <b>Definition eigener Klassen</b> .....                                    | <b>277</b> |
| 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        |
| <b>11</b> | <b>Klassenbibliotheken in Modulen speichern .....</b>                             | <b>315</b> |
| 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        |
| <b>12</b> | <b>Objektorientiertes Modellieren .....</b>                                       | <b>323</b> |
| 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<br>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        |

|           |  |     |
|-----------|--|-----|
| <b>13</b> | <b>Textverarbeitung</b> .....  | 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<br>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 |
| <b>14</b> | <b>Systemfunktionen</b> .....  | 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<br>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        |
| <b>15</b> | <b>Grafische Benutzungsoberflächen mit tkinter . . . . .</b> | <b>431</b> |
| 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        |
| <b>16</b> | <b>Layout . . . . .</b>                                      | <b>463</b> |
| 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        |
| <b>17</b> | <b>Grafik</b>  | <b>491</b> |
| 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        |
| <b>18</b> | <b>Event-Verarbeitung</b>  | <b>519</b> |
| 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        |
| <b>19</b> | <b>Komplexe Benutzungsoberflächen</b>                                  | <b>537</b> |
| 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        |
| <b>20</b> | <b>Threads</b> . . . . .                                    | <b>559</b> |
| 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        |
| <b>21</b> | <b>Fehler finden und vermeiden</b> . . . . .                | <b>571</b> |
| 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        |
| <b>22</b> | <b>Dynamische Webseiten – CGI und WSGI</b> . . . . .        | <b>593</b> |
| 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        |
| <b>23</b> | <b>Internet-Programmierung</b>                                     | <b>635</b> |
| 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        |
| <b>24</b> | <b>Datenbanken</b>   | <b>663</b> |
| 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        |
| <b>25</b> | <b>Testen und Tuning . . . . .</b>                               | <b>691</b> |
| 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        |
| <b>26</b> | <b>XML und JSON . . . . .</b>                                    | <b>729</b> |
| 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<br>gut lesbar? . . . . . | 756        |
| 26.9.4    | Projekt: Verarbeitung von Wetterdaten . . . . .                                 | 759        |
| 26.10     | Aufgaben . . . . .  | 762        |
| 26.11     | Lösungen . . . . .  | 763        |
| <b>27</b> | <b>Modellieren mit Kellern, Schlangen und Graphen . . . . .</b>                 | <b>765</b> |
| 27.1      | Stack (Keller, Stapel) . . . . .  | 765        |
| 27.2      | Queue (Schlange) . . . . .  | 768        |
| 27.3      | Graphen . . . . .   | 769        |
| 27.4      | Aufgaben . . . . .  | 779        |
| 27.5      | Lösungen . . . . .  | 781        |
| <b>28</b> | <b>Benutzungsoberflächen mit Qt . . . . .</b>                                   | <b>785</b> |
| 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      | Checkboxes 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        |

|           |   |     |
|-----------|---|-----|
| <b>29</b> | <b>Multimediaanwendungen mit Qt</b> . . . . .                             | 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 |
| <b>30</b> | <b>Rechnen mit NumPy</b> . . . . .  | 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 |
| <b>31</b> | <b>Messdaten verarbeiten</b> . . . . .                                    | 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        |
| <b>32</b> | <b>Parallele Datenverarbeitung . . . . .</b>                  | <b>933</b> |
| 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        |
| <b>33</b> | <b>Django . . . . .</b>                                       | <b>955</b> |
| 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 Suche nach Ideen . . . . .                                  | 999  |
| 33.15    | Lösung . . . . .  | 1000 |
| <b>A</b> | <b>Anhang</b> . . . . .   | 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>B</b> | <b>Glossar</b> . . . . .  | 1013 |
| <b>C</b> | <b>Download der Programmbeispiele</b> . . . . .                     | 1025 |
| <b>D</b> | <b>Ein Python-Modul veröffentlichen: PyPI</b> . . . . .             | 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 |
|          | <b>Stichwortverzeichnis</b> . . . . .                               | 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 »mögeln« 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 Programmierertechnik 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.

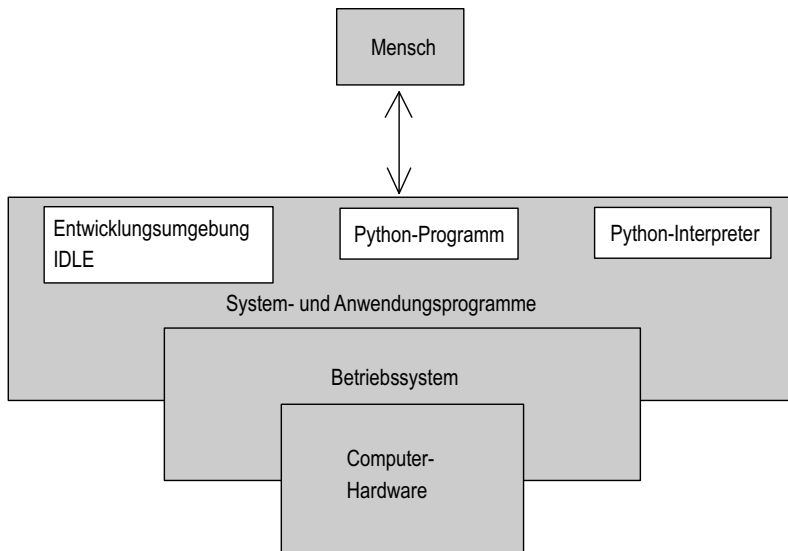


Abb. 1.1: Komponenten eines Computer-Systems

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 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 »verstehen« – 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.

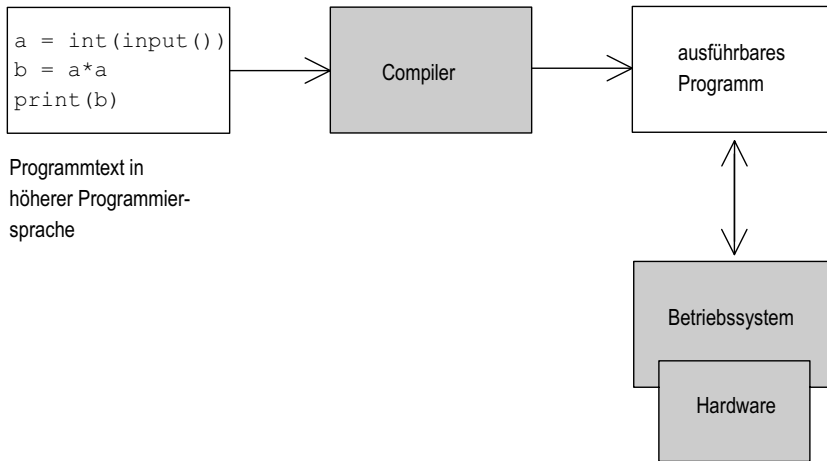
## 1.5 Interpreter und Compiler

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.

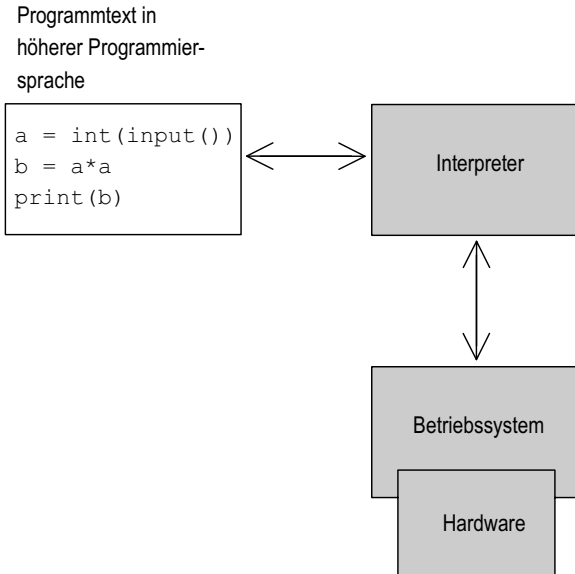


Abb. 1.4: Arbeitsweise eines Interpreters

## 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 zurecht kommen.

### 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.

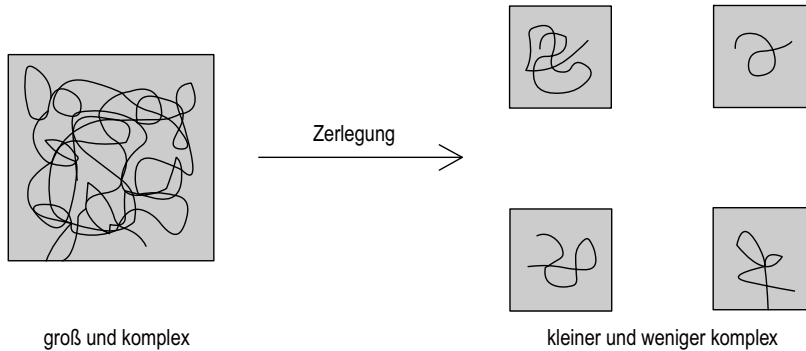
Zum Schluss sei noch das Paradigma der *funktionalen* Programmierung erwähnt. Mit funktionalen Programmiersprachen wie z.B. *Haskell* oder *Scheme* kann man ein Programm als (mathematische) Funktion definieren. Einfache vorgegebene Funktionen werden zu einer komplexen Funktion verknüpft, die das Gewünschte 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.

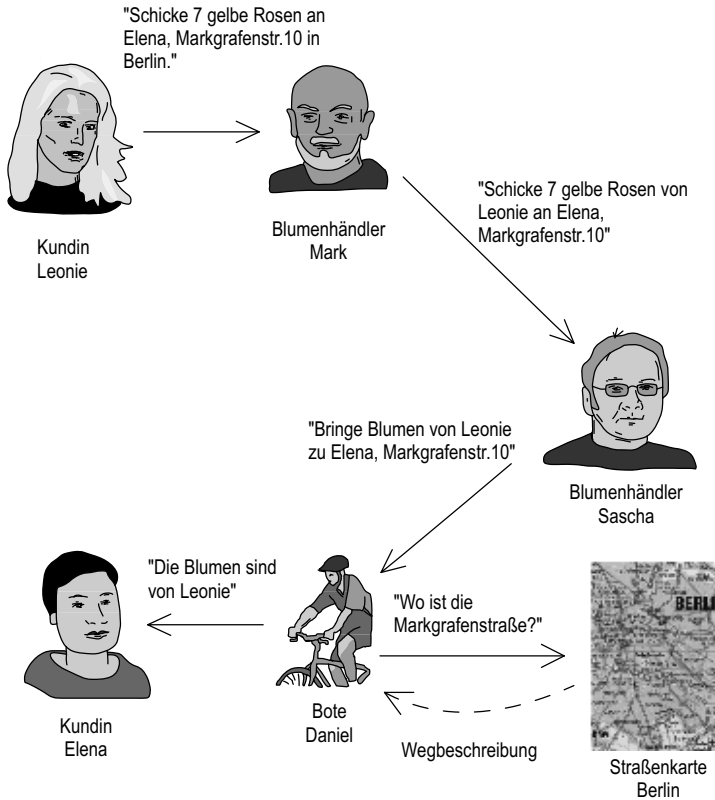


Abb. 1.6: Objektorientiertes Modell eines Blumenversandsystems

### 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.

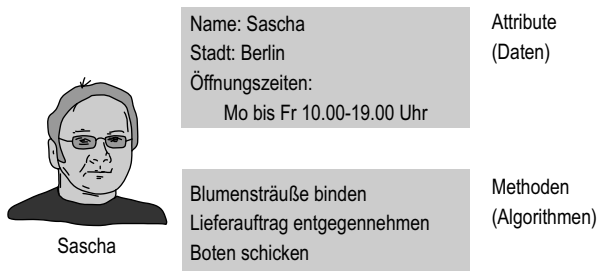


Abb. 1.7: Objekte besitzen Attribute und beherrschen Methoden.

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)

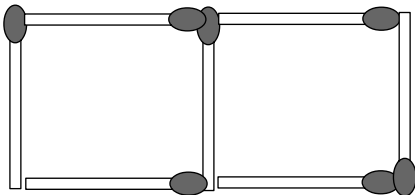


Abb. 1.8: Eine mögliche Lösung des Streichholz-Experiments

# Stichwortverzeichnis

\_\_abs\_\_() 291  
\_\_add\_\_() 290  
\_\_contains\_\_() 291  
\_\_debug\_\_ 578  
\_\_del\_\_() 291  
\_\_delitem\_\_() 291  
\_\_eq\_\_() 291  
\_\_float\_\_() 291  
\_\_ge\_\_() 291  
\_\_getitem\_\_() 291  
\_\_gt\_\_() 291  
\_\_init\_\_() 280  
\_\_le\_\_() 291  
\_\_len\_\_() 291  
\_\_lt\_\_() 291  
\_\_main\_\_ 580  
\_\_mod\_\_() 291  
\_\_mul\_\_() 291  
\_\_name\_\_ 580  
\_\_ne\_\_() 291  
\_\_neg\_\_() 291  
\_\_nonzero\_\_() 291  
\_\_str\_\_() 291, 368  
\_thread 560

## A

Abbruch  
    Schleife 137  
Abfangen  
    Laufzeitfehler 139  
Abstrakter Datentyp 765  
Abstraktion 296  
access() 409  
add\_cascade() 543  
add\_checkbutton() 543  
add\_choice() 543  
add\_command() 543  
add\_radiobutton() 543  
add\_separator() 543  
Additive Farbmischung 454  
Adjazenzliste 771  
Aggregat 336  
Aggregation 336  
Aktueller Parameter 49, 151  
Algorithmus 25, 27  
Analyse 371  
    objektorientierte 674  
anchor 464

and 123  
Animation 563  
Anonymes Objekt 283  
Ansichtskarte 837  
Anweisung 46  
    bedingte 126  
        global 163  
Anweisungsblock 70, 154  
Anwendungssoftware 26  
Apache-Webserver 608  
Apfelmännchen 501  
appendChild 739  
Arbeitsverzeichnis 408  
Argument 49, 151  
argv 403  
Arithmetischer Ausdruck 96  
Array 865  
    ändern 873  
Array-Funktion 879  
asctime() 419  
askokcancel() 547  
askopenfile() 546  
askopenfilename() 546  
asksaveasfile() 547  
asksaveasfilename() 547  
askyesno() 547  
assert 572  
AssertionError 572  
Assignment siehe Zuweisung  
Assoziation 328, 334  
    Aggregat 336  
    Kardinalität 338  
    reflexiv 334  
Asynchrone Kommunikation 431  
Attribut 278, 283  
    dynamische Erzeugung 289  
    Klassenattribut 284  
    Objektattribut 284  
    öffentlich 284  
    privat 285  
    Zugriff 284  
AttributError 572  
Ausdruck 47, 121  
    arithmetischer 96  
    bedingter 130  
    regulärer 374  
Ausdruckanweisung 47  
Ausgabe 249

Ausnahme 139, 571  
 Authentifizieren 676

## B

background 438  
 backward() 175  
 Basisklasse 297  
 Baud 918  
 Baum 177, 732  
 bd 438, 440  
 Bedingte Anweisung 126  
 Bedingter Ausdruck 130  
 Bedingung 117  
 Befragung 699  
 Begrenzungskasten 493  
 Beliebige Anzahl von Parametern 170  
 Benutzungsoberfläche 324, 473  
     grafische 431  
 Betriebssystem 26, 407  
 Bezeichner 45  
 Beziehung 328  
 bg 438  
 Bildergalerie 513  
 Binäre Suche 212  
 binden 528  
 Block 70  
 BNF 1008  
 bool 86, 87  
 bool() 110  
 borderwidth 438, 440  
 Botschaft 50  
 Bounding box 493  
 Box-Layout 791  
 break 137  
 Breitensuche 775  
 Bubblesort 214  
 Bug 590  
 Built in function 52  
 Button 445  
     Checkbutton 453  
     Radiobutton 451  
     Submit-Button 603  
 Bytestring 100

## C

Caesars Algorithmus 268  
 Callable object 48  
 Canvas 491  
     Display List 493  
     ID 493  
     Item 492  
     Koordinatensystem 493  
     Optionen der Items 494  
 capitalize 356  
 center 356  
 cgi 604  
 cgi.FieldStorage() 604  
 CGI-Skript 610

Aufbau 595  
 debuggen 607  
 erste Zeile 597  
 interaktive Webseite 599  
 Querystring 601  
 Verarbeitung von Eingabedaten 604  
 cgitb 607  
 cgitb.enable() 607  
 Charts 217  
 Chat Bot 371  
 Chatroom 619  
 chdir() 408  
 Checkbutton 453  
     Erscheinungsformen 453  
     Werte 453  
 childNodes 738  
 chmod () 409  
 choice() 484  
 clear() 236  
 Client-Server-System 636  
 close() 252, 256, 644  
 closed 256  
 column 467  
 colspan 467  
 ComboBox 798  
 command 445, 455  
 Compiler 28  
 complex 86  
 complex() 110  
 COM-Port 916  
 Container 86  
 continue 138  
 Cookie 626  
 Coordinated Universal Time 418  
 coords() 491  
 count() 359  
 CREATE TABLE 668  
 create\_arc() 491  
 create\_image() 491  
 create\_line() 491  
 create\_oval() 492  
 create\_polygon() 492  
 create\_rectangle () 492  
 create\_text() 492  
 create\_window() 492  
 createsuperuser 976  
 crop() 510  
 CSRF-Token 993  
 CSV-Datei 926  
 ctime() 419  
 Current working directory 408  
 Cursor 667  
 cwd() 637

## D

Datei  
     anlegen 411  
     externe 250

- Merkmale abfragen 412
    - suchen 408
  - Datenbank 663
    - relationale 665
  - Datenbank-Management-System 663
  - Datenbanksystem 663
  - Datenkommunikation 745
  - Datentyp 83
    - abstrakter 765
  - Datenverarbeitung
    - parallele 933
  - datetime 422
  - Datum 418
  - DBMS siehe Datenbank-Management-System
  - Debugger 590
  - Debugging-Modus 577
  - def 154
  - Deklarativ 30
  - delete() 449, 492, 539
  - deselect() 452
  - Dezimalbruch 90
  - Dialog 835
  - Dialogbox 546
  - Dialog-Widget 836
  - dict() 111
  - Dictionary 107, 235, 666
    - Display 236
    - Operationen 235
    - Schlüssel 107
    - schrittweiser Aufbau 238
    - Zugriff auf Daten 239
  - digest() 677
  - Digitales Multimeter 907
  - Digitaluhr 565, 822
  - Disjunktion 124
  - Display List 493
  - Divide and conquer 215
  - Division 94
  - Django 955
    - Administration 976
    - Datenbankanbindung 961
    - Klassenattribute eines Modells 971
    - Manager 969, 973
    - Modell erstellen 962
    - Modelle aktivieren 963
    - Route 959
    - Server starten 958
    - View 959, 982
  - DMM 907
  - Docstring 154, 188, 691
  - doctest 691, 701
  - Document Object Model siehe DOM
  - DOM 733
    - createTextNode() 741
    - Document 734, 740
    - documentElement 741
    - Element 740
    - getElementsByName() 741
    - tagName 741
    - Text 740
  - Download 637
  - Drag&Drop 530
  - Duck-Typing 57
  - dump() 262
  - Dynamische Typisierung 57
- E**
- EBNF-Grammatik 1008
  - Editieren 63
  - Eingabe 249
  - Eingabefeld 602
  - Einrückung 70
  - Einwegfunktion 676
  - elif 128
  - Eliza 371
  - Ellipse 702
  - E-Mail 651
  - E-Mail-Client 651
  - end() 382
  - Endlosschleife 131
  - Endrekursion 177
  - endswith() 357
  - Entity-Relationship-Diagramm 664
  - Entry 449
    - delete() 449
    - get() 449
    - Passworteingabe 449
    - show 449
  - Entwicklungsumgebung 550
  - environ 415
  - ER-Diagramm siehe Entity-Relationship-Diagramm
  - Erweiterte Zuweisung 57
  - Escape-Sequenz 99
  - EVA-Prinzip 71
  - Event 803
    - binden 528
    - Taste 524
  - Eventhandler 526
  - Event-Modifizierer 524
  - Event-Sequenz 522
  - exc\_info() 404
  - except 140
  - Exception siehe Ausnahme
  - exec\_prefix 404
  - executable 404
  - execute() 667
  - exists() 412, 413
  - exit() 404
  - exitfunc 404
  - expand 464
  - Exponentialschreibweise 90
  - Externe Datei 250
  - Externes Modell 664
  - Extreme Programming 691

**F**

- Fallunterscheidung 128
- False 86, 87
- Farbe 440
- Farbmischung
  - additive 454
- Farbverlauf 808
- Farbwechselleuchte 803
- Fehler 76, 242, 305, 571
  - logischer 77, 571
  - Syntaxfehler 77, 571
- Fenster
  - mehrere Fenster 550
- fg 438
- FieldStorage 604
- FIFO 768
- File 249
  - laden 253
  - lesen und schreiben 256
  - Modus 251
  - speichern 252, 258
- File Transfer Protocol 636
- file() 251
- fill 464
- finally 260
- find\_all() 492
- find\_closest() 492
- find\_overlapping() 492
- find() 359
- findall() 378, 379
- Finden
  - gieriges 380
  - nicht gieriges 380
- Fingerabdruck 676
- Fingerprint 676
- firstChild 738
- Flash 745
- Fleisch-Analyse 393
- FLOAT 668
- float 86
- float() 109
- flush() 256
- Folge
  - rekursive 137
- Font 439, 804
- font 438
- for 133
- foreground 438
- Formaler Parameter 154
- Formatierung
  - Tabelle 266
- Formatierungsoperator % 363
- Form-Layout 815
- Foto 502
- Frame 457
- Fremdschlüssel 666
- from 52
- from\_ 455
- frozenset 106
- FTP 636
- ftplib 636
- FTP-Server 637, 639
- Funktion 48, 151
  - als Objekt 184
  - Aufruf 151
  - Ausführung 160
  - beliebige Anzahl von Parametern 170
  - Definition 154
  - Kopf 154
  - Körper 154
  - Lambda-Form 185
  - lokale Funktion 171
  - Parameter 151
  - Parameterübergabe 164
  - rekursive 172
  - Schlüsselwort-Argument 168
  - Seiteneffekt 163
  - voreingestellter Parameterwert 166
- Funktionskopf 154
- Funktionskörper 154
- Funktionsplotter 514

**G**

- Ganze Zahl 88
- Geheimnisprinzip 296
- Generator 221
- Generatordruck 222
- Generatorfunktion 222
- geopy 856
- Geräte-Manager 917
- Geschäftsprozess 325
- Geschäftsprozessdiagramm 325
- Geschwister 732
- Gesprächsroboter 372
- get() 449, 539
- getatime() 412, 413
- getcwd() 408
- getenv() 415
- getfirst() 606
- getlist() 606
- getmtime() 412, 413
- getPixel() 510
- getrefcount() 406
- getrefcount(objekt) 404
- getresponse() 644
- getsize() 412
- getvalue() 606
- Gieriges Finden 380
- Gleich 44
- Gleitkommazahl 90
- global 163
- Global Interpreter Lock (GIL) 934
- Globaler Name 160
- Globals 590

gmtime() 419  
 Go 591  
 Grafik 491  
 Grammatik 45, 1008  
 Graph 769  
 grid() 466  
 GUI 431, 470

## H

Hardware 26  
 hasAttributes() 739  
 hasChildNodes() 739  
 height 438, 442  
 Hexadezimalzahl 89  
 hidden 603  
 Hintergrundbild 506  
 Hotkey 42  
 HTML  
   Checkbox 603  
   Eingabefeld 602  
   Formular 600  
   Passworteingabe 602  
   Radiobutton 602  
   Submit-Button 603  
   versteckte Variablen 603  
 HTML-Formular 600  
   Checkbox 603  
   Eingabefeld 602  
   Radiobutton 602  
 HTTP 644  
 HTTPConnection 644  
 HTTP-Paket 594  
 HTTP-Server 594, 599  
 Hypertext Transfer Protocol 644

## I

Icon 505  
 id() 121  
 Identifier 45  
 Identisch 44  
 Identität 43  
 IDLE 41, 63  
 if 127  
 if-else 127  
 IGNORECASE 378  
 image 438  
 Imperativ 31  
 import 52  
 in 121, 197  
 Index 539  
 IndexError 572  
 indicatoron 452, 453  
 Informatik 25  
 Information hiding 296  
 insert() 539  
 insertBefore(newChild, 739  
 Installation 38

Instanz 34, 281  
 INT 668  
 int 86, 88  
 int() 108  
 Interaktive Webseite 599  
 Interaktiver Modus 37, 40  
 Internes Modell 663  
 Internet-Programmierung 635  
 Interpreter 28  
 IOError 572  
 isalnum() 357  
 isalpha() 357  
 isdigit() 357  
 isdir() 408  
 isfile() 408  
 islower() 357  
 isupper() 357  
 Item 492  
 itemcget () 492  
 itemconfigure() 492  
 items() 236  
 Iteration 133  
 Iterator 224  
 Iterierbar 86

## J

justify 438, 537

## K

Kalender 827  
 Kalenderdatum 422  
 Kamerabild 833  
 Kante 769  
 Kardinalität 338  
 Keller 765  
 Key siehe Schlüssel  
 KeyError 572  
 keys() 236, 606  
 Keyword siehe Schlüsselwort  
 Kind 732  
 Klasse 34, 277, 279  
   Beziehung zwischen Klassen 328  
   Definition 279  
   Dokumentation 304  
   Fehler 305  
   Konstruktor 280  
   Kopf 279  
   Oberklasse 279  
   Programmierstil 303  
   Spezialisierung 298  
 Klassenattribut 278  
 Klassenbibliothek 315  
 Klassenstruktur 324  
 Knoten 732, 769  
 Kollektion 86  
 Kommandozeilen-Argument 267  
 Kommentar 67, 75

Kommunikation 431, 635  
 asynchrone 431  
 Komplexe Zahl 91  
 Konjunktion 123  
 Konkatenation 104  
 Konstruktor 280  
 Kontrollstruktur 117  
 Endlosschleife 131  
 try 139  
 Kontrollvariable 447  
 Konzeptuelles Modell 663  
 Kopie  
 flache 209  
 tiefe 209  
 Kreisdiagramm 496  
 Kunststoff 458  
 Kurze Zeichenkette 98

## L

Label 445  
 label 455  
 Lambda-Form 185  
 Landesumweltamt 654  
 Lange Zeichenkette 99  
 lastChild 738  
 Laufzeitfehler  
 abfangen 139  
 Laufzeitsystem 403  
 Layout 75, 463  
 Layout-Fehler 465  
 Leerraum 443  
 Leichtgewichtprozess 560  
 Lichtschalter 457  
 LIFO 765  
 List comprehension 205  
 list() 111  
 listdir () 408  
 Liste 102, 204, 666  
 erzeugen 204  
 list comprehension 205  
 Modellierung 217  
 Operationen 204  
 sortieren 210  
 verändern 207  
 Literal 43, 83  
 ljust 356  
 load() 263  
 Locals 590  
 localtime() 419  
 Lock-Mechanismus 946  
 Log 581  
 Log-Datei 581  
 Logger-Objekt 588  
 logging 581  
 Logging-Level 583  
 login() 651  
 Logischer Fehler 77, 571

Lokaler Name 160  
 long 86  
 lower() 357  
 lstrip ([chars]) 358

## M

Mandelbrotmenge 501  
 Maskieren 377  
 Master-Slave-Hierarchie 436  
 match() 379  
 Match-Objekt 382  
 Matrix 866  
 Matrizenmultiplikation 878  
 MD5 676  
 Medianfilter 893  
 Mehrere Fenster 550  
 Memory 529  
 Menge 106, 121  
 Menu 542  
 Methoden 543  
 Optionen der Choices 544  
 Menü 542  
 Mergesort 584  
 Messagebox 547  
 Metasprache 729  
 Methode 33, 50, 278, 289  
 Migration 964  
 MIT-Lizenz 1030  
 mkdir() 411  
 mktime() 419  
 mod\_wsgi 611  
 mode 256  
 Modell  
 externes 664  
 internes 663  
 konzeptuelles 663  
 Modellieren 323  
 Model-View-Template 955  
 Modul 315  
 importieren 317  
 kompilieren 320  
 Programmierstil 321  
 speichern 317  
 Zugang sicherstellen 319  
 modules 404  
 Modulo 95  
 Modus  
 Debugging 577  
 interaktiver 37, 40  
 optimierter 577  
 move() 492  
 Multimedial 431  
 Multimeter  
 digitales 907  
 Multiplikation 93  
 Musical 337

**N**

Nachbedingung 572  
 Name 44  
   globaler 160  
   lokaler 160  
 NameError 572  
 Navigieren 637  
 Negation 122  
 Netiquette 640  
 next() 224  
 nextSibling 739  
 Nichtterminalsymbol 1008  
 Node 738  
 nodeType 739  
 None 87  
 NoneType 87  
 not 122  
 not in 121  
 NumPy 865

**O**

Oberklasse 279  
 Object Relational Mapping (ORM) 966  
 Objekt 43, 83, 277  
   Abstraktion 296  
   anonymes 283  
   Attribut 278  
   Botschaft 50  
   callable object 48  
   für reguläre Ausdrücke 378  
   Geheimnisprinzip 296  
   Identität 43  
   Instanz 281  
   laden 263  
   Match-Objekte 382  
   Methode 50, 289  
   Name 44  
   speichern 262  
   textuelle Repräsentation 368  
   Typ 43  
   Verkapselung 296  
   Wert 43  
   Zustand 283  
 Objektattribut 284  
 Objektdiagramm 283  
 Objektorientierte Analyse 323, 674  
 Objektorientierte Programmierung 31  
 Objektorientierte Software-Entwicklung 323  
 Objektorientierter Entwurf 323  
 Objektorientiertes Modellieren 323  
 Objektsymbol (UML) 283  
 offvalue 453  
 Online-Abstimmung 629  
 Online-Redaktionssystem 672  
 Online-Shop 654  
 onvalue 453  
 OOA siehe Objektorientierte Analyse

OOD siehe Objektorientierter Entwurf  
 OOP siehe Objektorientierte Programmierung  
 OpenStreetView 856  
 OpenWeatherMap 759  
 Operator  
   - 93  
   % 363  
   + 93  
   in 121, 197  
   logischer Operator 122  
   Priorität 97  
   überladen 290  
   Vergleichsoperatoren 118, 884  
   Vorzeichenoperator 93  
 Optimierter Modus 577  
 Option 267  
 or 124  
 orient 455, 541  
 os 407  
 Over 591  
 Ozonkonzentration 653

**P**

pack() 463  
 Packer 463  
 padx 438, 464, 467  
 pady 438, 464, 467  
 Paradigma 30  
 Parallele Datenverarbeitung 933  
 Parallele Programmierung 933  
 Parameter 151  
   aktueller 49, 151  
   formaler 154  
 Parameterliste 154  
 Parameterübergabe 164, 165  
 Parameterwert  
   voreingestellter 166  
 parentNode 739  
 parse() 737  
 parseString() 737  
 Passende Zeichenkette 374  
 Passwort 449  
 Passworтеingabefeld 602  
 path 404  
 pendown() 175  
 Performance-Analyse 711, 719  
 Pfad 253, 254  
   absolut 253  
   relativ 254  
 Pfadbezeichnung 252  
 PhotoImage 499  
   copy() 499  
   height() 499  
   put 500  
   width() 499  
   write() 500  
 pickle 262



- PIL.Image
    - crop() 510
    - resize() 511
    - size 511
  - pip 1027
  - Pixelgrafik 500
  - platform 404
  - Platonisches Schriftzeichen 359
  - Playlist 845
  - Polymorphie 290
  - Polymorphismus 290
  - pop() 766
  - Portable Pixmap 508
  - Positionsargument 169
  - Potenz 92
  - PPM 508
  - previousSibling 739
  - Primfaktor 575
  - print 51
  - Priorität 97
  - Problem 74
  - Problemspezifikation 74
  - Profiler 711
  - Programm 25
  - Programmieren
    - objektorientiertes 31
  - Programmierparadigma 30
  - Programmierstil 74, 126, 187, 303, 321
  - Programmierung
    - parallele 933
  - Programmverzweigung 126
  - Prompt 41
  - Protokoll 635
  - Prozess 559
    - unterbrechen 422
  - Pseudofile 264
  - Pulldown-Menü 542
  - push() 766
  - put() 500
  - putenv() 415
  - PyPI 1028
  - PyQT5 785
  - PySerial 916
  - Python Package Index 1028
  - Python-Homepage 37
  - Python-Interpreter 40
  - Python-Shell 41
- Q**
- QCalendarWidget 827, 832
  - QCamera 835
  - QCheckBox 795
  - QFileDialog 837
  - QIcon 809
  - QInputDialog 837
  - QLabel 793
  - QMessageBox 809
  - QPixmap 793
  - QPlaylist 848
  - QRadioButton 795
  - Qt 785
  - QTextEdit 827
  - Qt-Fenster 801
  - QTimer 810
  - Qt-Layout 812
  - Qt-Widgets 792
  - Qualifizierer 522
  - QueryString 601
  - Queue 768
  - Quicksort 215, 580
  - Quit 591
  - quit() 651
  - QMediaPlayer 847
  - QVideoWidget 852
  - QViewFinder 833
  - QWebView 817
- R**
- RadioButton 451
    - command 451
    - Erscheinungsform 451
    - Selektion 452
    - variable 452
  - Rahmen 440
  - raise 578
  - range() 134
  - Raster-Layout 466
  - Raumplan 771
  - re 382
  - read() 256, 644
  - readline() 256
  - reason 644
  - Regel 45, 1009
  - Regulärer Ausdruck 374
  - Rekursionstiefe 182
  - Rekursive Folge 137
  - Rekursive Funktion 172
  - Relation 665
  - Relationale Datenbank 665
  - relief 438
  - removeChild(oldChild) 739
  - Rendern 987
  - RE-Objekt 378
  - replace() 359
  - request() 644
  - resolution 455
  - reST 1031
  - reStructuredText 1031
  - retrbinary() 637
  - retrlines() 637
  - rjust 356
  - Rollbalken 540
  - row 467
  - rowspan 467

rstrip() 358  
 run module 64  
 run() 563

## S

Scale 455  
 Schiffe versenken 478  
 Schlange 768  
 Schleife  
   Abbruch 137  
 Schlüssel 107  
 Schlüsselwort 46  
 Schlüsselwort-Argument 168  
 Schriftzeichen  
   platonische 359  
 Scrollbar 540  
 sdist 1033  
 search() 379  
 see() 539  
 seek() 256  
 Seiteneffekt 163  
 Sekundenformat 419  
 Selbstähnlich 178  
 Selbstdokumentation 579  
 select() 452  
 SELECT-Anweisung 669  
 Semantik 28  
 sendmail() 651  
 Sequenz 97  
   gemeinsame Operationen 197  
   in 197  
   Konkatenation 104, 197  
   Länge 105, 198  
   not in 197  
   Slicing 199  
   veränderbar und unveränderbar 105  
   Vervielfältigung 104  
   Zugriff 103  
 serial 918  
 set\_debuglevel() 651  
 setup.py 1029  
 Shell 40  
 Shell-Fenster 64  
 Shortcut 42  
 show 449  
 showerror() 547  
 showinfo() 547  
 showturtle() 175  
 showvalue 456  
 showwarning() 547  
 Sicht 664  
 Sichtbarkeit 284  
 side 464  
 Sierpinski-Dreieck 178  
 Signal 794  
 Simple Mail Transfer Protocol 651  
 SimpleCookie 626

Size Policy 848  
 Skript 63  
   Ausführung beenden 407  
 sleep() 419  
 Slicing 199, 201, 872  
 slider 540  
 Slot 795  
 SMTP 651  
 Software 26  
 Sommerzeit 420  
 Sortieren 210  
 Sortierverfahren 213  
 Soziogramm 781  
 Speech SDK 385  
 speed() 175  
 Speichern  
   Files 258  
   Objekte 262  
 Spezialisierung 298  
 Spirale 175  
 split() 358, 379, 381  
 splitlines() 358  
 Sprachsynthese 385  
 SQL 667  
 SQL-Injection 671  
 sqlite3 667  
 Stack 765  
 Stand-alone-Skript 315  
 Standardausgabe 405  
 Standardeingabe 405  
 Stapel 765  
 start\_new\_thread() 560, 561  
 start() 382, 563  
 Startsymbol 1009  
 Statement siehe Anweisung  
 staticmethod() 294  
 Statische Methoden 294  
 status 644  
 stderr 404  
 stdin 404  
 stdout 404  
 Steganografie 511, 887  
 Step 590  
 Sternenhimmel 715  
 sticky 467  
 StopIteration 224  
 str() 110  
 String 98  
   kurze Zeichenkette 98  
   lange Zeichenkette 99  
   siehe auch Zeichenkette  
 strip() 358  
 Stylesheet 806  
 sub() 379, 382  
 Subklasse 297  
 Submit-Button 603  
 Suche  
   binäre 212

Suchroboter 639  
 Synchronisation 945  
 Syntax 28  
 Syntaxfehler 77  
 sys 403  
 sys.argv 268  
 sys.path 319  
 sys.stdin 264  
 sys.stdout 264  
 Systemfunktion 403  
 Systemsoftware 26  
 Systemumgebung 404

**T**

Tabelle 266  
 tabs 537  
 tag\_bind() 492  
 Taschenrechner 467  
 Tastenkombination 42  
 Tastenname 524  
 TCP/IP-Modell 635  
 tell() 256  
 Terminalsymbol 1008  
 Test
 

- Turing 371
- Vorkommenstest 372

 Test Driven Development 691  
 Testen 315, 691  
 TestPyPI 1028  
 Testreihe 703  
 Text
 

- Index-Formate 540
- Methoden 538
- Optionen 537
- Rollbalken 540

 text 438  
 Textanalyse 372  
 Texteditor 538, 544  
 textvariable 438  
 Thread 559, 562, 933  
 threading 560, 562  
 time 52  
 time() 419  
 timedelta 425  
 title() 445  
 Tk 444  
 tkFileDialog 546  
 Tkinter 431, 491  
 tkMessageBox 547  
 top() 766  
 toprettyxml() 737  
 toxml() 737  
 Trennstring 381  
 trough 540  
 True 86, 87  
 try 139  
 Tuning 711, 719

Tupel 101, 203  
 tuple() 111  
 Turing-Test 371  
 Türme von Hanoi 191  
 Turtle-Grafik 174  
 Typ 43
 

- None 87

 TypeError 572  
 Typumwandlung 107

**U**

Überladen 290  
 Umgebungsvariable 415  
 UML 283  
 UML-Klassendiagramm 330  
 underline 438  
 Unicode 359
 

- utf-8 730

 unicode() 110  
 unittest 703  
 Unix
 

- Programmausführung 66

 unlink() 739  
 Unterklasse 297  
 update() 239  
 upper() 357  
 URL 594  
 URL-Pattern 984  
 USB-to-Serial 917  
 use case 325  
 UTC 418  
 utf-8 730

**V**

ValueError 572  
 values() 236  
 VARCHAR 668  
 Variablenname 54  
 Vektor 866  
 Verarbeitungsschicht 636  
 Verbinden
 

- Zeilen 69

 Vererbung 297  
 Verfeinerung 156  
 Vergleich 117  
 Vergleichsoperator 118, 884  
 Verkapselung 296  
 Verzeichnis 251
 

- anlegen 411
- Merkmale abfragen 412
- suchen 408

 Verzeichnisbaum 416  
 Verzweigung 127  
 Videoplayer 841  
 View 959, 982  
 View-Funktion 955, 983

Vokabeltrainer 240  
 Vorbedingung 572  
 Voreingestellter Parameterwert 166  
 Vorkommenstest 372

**W**

Wahrheitswert 87  
 Währungsumrechner 457  
 walk() 416  
 Webbrowser 788  
 Webseite 599  
 Wegenetz 772  
 Wegsuche 775  
 Weltkarte 854  
 Wert 43  
 Wetterdaten 760  
 while 130  
 Widerstandsthermometer 924  
 Widget 435, 444
 

- Button 445
- einfach 435
- Farbe 440
- Font 439
- Größe 441
- konfigurieren 438
- Layout 463
- Leerraum 443
- Master-Slave-Hierarchie 436
- Methoden 444
- Option 437, 438
- Rahmen 440
- Text 537

 width 438, 442  
 Wiederholung 130  
 Windows 65  
 Wort des Jahres 694  
 Wörterbuch 107, 324  
 Wörterraten 479  
 wrap 537  
 write() 252, 256  
 WSGI 610  
 WSGI-Skript 680

**X**

XML 729
 

- Attribute 742
- Datenkommunikation 745
- Tags 730

 xml.dom.minidom 736  
 xscrollcommand 537

**Y**

yield 223  
 yscrollcommand 537

**Z**

Zahl
 

- ganze 88
- Gleitkommazahl 90
- Hexadezimalzahl 89
- komplexe 91

 Zeichenkette 98, 355
 

- formatieren 356
- kurze 98
- lange 99
- passende 374
- zerlegen 381

 Zeile
 

- Einrückung 70
- verbinden 69

 Zeilenstruktur 68  
 Zeit 418  
 Zeitkomplexität 213  
 Zeitstring 421  
 Zeittupel 420  
 Zeitzone 424  
 ZeroDivisionError 572  
 Zugriffsrecht 409  
 Zuweisung 53
 

- erweitert 57

 Zuweisungsoperator 54