HANSER

Leseprobe

zu

„Einführung in Python 3“ (3. Auflage)

von Bernd Klein

ISBN (Buch): 978-3-446-45208-4
ISBN (E-Book): 978-3-446-45387-6

Weitere Informationen und Bestellungen unter
http://www.hanser-fachbuch.de/9783446452084
sowie im Buchhandel

Inhalt

Vorwort ..... **XVII**

Danksagung ..... **XVIII**

1 Einleitung **1**
1.1 Einfach und schnell zu lernen 1
1.2 Zielgruppe des Buches 1
1.3 Aufbau des Buches 2
1.4 Programmieren lernen „interaktiv" 3
1.5 Download der Beispiele und Hilfe 4
1.6 Anregungen und Kritik 4

Teil I Grundlagen **5**

2 Kommandos und Programme **7**
2.1 Erste Schritte mit Python 7
2.1.1 Linux 7
2.1.2 Windows 8
2.1.3 macOS 9
2.2 Herkunft und Bedeutung des Begriffes interaktive Shell 9
2.2.1 Erste Schritte in der interaktiven Shell 10
2.3 Verlassen der Python-Shell 11
2.4 Benutzung von Variablen 12
2.5 Mehrzeilige Anweisungen in der interaktiven Shell 12
2.6 Programme schreiben oder schnell mal der Welt "Hallo" sagen 13

3 Bytecode und Maschinencode **17**
3.1 Einführung 17
3.2 Unterschied zwischen Programmier- und Skriptsprachen 17
3.3 Interpreter- oder Compilersprache 17

4 Datentypen und Variablen ... 21
4.1 Einführung ... 21
4.2 Variablennamen ... 23
4.2.1 Gültige Variablennamen ... 23
4.2.2 Konventionen für Variablennamen ... 24
4.3 Datentypen ... 24
4.3.1 Ganze Zahlen ... 24
4.3.2 Fließkommazahlen ... 26
4.3.3 Zeichenketten ... 26
4.3.4 Boolesche Werte ... 26
4.3.5 Komplexe Zahlen ... 27
4.3.6 Operatoren ... 27
4.4 Statische und dynamische Typdeklaration ... 28
4.5 Typumwandlung ... 30
4.6 Datentyp ermitteln ... 31

5 Sequentielle Datentypen ... 33
5.1 Übersicht ... 33
5.1.1 Zeichenketten oder Strings ... 34
5.1.2 Listen ... 36
5.1.3 Tupel ... 36
5.2 Indizierung von sequentiellen Datentypen ... 37
5.3 Teilbereichsoperator ... 38
5.4 Die len-Funktion ... 40
5.5 Aufgaben ... 41

6 Dictionaries ... 43
6.1 Dictionaries und assoziative Felder ... 43
6.2 Definition und Benutzung ... 44
6.3 Fehlerfreie Zugriffe auf Dictionaries ... 46
6.4 Zulässige Typen für Schlüssel und Werte ... 47
6.5 Verschachtelte Dictionaries ... 48
6.6 Methoden auf Dictionaries ... 48
6.7 Operatoren ... 52
6.8 Die zip-Funktion ... 52
6.9 Dictionaries aus Listen erzeugen ... 54
6.10 Aufgaben ... 55

7 Mengen ... 57
7.1 Übersicht ... 57
7.2 Mengen in Python ... 57
7.2.1 Sets erzeugen ... 58
7.2.2 Mengen von unveränderlichen Elementen ... 58
7.3 Frozensets ... 59
7.4 Operationen auf „set"-Objekten ... 59
7.4.1 add(element) ... 59
7.4.2 clear() ... 60
7.4.3 copy ... 60
7.4.4 difference() ... 60
7.4.5 difference_update() ... 61
7.4.6 discard(el) ... 61
7.4.7 remove(el) ... 62
7.4.8 intersection(s) ... 62
7.4.9 isdisjoint() ... 62
7.4.10 issubset() ... 63
7.4.11 issuperset() ... 63
7.4.12 pop() ... 64

8 Eingaben ... 65
8.1 Eingabe mittels input ... 65

9 Verzweigungen ... 67
9.1 Anweisungsblöcke und Einrückungen ... 67
9.2 Bedingte Anweisungen in Python ... 70
9.2.1 Einfachste if-Anweisung ... 70
9.2.2 if-Anweisung mit else-Zweig ... 71
9.2.3 elif-Zweige ... 71
9.3 Vergleichsoperatoren ... 72
9.4 Zusammengesetzte Bedingungen ... 72
9.5 Wahr oder falsch: Bedingungen in Verzweigungen ... 72
9.6 Aufgaben ... 73

10 Schleifen ... 75
10.1 Übersicht ... 75
10.2 while-Schleife ... 76
10.3 break und continue ... 77
10.4 die Alternative im Erfolgsfall: else ... 78
10.5 For-Schleife ... 80
10.6 Aufgaben ... 83

11 Dateien lesen und schreiben ... 87
11.1 Dateien ... 87
11.2 Text aus einer Datei lesen ... 87
11.3 Schreiben in eine Datei ... 89
11.4 In einem Rutsch lesen: readlines und read ... 89
11.5 with-Anweisung ... 90
11.6 Aufgaben ... 91

12 Formatierte Ausgabe und Strings formatieren ... 93
12.1 Wege, die Ausgabe zu formatieren ... 93
12.2 print-Funktion ... 93
12.3 Stringformatierung im C-Stil ... 95
12.4 Der pythonische Weg: Die String-Methode „format" ... 99
12.5 Benutzung von Dictionaries beim Aufruf der „format"-Methode ... 102
12.6 Benutzung von lokalen Variablen in „format" ... 103
12.7 Formatierte Stringliterale ... 104
12.8 Weitere String-Methoden zum Formatieren ... 104

13 Flaches und tiefes Kopieren ... 107
13.1 Einführung ... 107
13.2 Kopieren einer Liste ... 108
13.3 Flache Kopien ... 110
13.4 Kopieren mit deepcopy ... 111
13.5 Deepcopy für Dictionaries ... 112

14 Funktionen ... 113
14.1 Allgemein ... 113
14.2 Funktionen ... 113
14.3 Docstring ... 115
14.4 Standardwerte für Funktionen ... 117
14.5 Schlüsselwortparameter ... 118
14.6 Funktionen ohne oder mit leerer return-Anweisung ... 118
14.7 Mehrere Rückgabewerte ... 119
14.8 Lokale und globale Variablen in Funktionen ... 120
14.9 Parameterübergabe im Detail ... 122
14.10 Effekte bei veränderlichen Objekten ... 124
14.11 Kommandozeilenparameter ... 125
14.12 Variable Anzahl von Parametern / Variadische Funktionen ... 126
14.13 * in Funktionsaufrufen ... 128

14.14 Beliebige Schlüsselwortparameter 129
14.15 Doppeltes Sternchen im Funktionsaufruf 129
14.16 Aufgaben 130

15 Rekursive Funktionen 133
15.1 Definition und Herkunft des Begriffs 133
15.2 Definition der Rekursion 134
15.3 Rekursive Funktionen in Python 134
15.4 Die Tücken der Rekursion 135
15.5 Fibonacci-Folge in Python 136
15.6 Aufgaben 140

16 Listen und Tupel im Detail 143
16.1 Stapelspeicher 143
16.2 Stapelverarbeitung in Python: pop und append 144
16.3 extend 144
16.4 ‚+'-Operator oder append 145
16.5 Entfernen eines Wertes 147
16.6 Prüfen, ob ein Element in Liste enthalten ist 147
16.7 Finden der Position eines Elementes 147
16.8 Einfügen von Elementen 148
16.9 Besonderheiten bei Tupel 148
16.9.1 Leere Tupel 149
16.9.2 1-Tupel 149
16.9.3 Mehrfachzuweisungen, Packing und Unpacking 149
16.10 Die veränderliche Unveränderliche 151
16.11 Sortieren von Listen 151
16.11.1 „sort" und „sorted" 151
16.11.2 Umkehrung der Sortierreihenfolge 152
16.11.3 Eigene Sortierfunktionen 152
16.12 Aufgaben 155

17 Modularisierung 157
17.1 Module 157
17.1.1 Namensräume von Modulen 158
17.1.2 Namensräume umbenennen 159
17.1.3 Modularten 159
17.1.4 Suchpfad für Module 160
17.1.5 Inhalt eines Modules 161

17.1.6 Eigene Module ... 161
17.1.7 Dokumentation für eigene Module ... 162
17.2 Pakete ... 163
17.2.1 Einfaches Paket erzeugen ... 164
17.2.2 Komplexeres Paket ... 165
17.2.3 Komplettes Paket importieren ... 168

18 Globale und lokale Variablen ... 171
18.1 Einführung ... 171
18.2 Globale und lokale Variablen in Funktionen ... 172

19 Alles über Strings 175
19.1 ... fast alles ... 175
19.2 Aufspalten von Zeichenketten ... 176
19.2.1 split ... 176
19.2.2 Standardverhalten und „maxsplit" ... 178
19.2.3 rsplit ... 179
19.2.4 Folge von Trennzeichen ... 181
19.2.5 splitlines ... 182
19.2.6 partition ... 183
19.3 Zusammenfügen von Stringlisten mit join ... 183
19.4 Suchen von Teilstrings ... 183
19.4.1 „in" oder „not in" ... 183
19.4.2 s.find(substring[, start[, end]]) ... 184
19.4.3 s.rfind(substring[, start[, end]]) ... 184
19.4.4 s.index(substring[, start[, end]]) ... 185
19.4.5 s.rindex(substring[, start[, end]]) ... 185
19.4.6 s.count(substring[, start[, end]]) ... 185
19.5 Suchen und Ersetzen ... 186
19.6 Nur noch Kleinbuchstaben oder Großbuchstaben ... 186
19.7 capitalize und title ... 186
19.8 Stripping Strings ... 187
19.9 Strings ausrichten ... 187
19.10 String-Tests ... 188
19.11 Aufgaben ... 190

20 Ausnahmebehandlung ... 193
20.1 Abfangen mehrerer Exceptions ... 195
20.2 except mit mehrfachen Ausnahmen ... 196
20.3 Die optionale else-Klausel ... 196
20.4 Fehlerinformationen über sys.exc_info ... 197
20.5 Exceptions generieren ... 197
20.6 Finalisierungsaktion ... 198

Teil II Objektorientierte Programmierung ... 199

21 Grundlegende Aspekte ... 201
21.1 Bibliotheksvergleich ... 201
21.2 Objekte und Instanzen einer Klasse ... 203
21.3 Kapselung von Daten und Methoden ... 204
21.4 Eine minimale Klasse in Python ... 204
21.5 Eigenschaften und Attribute ... 205
21.6 Methoden ... 208
21.7 Instanzvariablen ... 209
21.8 Die __init__-Methode ... 209
21.9 Destruktor ... 211
21.10 Datenkapselung, Datenabstraktion und Geheimnisprinzip ... 212
21.10.1 Definitionen ... 212
21.10.2 Zugriffsmethoden ... 214
21.10.3 Properties ... 215
21.10.4 Public-, Protected- und Private-Attribute ... 217
21.10.5 Weitere Möglichkeiten der Properties ... 218
21.10.6 Properties mit Dekorateuren ... 221
21.11 Die __str__- und die __repr__-Methode ... 222
21.12 Klassenattribute ... 227
21.13 Statische Methoden ... 229
21.13.1 Einleitendes Beispiel ... 230
21.13.2 Getter und Setter für private Klassenattribute ... 231
21.14 Public-Attribute statt private Attribute ... 232
21.15 Magische Methoden und Operatorüberladung ... 233
21.15.1 Einführung ... 233
21.15.2 Übersicht magische Methoden ... 235
21.15.3 Beispielklasse: Length ... 236

22 Bruchklasse ... 241
22.1 Brüche à la 1001 Nacht ... 241
22.2 Zurück in die Gegenwart ... 242
22.3 Rechenregeln ... 244
22.3.1 Multiplikation von Brüchen ... 244
22.3.2 Division von Brüchen ... 245
22.3.3 Addition von Brüchen ... 246
22.3.4 Subtraktion von Brüchen ... 246
22.3.5 Vergleichsoperatoren ... 247
22.4 Integer plus Bruch ... 247
22.4.1 Die Bruchklasse im Überblick ... 248
22.5 Fraction-Klasse ... 250

23 Vererbung ... 251
23.1 Oberbegriffe und Oberklassen ... 251
23.2 Ein einfaches Beispiel ... 252
23.3 Überladen, Überschreiben und Polymorphie ... 253
23.4 Vererbung in Python ... 256
23.5 Klassenmethoden ... 259
23.6 Standardklassen als Basisklassen ... 261

24 Mehrfachvererbung ... 263
24.1 Einführung ... 263
24.2 Beispiel: CalendarClock ... 264
24.3 Diamand-Problem oder „deadly diamond of death" ... 272
24.4 super und MRO ... 273

25 Slots ... 279
25.1 Erzeugung von dynamischen Attributen verhindern ... 279

26 Dynamische Erzeugung von Klassen ... 281
26.1 Beziehung zwischen „class" und „type" ... 281

27 Metaklassen ... 285
27.1 Motivation ... 285
27.2 Definition ... 290
27.3 Definition von Metaklassen in Python ... 290
27.4 Singletons mit Metaklassen erstellen ... 292

27.5 Beispiel – Methodenaufrufe zählen ... 293
27.5.1 Einführung ... 293
27.5.2 Vorbereitungen ... 293
27.5.3 Ein Dekorateur, um Funktionsaufrufe zu zählen ... 295
27.5.4 Die Metaklasse „Aufrufzähler" ... 295

28 Abstrakte Klassen ... 297

29 Aufgaben ... 301

Teil III Fortgeschrittenes Python ... 305

30 lambda, map, filter und reduce ... 307
30.1 lambda ... 307
30.2 map ... 310
30.3 Filtern von sequentiellen Datentypen mittels „filter" ... 312
30.4 reduce ... 313
30.5 Aufgaben ... 314

31 Listen-Abstraktion/List Comprehension ... 315
31.1 Die Alternative zu Lambda und Co. ... 315
31.2 Syntax ... 316
31.3 Weitere Beispiele ... 316
31.4 Die zugrunde liegende Idee ... 317
31.5 Anspruchsvolleres Beispiel ... 318
31.6 Mengen-Abstraktion ... 318
31.7 Rekursive Primzahlberechnung ... 319
31.8 Generatoren-Abstraktion ... 319
31.9 Aufgaben ... 320

32 Generatoren und Iteratoren ... 323
32.1 Einführung ... 323
32.2 Iteration in for-Schleifen ... 323
32.3 Generatoren ... 325
32.4 Generatoren zähmen mit firstn und islice ... 327
32.5 Beispiele ... 328
32.5.1 Permutationen ... 328
32.5.2 Variationen und Kombinationen ... 329
32.6 Generator-Ausdrücke ... 331

32.7 return-Anweisungen in Generatoren ... 332
32.8 send-Methode / Koroutinen ... 332
32.9 Die throw-Methode ... 334
32.10 Dekoration von Generatoren ... 336
32.11 yield from ... 337
32.12 Aufgaben ... 339

33 Dekorateure ... 341
33.1 Einführung Dekorateure ... 341
33.1.1 Verschachtelte Funktionen ... 342
33.1.2 Funktionen als Parameter ... 343
33.1.3 Funktionen als Rückgabewert ... 344
33.1.4 Fabrikfunktionen ... 345
33.2 Ein einfacher Dekorateur ... 346
33.3 @-Syntax für Dekorateure ... 347
33.4 Anwendungsfälle für Dekorateure ... 350
33.4.1 Überprüfung von Argumenten durch Dekorateure ... 350
33.4.2 Funktionsaufrufe mit einem Dekorateur zählen ... 351
33.5 Dekorateure mit Parametern ... 352
33.6 Benutzung von Wraps aus functools ... 354
33.7 Klassen statt Funktionen ... 355
33.7.1 Die __call__-Methode ... 355
33.8 Eine Klasse als Dekorateur benutzen ... 357
33.9 Memoisation ... 358
33.9.1 Bedeutung und Herkunft des Begriffs ... 358
33.9.2 Memoisation mit Dekorateur-Funktionen ... 358
33.9.3 Memoisation mit einer Klasse ... 359
33.9.4 Memoisation mit functools.lru_cache ... 360

Teil IV Weiterführende Themen ... 363

34 Tests und Fehler ... 365
34.1 Einführung ... 365
34.2 Modultests ... 367
34.3 Modultests unter Benutzung von __name__ ... 368
34.4 doctest-Modul ... 370
34.5 Testgetriebene Entwicklung oder „Im Anfang war der Test" ... 373
34.6 unittest ... 375
34.7 Methoden der Klasse TestCase ... 377
34.8 Aufgaben ... 380

35 Daten konservieren ... 381
35.1 Persistente Speicherung ... 381
35.2 Pickle-Modul ... 382
35.2.1 Daten „einpökeln" mit pickle.dump ... 382
35.2.2 pickle.load ... 383
35.3 Ein persistentes Dictionary mit shelve ... 383

36 Reguläre Ausdrücke ... 387
36.1 Ursprünge und Verbreitung ... 387
36.2 Stringvergleiche ... 387
36.3 Überlappungen und Teilstrings ... 389
36.4 Das re-Modul ... 389
36.5 Matching-Problem ... 390
36.6 Syntax der regulären Ausdrücke ... 392
36.6.1 Beliebiges Zeichen ... 392
36.7 Zeichenauswahl ... 393
36.8 Endliche Automaten ... 394
36.9 Anfang und Ende eines Strings ... 394
36.10 Vordefinierte Zeichenklassen ... 397
36.11 Optionale Teile ... 398
36.12 Quantoren ... 399
36.13 Gruppierungen und Rückwärtsreferenzen ... 401
36.13.1 Match-Objekte ... 401
36.14 Umfangreiche Übung ... 404
36.15 Alles finden mit findall ... 406
36.16 Alternativen ... 407
36.17 Compilierung von regulären Ausdrücken ... 407
36.18 Aufspalten eines Strings mit oder ohne regulären Ausdruck ... 409
36.18.1 split-Methode der String-Klasse ... 409
36.18.2 split-Methode des re-Moduls ... 410
36.18.3 Wörter filtern ... 412
36.19 Suchen und Ersetzen mit sub ... 413
36.20 Aufgaben ... 414

37 Typanmerkungen ... 417
37.1 Einführung ... 417
37.2 Einfaches Beispiel ... 418
37.3 Variablenanmerkungen ... 419
37.4 Listenbeispiel ... 420
37.5 typing-Modul ... 421

38 Systemprogrammierung ... 425
38.1 Systemprogrammierung ... 425
38.2 Häufig falsch verstanden: Shell ... 425
38.3 os-Modul ... 426
38.3.1 Vorbemerkungen ... 427
38.3.2 Umgebungsvariablen ... 427
38.3.3 Dateiverarbeitung auf niedrigerer Ebene ... 429
38.3.4 Die exec-„Familie" ... 434
38.3.5 Weitere Funktionen im Überblick ... 441
38.3.6 os.path - Arbeiten mit Pfaden ... 455
38.4 shutil-Modul ... 463

39 Forks ... 469
39.1 Fork ... 469
39.2 Fork in Python ... 469

Teil V Lösungen zu den Aufgaben ... 473

40 Lösungen zu den Aufgaben ... 475
40.1 Lösungen zu Kapitel 5 (Sequentielle Datentypen) ... 475
40.2 Lösungen zu Kapitel 6 (Dictionaries) ... 478
40.3 Lösungen zu Kapitel 9 (Verzweigungen) ... 480
40.4 Lösungen zu Kapitel 10 (Schleifen) ... 482
40.5 Lösungen zu Kapitel 11 (Dateien lesen und schreiben) ... 485
40.6 Lösungen zu Kapitel 16 (Listen und Tupel im Detail) ... 487
40.7 Lösungen zu Kapitel 14 (Funktionen) ... 490
40.8 Lösungen zu Kapitel 15 (Rekursive Funktionen) ... 495
40.9 Lösungen zu Kapitel 19 (Alles über Strings ...) ... 499
40.10 Lösungen zu Kapitel 21 (Grundlegende Aspekte) ... 502
40.11 Lösungen zu Kapitel 34 (Tests und Fehler) ... 512
40.12 Lösungen zu Kapitel 36 (Reguläre Ausdrücke) ... 512
40.13 Lösungen zu Kapitel 30 (lambda, map, filter und reduce) ... 518
40.14 Lösungen zu Kapitel 31 (Listen-Abstraktion/List Comprehension) ... 519
40.15 Lösungen zu Kapitel 32 (Generatoren und Iteratoren) ... 520

Stichwortverzeichnis ... 525

Vorwort

Ist es wirklich so, dass Vorwörter – ähnlich wie Bedienungsanleitungen – meistens nicht gelesen werden? Auch wenn dies sicherlich für viele zutreffen mag, so gibt es Situationen, in denen gerade ein Vorwort wertvolle Dienste leisten kann. Zum Beispiel, um die Kaufentscheidung für ein Buch zu erleichtern. So stehen auch Sie jetzt vielleicht am Regal einer guten Buchhandlung und werden möglicherweise von zwei Fragen bewegt: Sie brauchen noch eine Bestätigung, dass Python die richtige Programmiersprache für Sie ist, und möchten wissen, wie Ihnen dieses Buch helfen wird, die Sprache schnell und effizient zu erlernen.

Für Python spricht der traumhafte Anstieg der Bedeutung in Wissenschaft, Forschung und Wirtschaft in den letzten Jahren. Im renommierten PYPL-Index stand Python im Juli 2017 auf dem zweiten Platz hinter Java. Während Java im Vergleich zum Vorjahr jedoch -1,1 % Anteil verloren hatte, gewann Python +4,0 % hinzu. Weitere wichtige Gründe, Python zu lernen, sind: Python ist mit seiner einfachen natürlichen Syntax sehr einfach zu lernen. Python läuft auf allen Plattformen und erfreut sich auch beim Raspberry Pi größter Beliebtheit. Außerdem ist Python eine universell einsetzbare Programmiersprache, die sich bei den Aufgaben der Systemadministration ebenso effektiv einsetzen lässt wie im Maschinenbau, der Linguistik, der Pharmaindustrie, in der Banken- und Finanzwelt, der Physik und vielen anderen Bereichen. Weil Firmen wie Google Python lieben und unterstützen, wird Python auch kontinuierlich weiterentwickelt.

Dieses Buch erscheint nun in der dritten, weitgehend überarbeiteten Ausgabe. Es eignet sich ebenso für Programmieranfänger als auch für Umsteiger von anderen Programmiersprachen. Behandelt werden nicht nur alle grundlegenden Sprachelemente von Python, sondern auch weiterführende Themen wie Generatoren, Dekorateure, Systemprogrammierung, Threads, Forks, Ausnahmebehandlungen und Modultests. Auf über Hundert Seiten wird anschaulich und mit zahlreichen Beispiele auf die vielfältigen Aspekte der Objektorientierung eingegangen.

Brigitte Bauer-Schiewek, Lektorin

Danksagung

Zum Schreiben eines Buches benötigt es neben der nötigen Erfahrung und Kompetenz im Fachgebiet vor allem viel Zeit. Zeit außerhalb des üblichen Rahmens. Zeit, die vor allem die Familie mitzutragen hat. Deshalb gilt mein besonderer Dank meiner Frau Karola, die mich während dieser Zeit tatkräftig unterstützt hat.

Außerdem danke ich den zahlreichen Teilnehmerinnen und Teilnehmern an meinen Python-Kursen, die mir geholfen haben, meine didaktischen und fachlichen Kenntnisse kontinuierlich zu verbessern. Ebenso möchte ich den Besucherinnen und Besuchern meiner Online-Tutorials unter www.python-kurs.eu und www.python-course.eu danken, vor allem jenen, die sich mit konstruktiven Anmerkungen bei mir gemeldet haben. Wertvolle Anregungen erhielt ich auch von denen, die das Buch vorab Korrektur gelesen hatten: Stefan Günther für die Erstauflage des Buches. Für die Hilfe zur zweiten – weitestgehend überarbeiteten – Auflage möchte ich im besonderen Maße Herrn Jan Lendertse und Herrn Vincent Bermel Dank sagen. Für die dritte – wiederum stark veränderte – Auflage danke ich Herrn Wilhelm Wall für seine wertvolle Hilfe, insbesondere Tests unter macOS.

Zuletzt danke ich auch ganz herzlich dem Hanser Verlag, der dieses Buch – nun auch in der dritten Auflage – ermöglicht hat. Vor allem danke ich Frau Brigitte Bauer-Schiewek, Programmplanung Computerbuch, für die kontinuierliche ausgezeichnete Unterstützung. Für die technische Unterstützung bei LaTeX-Problemen danke ich Herrn Stephan Korell und Frau Irene Weilhart. Herrn Jürgen Dubau danke ich fürs Lektorat.

Bernd Klein, Singen

10 Schleifen

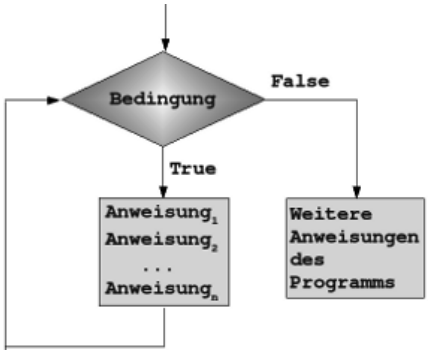
10.1 Übersicht

Schleifen werden benötigt, um einen Codeblock, also eine oder mehrere Python-Anweisungen, wiederholt auszuführen. Einen solchen Codeblock bezeichnet man auch als Schleifenkörper oder Body. In Python gibt es zwei Schleifentypen: die while-Schleife und die for-Schleife.

Bild 10.1 Karussell

Die meisten Schleifenarten, die in Programmiersprachen Verwendung finden, enthalten einen Zähler oder ganz allgemein Variablen, die im Verlauf der Berechnungen innerhalb des Schleifenkörpers ihre Werte ändern. Außerhalb, das heißt noch vor Beginn der Schleife, werden diese Variablen initialisiert. Vor jedem Schleifendurchlauf wird geprüft, ob ein Ausdruck, in dem diese Variable oder Variablen vorkommen, wahr ist. Dieser Ausdruck bestimmt das Endekriterium der Schleife. Solange die Berechnung dieses Ausdrucks wahr ist, d.h. „True" liefert, wird der Rumpf der Schleife ausgeführt. Nachdem alle Anweisungen des Schleifenkörpers durchgeführt worden sind, springt die Programmsteuerung automatisch zum Anfang der Schleife, also zur Prüfung des Endekriteriums zurück und prüft wieder, ob diese nochmals erfüllt ist. Wenn ja, geht es wie oben beschrieben weiter, ansonsten wird der Schleifenkörper nicht mehr ausgeführt, und es wird mit dem Rest des Skripts fortgefahren. Das unten stehende Diagramm zeigt dies schematisch.

10.2 while-Schleife



Das folgende Skript, das wir in der interaktiven Shell direkt eintippen können, gibt die Zahlen von 1 bis 4, gefolgt von ihrem Quadrat, unter Benutzung einer while-Schleife aus:

```
>>> i = 1
>>> while i <= 4:
...     print(i, i**2)
...     i += 1
...
1 1
2 4
3 9
4 16
```

Auch die Summe der Zahlen von 1 bis 100 lässt sich mittels einer while-Schleife leicht berechnen, wie wir im folgenden Programm sehen können:

```
n = 100
sum = 0
i = 1
while i <= n:
    sum = sum + i
    i = i + 1
print("Summe von 1 bis " + str(n) + ": " + str(sum) )
```

Zu obigem Programm lässt sich Folgendes sagen. Zur Aufnahme der Summe haben wir eine Variable `sum` benutzt. `sum` ist aber auch eine Python-Funktion, die die Summe einer aus numerischen Werten bestehenden Liste oder eines Tupels berechnet:

```
>>> sum([3, 4.5, 9])
16.5
```

Damit hätten wir uns natürlich auch das obige Programm zur Berechnung der Summe der Zahlen von 1 bis 100 sparen können, indem wir „sum" und „range" benutzen:

```
>>> n = 100
>>> sum(range(1, n+1))
5050
```

Einige kennen sicherlich auch die Gaußsche Summenformel, mit der wir die Aufgabenstellung auch ohne sum und range direkt lösen können:

```
>>> n = 100
>>> summe = n * (n + 1) / 2
>>> summe
5050.0
```

10.3 break und continue

Für die Schleifen existieren zwei wichtige Anweisungen: „break" zum vorzeitigen Abbruch der Schleife und „continue", um einen Durchlauf zu beenden und mit dem nächsten Durchlauf bzw. der Überprüfung der Abbruchbedingung weiterzumachen. Die break-Anweisung steht innerhalb des Schleifenrumpfs meist in Verbindung mit einer if-Abfrage. Stößt der Programmablauf auf eine break-Anweisung, wird die Schleife unmittelbar abgebrochen. Das Programm wird mit der ersten Anweisung nach der Schleife fortgesetzt. Bei geschachtelten Schleifen wird mittels break nur die innerste Schleife abgebrochen. Trifft der Programmablauf auf eine continue-Anweisung, wird der Schleifendurchlauf abgebrochen, und der Programmablauf kehrt zum Schleifenkopf zurück, wo geprüft wird, ob die Bedingung für einen weiteren Durchlauf erfüllt ist.

Im folgenden Beispiel benutzen wir sowohl break als auch continue. Falls ein doppelter Eintrag in der Liste steht, wird mittels continue auf das nächste Listenelement weitergegangen. Falls die Liste eine negative Zahl oder eine Null enthält, wird die Schleife komplett abgebrochen:

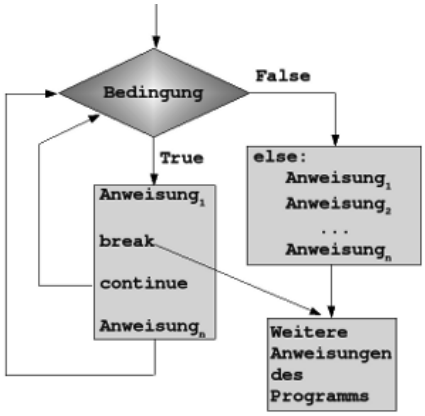
```
liste = eval(input("Liste mit positiven Zahlen eingeben: "))
n = len(liste)
i = 0
previous = None
erg = []
while i < n:
    current = liste[i]
    i += 1
    if current == previous:
        continue
    if current <= 0:
        print("Abbruch: Nicht-positive Zahl gefunden!")
        break
    erg.append(current)
    previous = current

print(erg)
```

Zum besseren Verständnis des obigen Programms zeigen wir einen Beispiellauf:

```
$ python3 break_example.py
Liste mit positiven Zahlen eingeben: [37, 99, 123, 17, 17, 17, 89, 32,
     -3]
Abbruch: Nicht-positive Zahl gefunden!
[37, 99, 123, 17, 89, 32]
```

10.4 else-Teil



C-Programmierern[1] wie auch Programmierern anderer Programmiersprachen kommt es meist sehr merkwürdig vor, wenn sie ein else ohne zugehöriges if finden. Bei Schleifen kann ein else-Teil folgen, das muss aber nicht so sein. Die Anweisungen im else-Teil werden ausgeführt, sobald die Bedingung nicht mehr erfüllt ist. Sicherlich fragen sich einige nun, worin dann der Unterschied zu einer normalen while-Schleife liegt. Hätte man die Anweisungen nicht in den else-Teil gesteckt, sondern einfach hinter die while-Schleife gestellt, wären sie ja auch genauso ausgeführt worden. Der else-Teil einer while-Schleife wird erst zusammen mit dem break-Kommando sinnvoll. Normalerweise wird eine Schleife nur beendet, wenn die Bedingung im Schleifenkopf erfüllt ist. Mit break kann man aber eine Schleife vorzeitig – also gewissermaßen als Notausstieg – verlassen.

Im folgenden Beispiel, einem einfachen Zahlenratespiel, kann man erkennen, dass in Kombination mit einem break der else-Zweig durchaus sinnvoll sein kann. Nur wenn die while-Schleife regulär beendet wird, d.h. der Spieler die Zahl erraten hat, gibt es einen Glückwunsch. Gibt der Spieler auf, d.h. break, dann wird der else-Zweig der while-Schleife nicht ausgeführt.

```
import random
n = 20
to_be_guessed = random.randint(1, n)
```

[1] Es gibt auch eine else-Schleife in C und C++. Diese wird aber nur selten verwendet.

```
guess = 0
while guess != to_be_guessed:
    guess = int(input("Neuer Versuch: "))
    if guess > 0:
        if guess > to_be_guessed:
            print("Zu gross")
        elif guess < to_be_guessed:
            print("Zu klein")
    else:
        print("Schade, dass du aufgibst!")
        break
else:
    print("Gratuliere, das war's")
```

Die Ausgabe einer Spielsitzung könnte beispielsweise so aussehen:

```
$ python3 number_game.py
Dein/Ihr Versuch: 10
Zu klein
Dein/Ihr Versuch: 15
Zu klein
Dein/Ihr Versuch: 18
Gratuliere, das war's
$
```

Bei der while-Schleife von Python handelt es sich um eine sogenannte kopfgesteuerte Schleife. Bevor der Schleifenrumpf ausgeführt wird, muss geprüft werden, ob die Bedingung im Schleifenkopf, also nach dem Schlüsselwort „while", erfüllt ist. Bei einer kopfgesteuerten Schleife erfolgt also immer zuerst die Abfrage der im Schleifenkopf stehenden Bedingung, bevor der Schleifenrumpf ausgeführt wird. Manchmal würde man lieber zuerst den Schleifenrumpf ausführen und dann eine Abbruchsbedingung prüfen. Dies ist ein Schleifenkonstrukt, welches viele Programmiersprachen bieten und unter dem Namen „fußgesteuerte Schleife" bekannt ist, z.B. in der syntaktischen Form „do SCHLEIFENKÖRPER until ABBRUCHBEDINGUNG" oder „repeat SCHLEIFENKÖRPER until ABBRUCHBEDINGUNG". Python bietet keine fußgesteuerte Schleife. Man kann diese jedoch mit einem „while True" in Verbindung mit einer „break"-Anweisung am Ende des Schleifenkörpers simulieren. Wir demonstrieren dies im folgenden Beispiel:

```
import random
n = 20
to_be_guessed = random.randint(1,n)

guess = 0
while True:
    guess = int(input("Ihr Versuch: "))
    if guess > 0:
        if guess > to_be_guessed:
            print("Zu gross")
```

```
        elif guess < to_be_guessed:
            print("Zu klein")
    # repeat until:
    if guess == to_be_guessed:
        print("Glückwunsch, das war's")
        break
```

10.5 For-Schleife

Wie auch die while-Schleife ist die for-Schleife eine Kontrollstruktur, mit der eine Gruppe von Anweisungen (ein Block) wiederholt ausführt werden kann. Die Syntax der for-Schleifen unterscheidet sich in den verschiedenen Programmiersprachen. Ebenso ist die Semantik einer for-Schleife, also wie sie vom Compiler oder Interpreter zu verstehen bzw. auszuführen ist, von Programmiersprache zu Programmiersprache unterschiedlich. Die „klassische" numerische Schleife, wie sie C und C++ kennt, besitzt eine Schleifenvariable, die mit einem Startwert initialisiert wird und sich nach jedem Durchlauf des Schleifenkörpers verändert, d.h. meistens um einen bestimmten Wert (z.B. 1) erhöht oder vermindert wird, bis der definierte Zielwert erreicht ist. Man nennt diese Schleifenform auch Zählschleife, weil die Schleifenvariable und damit auch der Startwert, der Endwert und die Schrittweite numerisch sein müssen.

Im folgenden Beispiel sehen wir eine for-Schleife in C, die die Zahlen von 1 bis 100 ausdruckt:

```
for( i = 0; i < 100; i++)
   printf("i: %d\n", i);
```

Auch wenn Sie diese Schleifenform bereits in C oder einer anderen Sprache liebgewonnen haben, müssen wir Sie leider enttäuschen: Python kennt keine solche for-Schleife. Wohlgemerkt „keine solche", aber sehr wohl eine for-Schleife. Die in Python benutzte Art von for-Schleife entspricht der in der Bash-Shell oder in Perl verwendeten foreach-Schleife. Bei dieser Schleifenart handelt es sich um ein Sprachkonstrukt, mit dessen Hilfe nacheinander beispielsweise die Elemente einer Menge oder Liste bearbeitet werden können. Dazu werden sie einer Variable zugewiesen.

Im Folgenden sehen wir die allgemeine Syntax der for-Schleife in Python. Sequenz steht für ein iterierbares Objekt.

```
for Variable in Sequenz:
        Anweisung_1
        Anweisung_2
        ...
        Anweisung_n
else:
        Else-Anweisung_1
        Else-Anweisung_2
        ...
        Else-Anweisung_m
```

Wie bereits gesagt, dient in Python die for-Schleife zur Iteration über eine Sequenz von Objekten, während sie in vielen anderen Sprachen meist nur „eine etwas andere while-Schleife" ist.

Beispiel einer for-Schleife in Python:

```
>>> languages = ["C", "C++", "Perl", "Python"]
>>> for language in languages:
...     print(language)
...
C
C++
Perl
Python
>>>
```

Wie die while-Schleife besitzt auch die for-Schleife einen optionalen else-Block. Wie bei der while-Schleife wird der else-Block nur ausgeführt, wenn die Schleife nicht durch eine break-Anweisung abgebrochen wurde. Das bedeutet, dass der else-Block nur dann ausgeführt wird, wenn alle Elemente der Sequenz abgearbeitet worden sind.

Trifft der Programmablauf auf eine break-Anweisung, so wird die Schleife sofort verlassen und das Programm nach der Anweisung fortgesetzt, die der for-Schleife folgt, falls es überhaupt noch Anweisungen nach der for-Schleife gibt.

Üblicherweise befindet sich die break-Anweisung wie im folgenden Beispiel innerhalb einer Konditionalanweisung:

```
essbares = ["Schinken", "Speck", "Spinat", "Nüsse"]
for gericht in essbares:
    if gericht == "Spinat":
        print("Ich mag keinen Spinat!")
        break
    print("Lecker, " + gericht)
else:
    print("Glück gehabt, kein Spinat dabei gewesen!")
print("Jetzt bin ich satt!")
```

Wenn wir obiges Beispiel unter `kein_spinat.py` speichern und aufrufen, erhalten wir folgende Ausgaben:

```
bernd@marvin ~/beispiele/kein_spinat.py
Lecker, Schinken
Lecker, Speck
Ich mag keinen Spinat!
Jetzt bin ich satt!
$
```

Wenn wir in der Liste der essbaren Dinge „Spinat" durch „Schokolade" austauschen, erhalten wir folgende Ausgabe:

```
bernd@marvin ~/beispiele $ python3 kein_spinat.py
Lecker, Schinken
Lecker, Speck
Lecker, Schokolade
Lecker, Nüsse
Glück gehabt, kein Spinat dabei gewesen!
Jetzt bin ich satt!
```

Vielleicht ist unsere Abscheu vor Spinat aber doch nicht so groß, dass wir sofort aufhören zu essen, wie es in unserem Programm der Fall ist. In diesem Fall kommt die `continue`-Anweisung ins Spiel. Im folgenden kleinen Skript benutzen wir `continue`, um mit dem nächsten Artikel der essbaren Dinge weiterzumachen. „continue" schützt uns davor, „Spinat" essen zu müssen:

```
essbares = ["Schinken", "Speck", "Spinat", "Nüsse"]
for gericht in essbares:
    if gericht == "Spinat":
        print("Ich mag keinen Spinat!")
        continue
    print("Lecker, " + gericht)
else:
    print("Glück gehabt, kein Spinat dabei gewesen!")
print("Jetzt bin ich satt!")
```

Die Ausgabe sieht dann wie folgt aus:

```
bernd@marvin ~/dropbox/python-buch/dritte_auflage_neu/beispiele $
    python3 kein_spinat.py
Lecker, Schinken
Lecker, Speck
Ich mag keinen Spinat!
Lecker, Nüsse
Glück gehabt, kein Spinat dabei gewesen!
Jetzt bin ich satt!
$
```

Mit `for`-Schleifen können wir über beliebige iterierbare Objekte iterieren, wie wir in den folgenden Beispielen zeigen So können wir beispielsweise auch über Dictionaries iterieren, obwohl diese ja keine Reihenfolge aufweisen:

```
cities = {"London":   8615246,
          "Berlin":   3562166,
          "Madrid":   3165235,
          "Rome":     2874038}

for city in cities:
    print(city)
```

An der Ausgabe erkennen wir, dass eine Iteration der Iteration über die Keys entspricht:

```
bernd@marvin ~/tmp $ python3 for_examples.py
London
Rome
Madrid
Berlin
```

Iterieren wir über einen String, so tun wir dies buchstabenweise:

```
for char in "Berlin":
    print(char)
```

Die Ausgabe sieht wie folgt aus:

```
bernd@marvin ~/tmp $ python3 for_examples.py
B
e
r
l
i
n
```

10.6 Aufgaben

Für die erste Aufgabe benötigen wir die römischen Zahlen. Für diejenigen, die mit römischen Zahlen nicht so ganz sicher sind, geben wir hier die Zuordnungen in der folgenden Tabelle.

Tabelle 10.1 Römische Zahlen

Römische Zahl	Wert (Dezimalzahl)
I	1
II	2
III	3
IV	4
V	5
VI	6
VII	7
VIII	8
IX	9
X	10
XI	11
XIV	14

(Fortsetzung nächste Seite)

Tabelle 10.1 Römische Zahlen *(Fortsetzung)*

Römische Zahl	Wert (Dezimalzahl)
XV	15
XVI	16
…	…
XIX	19
XX	20
XXI	21
…	…
XXIX	29
XXX	30
XL	40
L	50
LX	60
XC	90
C	100
CC	200
CD	400
D	500
CM	900
M	1000
MM	2000

1. Aufgabe:

Schreiben Sie ein Python-Programm, das eine beliebige römische Zahl in eine „gewöhnliche" Dezimalzahl umrechnet.

Lösung: 40.4 (1. Aufgabe), Seite 482

2. Aufgabe:

Bild 10.2 Achtung: Frösche

In der nächsten Aufgabe lernen wir einen besonderen Frosch kennen, so wie ihn sich nur Mathematiker ausdenken können. Besonders seine Art, eine Straße zu über-

queren, macht es zweifelhaft, ob er in der realen Welt lange überleben könnte. Er überquert eine 2,50 Meter breite Straße wie folgt: Mit dem ersten Sprung legt er die erstaunliche Distanz von einem Meter zurück, dann springt er wegen zunehmender Erschöpfung mit jedem weiteren Schritt immer nur noch halb so weit wie vorher.

Die Entfernung, die er dabei zurücklegt, berechnet sich also als Summe der Werte $1 + 0,5 + 0.25 + 0,125$ und so weiter.

Dies entspricht natürlich in mathematischer Schreibweise der folgenden Notation:

$$\sum_{i=0}^{n} \frac{1}{2^i} = 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} \ldots$$

Versuchen Sie, mittels eines Python-Programms herauszubekommen, ob der Frosch es auf die andere Straßenseite schafft.

Lösung: 40.4 (2. Aufgabe), Seite 483

3. Aufgabe:

Der indische Herrscher Shihram tyrannisierte seine Untertanen und stürzte sein Land in Not und Elend. Um die Aufmerksamkeit des Königs auf seine Fehler zu lenken, ohne seinen Zorn zu entfachen, schuf Dahers Sohn, der weise Brahmane Sissa, ein Spiel, in dem der König als wichtigste Figur ohne Hilfe anderer Figuren und Bauern nichts ausrichten kann. Der Unterricht im Schachspiel machte auf Shihram einen starken Eindruck. Er wurde milder und ließ das Schachspiel verbreiten, damit alle davon Kenntnis nähmen. Um sich für die anschauliche Lehre von Lebensweisheit und zugleich Unterhaltung zu bedanken, gewährte er dem Brahmanen einen freien Wunsch. Dieser wünschte sich Weizenkörner: Auf das erste Feld eines Schachbretts wollte er ein Korn, auf das zweite Feld die doppelte Menge, also zwei, auf das dritte wiederum doppelt so viele, also vier und so weiter. Der König lachte und war gleichzeitig erbost über die vermeintliche Bescheidenheit des Brahmanen.

Als sich Shihram einige Tage später erkundigte, ob Sissa seine Belohnung in Empfang genommen habe, musste er hören, dass die Rechenmeister die Menge der Weizenkörner noch nicht berechnet hätten. Der Vorsteher der Kornkammer meldete nach mehreren Tagen ununterbrochener Arbeit, dass er diese Menge Getreidekörner im ganzen Reich nicht aufbringen könne. Auf allen Feldern eines Schachbretts zusammen wären es 18.446.744.073.709.551.615 Weizenkörner. Nun stellte er sich die Frage, wie das Versprechen eingelöst werden könne. Der Rechenmeister half dem Herrscher aus der Verlegenheit, indem er ihm empfahl, er solle Sissa ibn Dahir ganz einfach das Getreide Korn für Korn zählen lassen.[2]

Berechnen Sie mithilfe eines Python-Programms, ohne eine Formel zu Hilfe zu nehmen, wie viele Weizenkörner angehäuft werden müssen.

Lösung: 40.4 (3. Aufgabe), Seite 484

[2] Die Weizenkornlegende wurde wörtlich von Wikipedia entnommen.

Stichwortverzeichnis

^ bei regulären Ausdrücken 394
. Match eines beliebigen Zeichens 392
$ bei regulären Ausdrücken 394
% Modulo-Operator 23
16-Bit-Unicode-Zeichen → Escape-Zeichen
32-Bit-Unicode-Zeichen → Escape-Zeichen

A

Abfragemethode → Getter
Abgeleitete Klasse → Unterklasse
abort 441
__abs__ 236
abspath 455
Abstrakte Klassen 297
Abstrakte Methoden 297
access 441
__add__ 233, 235, 506
– Beispiel 236
all 350
Allgemeine Klasse → Basisklasse
Anagramm
– als Beispiel einer Permutation 328
__and__ 235
Änderungsmethode → Setter
Anführungszeichen → Escape-Zeichen
Angestelltenklasse 252→ Personklasse
Ankerzeichen 396
Anweisungsblöcke 67
Anzahl der Elemente einer Liste 40
Anzahl der Elemente eines Tupels 40
Anzahl der Zeichen eines Strings 40
append 143, 145
Archivdatei erzeugen 466
Arität 126
ASCII-Zeichen hexadezimal → Escape-Zeichen
ASCII-Zeichen oktal → Escape-Zeichen
Asimovsche Gesetze 228
assertAlmostEqual 378
assertCountEqual 378
AssertEqual 377
assertEqual 378
assertFalse 379
assertGreater 379
assertGreaterEqual 379
assertIn 379
AssertionError 376
assertIs 379
assertIsInstance 379
assertIsNone 379
assertIsNot 379
assertItemsEqual 379
assertLess 379
assertLessEqual 379
assertListEqual 379
assertNotRegexpMatches 379
assertTrue 379
assertTupleEqual 379
assoziative Arrays 43
assoziatives Feld 43
atime 457
Attribute 205
– dynamische Erzeugung 207
– private 217
– protected 217
– public 217
AttributeError 258
Aufspalten von Zeichenketten 176
Augmented Assignment → erweiterte Zuweisungen
augmented assignment 145
Ausgabe
– formatieren 93
– in Datei umleiten 95
– in Fehlerkanal umleiten 95
Auskellern 143
Ausnahme
– StopIteration 324

Ausnahmebehandlung 193
– except 193
– finally 198
– try 193
Automatentheorie 387

B
Barry 365
basename 455
Bash 426
Bash-Befehle ausführen 453
Basisklasse 251
bedingte Anweisungen 70
Bibliothek 158
Bierdeckelarithmetik 302
Bierdeckelnotation 302
binäre Operatoren 235
Binärsystem 302
Binärzahlen 25
Blöcke 67
Boehm 365
Boolesche Werte 26
Bourne-Shell 426
break 81
Bruch
– kürzen 243
– Kürzungszahl 243
– Repräsentierung in Python 242
– vollständig gekürzte Form 243
Bruchklasse 241
Bruchrechnen 242
bztar-Datei → Archivdatei erzeugen

C
C3 superclass linearization 273
Calendar-Klasse 264
CalendarClock-Klasse 264
__call__ 355
callable 355
Call-by-Reference 123
Call-by-Value 123
Caret-Zeichen 393
Cast 30
cast-Operator 66
Casting 30
Catullus 88
CaveInt 506→ Neanderthal-Arithmetik
center 187
chdir 441
chmod 441
Chomsky-Grammatik 133
chown 442
chroot 442
clear → Dictionary
CLI 425
Clock-Klasse 264
close 432
Closure 359
Codeblock 75, 417
commonprefix 456
complex 27, 236
continue 81
copy 464→ Dictionary
copy2 464
copy-Modul 111
copyfile 464
copyfileobj 464
copymode 464
copystat 464
copytree 464
count 147, 183
ctime 457

D
data hiding → Geheimnisprinzip
Datei 87
– lesen 87
– öffnen 87
– schreiben 89
– zum Schreiben öffnen 89
Dateibaum
– rekursiv durchwandern 454
Dateibearbeitung mit os 429
Dateideskriptor 432, 433
Dateigröße 459
Daten konservieren 381
Daten sichern mit Pickle 381
Datenabstraktion 212
Datenkapselung 204, 212
Datenpersistenz 381
Datentypen 21
Deadly Diamond of Death 272
Decorator → Dekorateur
deepcopy 111
Definition einer Variablen 22
Dekorateur 341
– Anwendungsfälle 350
– classmethod 261
– Einführung 341
– Funktionsdekorateur 341
– Klassendekorateur 341
– Memoisation 358
– mit Parametern 352

– zur Überprüfung von Funktionsargumenten 350
Dekoration
– mit Klasse 357
__del__ 229
deleter 220
Destruktor 211
Dezimalpunkt 26
Dezimalsystem 302
Diamond-Problem 272
Dictionary 43, 49
– aus Listen erzeugen 54
– clear 48
– copy 49
– flache Kopie 49, 112
– get 47
– items 50
– keys 50
– nested Dictionaries 48
– pop 50
– popitem 50
– setdefault 51
– tiefe Kopie 49, 112
– update 51
– verschachtelte Dictionaries 48
Differenz 27
dirname 456
Division
– ganzzahlige 23
__doc__ 115
– bei Dekorateuren 354
Docstring 115
– property 220
doctest 370
Donald Michie → Memoisation
dup 434
Dynamische Attribute 227
Dynamische Erzeugung von Klassen 281
dynamische Typdeklaration 29

E

Eigenschaften 205
Eingabe 65
Eingabeaufforderung
– robust 194
Eingabeprompt 7, 65
Einkellern 143
Einrückungen 67
Einschubmethode → Hook-Methode
Elternklasse → Basisklasse
endliche Automaten 387, 394
__eq__ 226, 235
erben → Vererbung
errare humanum est 365
erweiterte Zuweisungen 23, 235
Escape-Zeichen
– 16-Bit-Unicode-Zeichen 35
– 32-Bit-Unicode-Zeichen 35
– Anführungszeichen 35
– ASCII-Zeichen hexadezimal 35
– ASCII-Zeichen oktal 35
– Hochkomma 35
– Horizontaler Tabulator 35
– Rückschritt 35
– Seitenumbruch 35
– Unicode-Zeichen 35
– Vertikaler Tabulator 35
– Zeilenumbruch 35
escape-Zeichen 35
Euklidischer Algorithmus 243
eval 66
except 193
execl 440
execle 440
execlp 439
execlpe 440
execv 438
execve 438
execvp 436
execvpe 437
exists 456
expandvars 457
explizite Typumwandlung 30
extend 144
extsep 443

F

Fabrikfunktion 345
f-Strings 104
Fakultätsfunktion 133
– iterative Lösung 135
– rekursive Implementierung 134
False 26
Fehler
– Semantik 366
– Syntax 365
Fehlerarten 365
fib → Fibonacci-Modul
fiblist → Fibonacci-Modul
Fibonacci 136
Fibonacci-Folge → Fibonacci-Zahlen
– rekursive Berechnung 142
Fibonacci-Modul 367
– fib-Funktion 367

– fiblist-Funktion 367
Fibonacci-Zahlen
– formale mathematische Definition 136
– Generator 339, 520
– Modul zur Berechnung der Fibonacci-Zahlen 367
– rekursive Berechnung mit Dekorateuren 358
– rekursive Funktion 135
filter 307
finally 198
find 147, 183
Finite State Machine 394
firstn-Generator 327
flache Kopie 49, 112
__float__ 236
__floordiv__ 235
flüchtige Daten 381
Flussdiagramm 68
for-Schleife 80
– optionaler else-Teil 80
– StopIteration 324
– Unterschied zur while-Schleife 81
– Vergleich zu C 80
fork 443, 469
Forking 469
forkpty 443
formale Sprachen 387
format-Methode 97
Formatstring 95
Fraction-Klasse 250
fractions-Modul 250
fromkeys → Dictionary
Frosch
– Aufgabe mit Summenbildung 84
– Straßenüberquerung 84
frozensets 59
fully qualified 158
functools 313
Funktionen 113
– Attribute 207
– Fabrikfunktionen 345
– globale Variablen 120
– lokale Variablen 120
– Parameterübergabe 122
– Rückgabewerte 118
– statische Variablen 207
– überladen 253
– Überschreiben 253
– variadische 126
– verschachtelte Funktionen 342
– Zählen der Aufrufe 207
Funktionen als Parameter 343
Funktionsabschluss 359
Funktionsdekorateur → Dekorateur

G

Ganze Zahlen 24
ganzzahlige Division 23, 27
Gaußsche Summenformel 77
__ge__ 235
Geheimnisprinzip 212
Generator
– Allgemein 323
– CLU 323
– Endlosschleife in 326
– firstn 327
– Icon 323
– isclice zur Ausgabe von unendlichen Generatoren 327
– pair_sum 339
– pendulum 339
– Permutationen 328
– round_robin 339
– yield 325
– zur Erzeugung von Variationen 329
Generator-Ausdrücke 331
Generator-Comprehension 319
Generatoren
– return 332
– send 332
Generatoren-Abstraktion 319
get → Dictionary
get_archive_formats 465
getatime 457
getattr 207
getcwd 443, 462
getcwdb 443
getegid 443
getenv 427
getenvb 428
geteuid 443
get_exec_path 443
getgid 443
getgroups 443
getloadavg 444
getlogin 444
getmtime 458
getpgid 444
getpgrp 444
getpid 444
getppid 444
getresgid 444
getresuid 444
getsize ⇒ os.path

Getter 214
getuid 444
get_unpack_formats 465
ggT 243
größter gemeinsamer Teiler 243
Gruppierungen 401
__gt__ 235
GUI 425
gztar-Datei → Archivdatei erzeugen

H
hasattr 207
Hash 43
__hex__ 236
Hexadezimalzahlen 25
Hochkomma → Escape-Zeichen
Hook-Methode 377
Hooks → Hook-Methode
Horizontaler Tabulator → Escape-Zeichen

I
__iadd__ 235
__iand__ 235
id 22, 107
Identitätsfunktion 22, 107
__idiv__ 235
if-Anweisung 70
__ifloordiv__ 235
ignore_patterns 466
__ilshift__ 235
__imod__ 235
implizite Typumwandlung 30
import-Anweisung 158
__imul__ 235
in
– Dictionaries 46
index 147, 183
index-Funktionen 434
Inheritance → Vererbung
__init__ 233
__init__-Methode 209
__init__.py 164
input 46, 65
insert 148
Instanz 203, 205
Instanzattribute 205, 207, 227
Instanzvariablen 209
int 24, 236
Integer 24
interaktive Shell
– Befehlsstack 11
– Starten unter Linux 7
– Starten unter Windows 9
– Unterstrich 11
__invert__ 236
__ior__ 235
__ipow__ 235
__irshift__ 235
isabs 459
isalnum 188
isalpha 188
isdigit 188
isdir 459
isfile 459
isinstance 31
islice 327
– als Ersatz für firstn 327
islink 460
islower 188
ismount 460
is_palindrome 131
is_prime 350
isspace 188
istitle 188
__isub__ 235
isupper 188
items → Dictionary
itemgetter 154
__iter__ 324
Iteration
– for-Schleife 323
– über Dictionary 83
– über String 83
Iteratoren 323
– for-Schleife 323
itertools 327
– islice 327
__ixor__ 235

J
join 183, 460
Junit 375

K
kanonischer Pfad 462
Keller 143
Kellerspeicher 143, 261
keys → Dictionary
KeyError 45
kill 444
killpg 445
Kindklasse → Unterklasse
Klasse
– Instanz 205

– Kopf 204
– Körper 204
– minimale Klasse in Python 204
klassen
– abstrakte 297
Klassenattribute 206, 227
Klassendekorateur 357→ Dekorateur
Klassenmethoden 231, 259
Klassenobjekt 206
Kombination 329
Kommandozeilenparameter 125
Kompilierung von regulären Ausdrücken 407
Komplexe Zahlen 27
Komponententest 367
Konstruktor 209
Kontrollstrukturen 70
Kopieren
– von Unterlisten 110
– von verschachtelten Listen 107
Koroutinen 332
Kundenklasse → Personklasse
Kürzen 243

L

lambda 307
Länge von dictionaries 52
__le__ 235
len 52
len-Funktion 40
Leonardo von Pisa → Fibonacci-Zahlen
Lesbia 88
lexists 460
Liber Abaci 136
Lieferant → Personklasse
linesep 445
link 445
Linux 426
– Python unter 7
list
– append 143
– augmented assignment 145
– extend 144
– find 147
– index 147
– Packing 149
– pop 143
– sort 151
– Unpacking 149
List Comprehension 315
– Kreuzprodukt 317
– pythagoreisches Tripel 316
– Sieb des Eratosthenes 318
– Syntax 316
– Vergleich mit konventionellem Code 316
– Wandlung Celsius in Fahrenheit 316
– Weitere Beispiele 316
– zugrundeliegende Idee 317
list:count 147
list:insert 148
list:remove 147
list-Klasse 203
listdir 445
list_iterator 324
Listen-Abstraktion 315
– Kreuzprodukt 317
– pythagoreisches Tripel 316
– Sieb des Eratosthenes 318
– Syntax 316
– Vergleich mit konventionellem Code 316
– Wandlung Celsius in Fahrenheit 316
– Weitere Beispiele 316
– zugrundeliegende Idee 317
ljust 187
Löffelsprache 130
__long__ 236
lower 186
lseek 433
__lshift__ 235
lstat 445
__lt__ 235

M

Magische Methoden 233
– __add__ 233
– binäre Operatoren 235
– Erweiterte Zuweisungen 235
– __floordiv__ 235
– __init__ 209, 233
– __mod__ 235
– __mul__ 235
– __new__ 233
– __pow__ 235
– __repr__ 222, 233
– __str__ 222
– __sub__ 235
– __truediv__ 235
– unäre Operatoren 236
– Vergleichsoperatoren 235
Magische Operatoren
– Beispielklasse Length 236
__main__ 205, 367
major 445
make_archive 466
makedev 446

makedirs 447
Manager-Funktion 288
map 307, 310
Mapping 43
Marvin 504
Match eines beliebigen Zeichens 392
Match-Objekte 401
Matching-Problem 390
– Over Matching 390
– Under Matching 390
math-Modul 158
maxsplit
– split 176
Mehrfachschnittstellenvererbung 263
Mehrfachvererbung 263
– Beispiel CalendarClock 264
Mehrfachzuweisung 36
Memoisation 139, 358
– mit functools.lru_cache 360
– mit Klasse 359
Memoization → Memoisation
Memoize 359
memoize-Funktion 359
Mengen 57
– add 59
– clear 59
– copy 60
– difference 60
– difference_update 61
– isdisjoint 62
– issubset 63
– issuperset 63
– Obermenge 63
– pop 64
– remove 62
– Teilmenge 63
Mengen-Abstraktion 318
Mengenlehre 57
Metaklassen 285
Method Resolution Order 273, 276
Methoden 204, 208
– abstrakte 297
– __init__ 209
– Overloading → Überladen
– Overwriting → Überschreiben
– Überladen 257
– überlagern 253
– Überschreiben 257
– Unterschiede zur Funktion 208
minor 445
mkdir 446
mkfifo 447
__mod__ 235
modulare Programmierung 157
modulares Design 157
Modularisierung 157
Module 157
– dir() 161
– Dokumentation 162
– dynamisch geladene C-Module 159
– eigene Module schreiben 161
– Inhalt 161
– lokal 157
– math 158
– Modularten 159
– Namensraum 158
– re 389
– Suchpfad 160
__module__
– bei Dekorateuren 354
Modulo-Division 27
Modulo-Operator 23
Modultest 367
– unter Benutzung von __name__ 367
Monty Python 44
move 467
MRO 273, 276
mtime 457
__mul__ 235, 506
mypy 418

N

__name__ 205, 367
– bei Dekorateuren 354
Namensraum
– umbenennen 159
__ne__ 235
Neanderthal-Arithmetik 302
Nebeneffekte 124
__neg__ 236
Nested Dictionaries 48
__new__ 233
next 324
nice 447
Noam Chomsky 133
None 114
normcase 461
normpath 461
NotImplemented 247

O

Oberklasse 251
object 251
Objekt 203

Objektorientierte Programmiersprache 201
Objektorientierte Programmierung 201
__oct__ 236
Oktalzahlen 25
OOP 201
open 430
open() 87
openpty 432
operator-Modul 154
– itemgetter 154
Operatoren überladen
– binäre Operatoren 235
– erweiterte Zuweisungen 235
– unäre Operatoren 236
– Vergleichsoperatoren 235
Operatorüberladung 233
__or__ 235
os
– abort 441
– access 441
– chdir 441
– chmod 441
– chown 442
– chroot 442
– close 432
– dup 434
– execl 440
– execle 440
– execlp 439
– execlpe 440
– execv 438
– execve 438
– execvp 436
– execvpe 437
– extsep 443
– fork 443
– forkpty 443
– getcwd 443
– getcwdb 443
– getegid 443
– getenv 427
– getenvb 428
– geteuid 443
– get_exec_path 443
– getgid 443
– getgroups 443
– getloadavg 444
– getlogin 444
– getpgid 444
– getpgrp 444
– getpid 444
– getppid 444
– getresgid 444
– getresuid 444
– getuid 444
– kill 444
– killpg 445
– linesep 445
– link 445
– listdir 445
– lseek 433
– lstat 445
– major 445
– makedev 446
– makedirs 447
– minor 445
– mkdir 446
– mkfifo 447
– nice 447
– open 430
– openpty 432
– popen 447
– putenv 428
– read 432
– readlink 449
– remove 449
– removedirs 449
– rename 451
– renames 451
– rmdir 451
– sep 456
– setegid 451
– seteuid 451
– setgid 451
– setgroups 444
– setpgid 451
– setpgrp 451
– setregid 451
– setresgid 451
– setresuid 452
– setreuid 452
– setsid 452
– setuid 451
– stat 452
– stat_float_times 452
– strerror 452
– supports_bytes_environ 428
– symlink 453
– sysconf 453
– system 453
– times 453
– umask 453
– uname 454
– unlink 454

– unsetenv 428
– urandom 454
– utime 454
– wait 454
– wait3 454
– waitpid 454
– walk 454
– write 429
os-Modul 426, 441
os.path 455
os.python
– abspath 455
– basename 455
– commonprefix 456
– dirname 456
– exists 456
– expandvars 457
– getatime 457
– getmtime 458
– getsize 459
– isabs 459
– isdir 459
– isfile 459
– islink 460
– ismount 460
– join 460
– lexists 460
– normcase 461
– normpath 461
– realpath 461
– relpath 462
– samefile 462
– split 462
– splitdrive 463
– splitext 463
Over Matching 390
Overloading 253→ Überladen
Overwriting 253→ Überschreiben

P

Packing 149
Paket 164
Paketkonzept 164
Palindrome 131
Parameter 115
– beliebige Anzahl 126
– Defaultwert 115
– optional 115
– Schlüsselwort 118
– Standardwert 115
Parameterliste 114
Parameterübergabe 122
partion 183
Pascalsches Dreieck 141
– Fibonacci-Zahlen 141
peek 143
Permutationen 328
– mit Wiederholungen 328
– ohne Wiederholungen 328
Permutations-Generator 328
Persistente Daten 381
Personenklasse 252
Personklasse → Vererbung
Pfadname
– absolut 459
– relativ 459
pickle-Modul 381
Platzhalter 96
Polymorphismus 255
Polynom-Fabrikfunktion 345
pop 143, 261→ Dictionary
popen 447
popitem → Dictionary
__pos__ 236
Potenzieren 27
__pow__ 235
print 93
– Anweisung 93
– Ausgabe in Fehlerkanal 95
– end 93
– file 95
– Funktion 93
– sep 93
– sys.stderr 95
– Umlenkung in Datei 95
printf 95
private Attribute 217
Produkt 27
Programmablaufplan 68
Programmiersprache
– Unterschied zu Skriptsprache 17
Properties 215
– Definition mit Dekorateuren 221
property
– deleter 220
– mit Dekorateuren 221
protected Attribute 217
public Attribute 217
push 143, 261
putenv 428

Q

Quantoren 399
Quotient 27

R
__radd__ 238
random-Methode
– sample 330
raw-Strings 35
read 432
readlink 449
re-Modul 389
realpath 461
reduce 313
Referenzen 21
Regeln zur Interpretation von römischen Zahlen 482
register_archive_format 467
register_unpack_format 467
reguläre Ausdrücke 387
– Alternativen 407
– Anfang eines Strings matchen 394
– beliebiges Zeichen matchen 392
– compile 407
– Ende eines Strings matchen 394
– Kompilierung 407
– optionale Teile 398
– Qunatoren 399
– Teilstringsuche 389
– Überlappungen 389
– vordefinierte Zeichenklassen 397
– Wiederholungen von Teilausdrücken 399
– Zeichenauswahl 393
reguläre Auswahl
– Caret-Zeichen, Zeichenklasse 393
reguläre Mengen 387
reguläre Sprachen 387
Rekursion 133
– Beispiel aus der natürlichen Sprache 133
rekursiv → Rekursion
rekursive Funktion 133, 134
relpath 462
remove 147, 449
removedirs 449
rename 451
renames 451
replace 186
__repr__ 222, 233
Restdivision 27
return 114
rfind 183
rindex 183
rjust 187
rmdir 451
rmtree 467
Robot-Klasse 502
Roboter → Roboterklasse
Robotergesetze 228
Roboterklasse 204, 301
römische Zahlen 83
Rossum, Guido van 307
round_robin 339
__rshift__ 235
rsplit 179
– Folge von Trennzeichen 181
rstrip 187
Rückgabewerte 118
Rückschritt → Escape-Zeichen
Rückwärtsreferenzen 401

S
samefile 462
sample 330
Schachbrett mit Weizenkörnern 85
Schaltjahrberechnung 74, 266
Schaltjahre 74
Schleife
– break 81
– continue 81
Schleifen 70, 75
– Endekriterium 75
– for-Schleife 80
– kopfgesteuert 79
– Schleifendurchlauf 75
– Schleifenkopf 75
– Schleifenkörper 75
– Schleifenzähler 75
– while-Schleife 76
Schleifenzähler 75
Schlüssel
– Dictionary 43
– zulässige Typen 47
Schlüsselwortparameter 118
Seiteneffekte 124
Seitenumbruch → Escape-Zeichen
Sekunden
– Additon zu Uhrzeiten 74
semantische Fehler 366
sep 456
Sequentielle Datentypen 33
Sequenz 33
set 57
set comprehension 318
setdefault → Dictionary
setegid 451
seteuid 451
setgid 451
setgroups 444

setpgid 451
setpgrp 451
setregid 451
setresgid 451
setresuid 452
setreuid 452
sets
– add 59
– clear 59
– copy 60
– difference 60
– difference_update 61
– discard 61
– intersection 62
– isdisjoint 62
– issubset 63
– issuperset 63
– Operationen auf sets 59
– pop 64
– remove 62
setsid 452
Setter 214
setuid 451
setUp-Methode 377
Shell 425
– Bash 426
– Bourne 426
– C-Shell 426
– CLI 425
– GUI 425
– Herkunft und Bedeutung des Begriffes 425
Shell-Skripte ausführen 453
shelve-Modul 383
Shihram 85
shutil
– copy 464
– copy2 464
– copyfile 464
– copyfileobj 464
– copymode 464
– copystat 464
– copytree 464
– get_archive_formats 465
– get_unpack_formats 465
– ignore_patterns 466
– make_archive 466
– move 467
– register_archive_format 467
– register_unpack_format 467
– rmtree 467
– unpack_archive 468
– unregister_archive 468
– unregister_unpack_format 468
shutil-Modul 463
Sieb des Eratosthenes 318
– Rekursive Berechnung 141
– Rekursive Funktion mit Mengen-Abstraktion 319
silly_merge 420
Simula 67 201
singleton-Klasse 292
Skriptsprache
– Unterschied zu Programmiersprache 17
Slicing 38
Slots 279
sort 151
– eigene Sortierfunktionen 152
– reverse 152
– Umkehrung der Sortierreihenfolge 152
sorted 151
spezialisierte Klasse → Unterklasse
splice 303
split 176, 462
– Folge von Trennzeichen 181
– maxsplit 176
splitdrive 463
splitext 463
splitlines 182
Sprachfamilie 387
sprintf 95
Stack 143
– Stapelspeicher 143
Standardausnahmebehandlung 195
Standardbibliothek 158
Standardklassen als Basisklassen 261
Stapelspeicher 143, 261
stat 452
stat_float_times 452
staticmethod 229
Statische Attribute 227
Statische Methoden 229
statische Typ-Deklaration 29
Stelligkeit 126
Stephen Cole Kleene 387
StopIteration 324
str 26, 222, 242
strerror 452
Strings
– escape-Zeichen 35
– raw 35
– Suchen und Ersetzen 186
String-Tests 188
Stringinterpolation 95
Stringliterale 104

Stringmethoden
– Alles in Großbuchstaben 186
– Alles in Kleinbuchstaben 186
– capitalize 186
– center 187
– count 183
– find 183
– index 183
– isalnum 188
– isalpha 188
– isdigit 188
– islower 188
– isspace 188
– istitle 188
– isupper 188
– ljust 187
– lower 186
– replace 186
– rfind 183
– rindex 183
– rjust 187
– rstrip 187
– String-Tests 188
– strip 187
– title 186
– upper 186
– zfill 187
Stringmodulo 95
Strings 26
– formatieren 93
strip 187
Strukturierungselement 113
__sub__ 235, 506
Subklasse → Unterklasse
Suchen und Ersetzen 186
Suchen von Teilstrings 183
sum 76
Summe 27
Summe von n Zahlen 76
– Berechnung mit while-Schleife 76
SUnit 375
super 273
Superklasse → Basisklasse
supports_bytes_environ 428
symlink 453, 460
syntaktische Fehler 365
Syntax 365
– Fehler 365
Syntaxfehler 365
sysconf 453
system 453
Systemprogrammierung 425

T

tar-Datei → Archivdatei erzeugen
TDD → test-driven development
tearDown-Methode 377
Teilbereichsoperator 38
Tests 365
test first development 374
test-driven development 374
TestCase 377
– Methoden 377
– setUp-Methode 377
– tearDown-Methode 377
testCase 375
Testgesteuerte Entwicklung 374
Testgetriebene Entwicklung 373, 374
Testmethoden 377
Textverarbeitung 175
Theoretische Informatik 387
tiefe Kopie 49, 112
times 453
Transiente Daten 381
True 26
__truediv__ 235, 506
try 193
Tupel 36
– entpacken 36
– leere 149
– mit einem Element 149
– Packing 149
– Unpacking 149
tuple 36
Typ-Alias 421
Typ-Anmerkungen 417
– Any 422
– Callables 422
– Dict 422
– List 422
– Listen 420
– Variablen 419
Typ-Variablenanmerkungen 419
type 31, 203
type annotations 417
type conversion 30
type hints 417
type-Klasse 281
TypeError 369
– unhashable type 48
typing-Modul 421
Typumwandlung 30
– explizit 30
– implizit 30
Typverletzung 29

U
Überladen 253, 257
überlagern 253
Überlappungen 389
Überschreiben 253, 257
umask 453
Umgebungsvariablen 427
uname 454
unäre Operatoren 236
Unärsystem 302
Unary System → Unärsystem
Under Matching 390
unhashable type 48
Unicode-Zeichen → Escape-Zeichen
unittest 375→ Modultest
Unix 426
unlink 454
unpack_archive 468
Unpacking 149
unregister_archive_format 468
unregister_unpack_format 468
unsetenv 428
Unterklasse 251
Unterstrich
– Bedeutung in der interaktiven Shell 11
update → Dictionary
upper 186
urandom 454
utime 454

V
ValueError 194
Variable Anzahl von Parametern 126
Variablen 21
– Referenzen auf Objekte 21
Variablennamen 23
– Binnenversalien 24
– CamelCase 24
– gültige 23
– Konventionen 24
variadische Funktion 126
Variation 329
Vererbung 251
– Beispiel Angestelltenklasse, die von Person erbt 251
– Beispiel Kundenklasse, die von Person erbt 251
– Beispiel Lieferantenklasse, die von Person erbt 251
– Beispiel Personenklasse 251
Vererbungsbaum 263
Vergleichsoperatoren 72, 235
verschachtelte Dictionaries 48
Verschachtelte Funktionen 342
Vertikaler Tabulator → Escape-Zeichen
Verzeichnis löschen 451
Verzweigungen 70
Vollständige Induktion 133

W
wait 454
wait3 454
waitpid 454
walk 454
Weizenkornaufgabe 85
while-Schleife 76
– optionaler else-Teil 78
with-Anweisung 90
wraps-Dekorateur 355
write 429

X
__xor__ 235

Y
yield 325
– im Vergleich zur return-Anweisung 325

Z
Zahlenratespiel 78
Zeichenauswahl 393
Zeichenkette 26
Zeilenumbruch → Escape-Zeichen
Zeitrechnung 74
zfill 187
zip-Datei → Archivdatei erzeugen
zip-Funktion 52
zip-Klasse 52
Zugriffsmethoden 214
Zustandsautomat 394
Zustandsmaschine 394
Zuweisung 22