



Leseprobe

Bernd Klein

Einführung in Python 3

Für Ein- und Umsteiger

ISBN (Buch): 978-3-446-44133-0

ISBN (E-Book): 978-3-446-44151-4

Weitere Informationen oder Bestellungen unter

<http://www.hanser-fachbuch.de/978-3-446-44133-0>

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 Einführung in Python 3 – Für Ein- und Umsteiger	5
2 Kommandos und Programme	7
2.1 Erste Schritte mit Python	7
2.1.1 Linux	7
2.1.2 Windows	8
2.2 Herkunft und Bedeutung des Begriffes interaktive Shell	9
2.2.1 Erste Schritte in der interaktiven Shell	9
2.3 Verlassen der Python-Shell	11
2.4 Benutzung von Variablen	11
2.5 Mehrzeilige Anweisungen in der interaktiven Shell	12
2.6 Programme schreiben oder schnell mal der Welt “Hallo” sagen	12
3 Bytecode und Maschinencode	15
3.1 Einführung	15
3.2 Unterschied zwischen Programmier- und Skriptsprachen	15
3.3 Interpreter- oder Compilersprache	15

4	Datentypen und Variablen	19
4.1	Einführung	19
4.2	Datentypen	21
4.2.1	Ganze Zahlen	21
4.2.2	Fließkommazahlen	22
4.2.3	Zeichenketten	23
4.2.4	Boolesche Werte	23
4.2.5	Komplexe Zahlen	23
4.2.6	Operatoren	24
4.3	Statische und dynamische Typdeklaration	25
4.4	Typumwandlung	27
4.5	Datentyp ermitteln	27
5	Sequentielle Datentypen	29
5.1	Übersicht	29
5.1.1	Zeichenketten oder Strings	29
5.1.2	Listen	31
5.1.3	Tupel	32
5.1.4	Sequenz von Binärdaten	32
5.2	Indizierung von sequentiellen Datentypen	32
5.3	Slicing oder Ausschneiden	33
5.4	Die len-Funktion	35
5.5	Aufgaben	36
6	Dictionaries	39
6.1	Dictionaries und assoziative Felder	39
6.2	Definition und Benutzung	40
6.3	Fehlerfreie Zugriffe auf Dictionaries	42
6.4	Zulässige Typen für Schlüssel und Werte	43
6.5	Verschachtelte Dictionaries	44
6.6	Methoden auf Dictionaries	44
6.7	Operatoren	47
6.8	Die zip-Funktion	48
6.9	Dictionaries aus Listen erzeugen	49
6.10	Aufgaben	50

7	Mengen	53
7.1	Übersicht	53
7.2	Mengen in Python	53
7.2.1	Sets erzeugen	54
7.2.2	Mengen von unveränderlichen Elementen	54
7.3	Frozensets	55
7.4	Operationen auf „set“-Objekten	55
7.4.1	add(element)	55
7.4.2	clear()	55
7.4.3	copy	56
7.4.4	difference()	56
7.4.5	difference_update()	57
7.4.6	discard(el)	57
7.4.7	remove(el)	57
7.4.8	intersection(s)	58
7.4.9	isdisjoint()	58
7.4.10	issubset()	58
7.4.11	issuperset()	59
7.4.12	pop()	59
8	Eingaben	61
8.1	Eingabe mittels input	61
9	Verzweigungen	63
9.1	Anweisungsblöcke und Einrückungen	63
9.2	Bedingte Anweisungen in Python	66
9.3	Vergleichsoperatoren	67
9.4	Zusammengesetzte Bedingungen	68
9.5	Wahr oder falsch: Bedingungen in Verzweigungen	68
9.6	Aufgaben	69
10	Schleifen	71
10.1	Übersicht	71
10.2	while-Schleife	72
10.3	break und continue	73
10.4	die Alternative im Erfolgsfall: else	74
10.5	For-Schleife	75
10.6	Aufgaben	78

11	Dateien lesen und schreiben	81
11.1	Dateien	81
11.2	Text aus einer Datei lesen	81
11.3	Schreiben in eine Datei	83
11.4	In einem Rutsch lesen: readlines und read	83
11.5	Aufgaben	84
12	Formatierte Ausgabe und Strings formatieren	87
12.1	Wege, die Ausgabe zu formatieren	87
12.2	Details zur print-Funktion	88
12.2.1	Unterschiede zu Python 2	88
12.2.2	Import aus der Zukunft: print_function	88
12.2.3	print-Funktion in Python 3	89
12.3	Stringformatierung im C-Stil	90
12.4	Der pythonische Weg: Die String-Methode „format“	95
12.5	Benutzung von Dictionaries beim Aufruf der „format“-Methode	98
12.6	Benutzung von lokalen Variablen in „format“	99
12.7	Weitere String-Methoden zum Formatieren	100
13	Flaches und tiefes Kopieren	103
13.1	Einführung	103
13.2	Kopieren einer Liste	104
13.3	Kopie mit Teilbereichsoperator	106
13.4	Kopieren mit deepcopy	107
13.5	Deepcopy für Dictionaries	107
14	Funktionen	109
14.1	Allgemein	109
14.2	Funktionen in Python	109
14.3	Optionale- und Schlüsselwortparameter	111
14.4	Docstring	112
14.5	Rückgabewerte	113
14.6	Mehrere Rückgabewerte	114
14.7	Lokale und globale Variablen in Funktionen	114
14.8	Parameterübergabe im Detail	116
14.9	Effekte bei veränderlichen Objekten	118
14.10	Kommandozeilenparameter	119
14.11	Variable Anzahl von Parametern / Variadische Funktionen	120
14.12	* in Funktionsaufrufen	121

14.13	Beliebige Schlüsselwortparameter	122
14.14	Doppeltes Sternchen im Funktionsaufruf	122
14.15	Aufgaben	123
15	Rekursive Funktionen	125
15.1	Definition und Herkunft des Begriffs	125
15.2	Definition der Rekursion	126
15.3	Rekursive Funktionen in Python	126
15.4	Die Tücken der Rekursion	127
15.5	Fibonacci-Folge in Python	128
15.6	Aufgaben	132
16	Listen und Tupel im Detail	135
16.1	Stapelspeicher	135
16.2	Stapelverarbeitung in Python: pop und append	136
16.3	extend	136
16.4	„+“-Operator oder append	137
16.5	Entfernen eines Wertes	138
16.6	Prüfen, ob ein Element in Liste enthalten ist	139
16.7	Finden der Position eines Elementes	139
16.8	Einfügen von Elementen	140
16.9	Besonderheiten bei Tupel	140
16.9.1	Leere Tupel	141
16.9.2	1-Tupel	141
16.9.3	Mehrfachzuweisungen, Packing und Unpacking	141
16.10	Die veränderliche Unveränderliche	142
16.11	Sortieren von Listen	143
16.11.1	„sort“ und „sorted“	143
16.11.2	Umkehrung der Sortierreihenfolge	144
16.11.3	Eigene Sortierfunktionen	144
16.12	Aufgaben	147
17	Modularisierung	149
17.1	Module	149
17.1.1	Namensräume von Modulen	150
17.1.2	Namensräume umbenennen	151
17.1.3	Modularten	151
17.1.4	Suchpfad für Module	152
17.1.5	Inhalt eines Modules	153

17.1.6	Eigene Module	153
17.1.7	Dokumentation für eigene Module	154
17.2	Pakete.....	155
18	Globale und lokale Variablen	157
18.1	Einführung	157
18.2	Globale und lokale Variablen in Funktionen	157
19	Alles über Strings	161
19.1	... fast alles	161
19.2	Aufspalten von Zeichenketten	162
19.2.1	split	162
19.2.2	Standardverhalten und „maxsplit“	164
19.2.3	rsplit	165
19.2.4	Folge von Trennzeichen	167
19.2.5	splitlines	168
19.2.6	partition	168
19.3	Zusammenfügen von Stringlisten mit join	169
19.4	Suchen von Teilstrings	169
19.4.1	„in“ oder „not in“	169
19.4.2	s.find(substring[, start[, end]])	169
19.4.3	s.rfind(substring[, start[, end]])	170
19.4.4	s.index(substring[, start[, end]])	170
19.4.5	s.rindex(substring[, start[, end]])	171
19.4.6	s.count(substring[, start[, end]])	171
19.5	Suchen und Ersetzen	171
19.6	Nur noch Kleinbuchstaben oder Großbuchstaben	172
19.7	capitalize und title	172
19.8	Stripping Strings	173
19.9	Strings ausrichten.....	173
19.10	String-Tests	174
19.11	Aufgaben	176
20	Ausnahmebehandlung.....	179
20.1	Abfangen mehrerer Exceptions	181
20.2	except mit mehrfachen Ausnahmen	181
20.3	Die optionale else-Klausel	182
20.4	Fehlerinformationen über sys.exc_info.....	182
20.5	Exceptions generieren.....	183
20.6	Finalisierungsaktion.....	184

21	Objektorientierte Programmierung	185
21.1	Einführung	185
21.2	Klassen in Python	187
21.2.1	Objekte und Instanzen einer Klasse	187
21.2.2	Kapselung von Daten und Methoden	188
21.2.3	Ein minimale Klasse in Python	188
21.2.4	Eigenschaften und Attribute	189
21.3	Methoden	191
21.3.1	Instanzvariablen	191
21.3.2	Die <code>__init__</code> -Methode	193
21.4	Datenkapselung, Datenabstraktion und Geheimnisprinzip	195
21.4.1	Begriffsbestimmungen	195
21.4.2	Die <code>__str__</code> - und die <code>__repr__</code> -Methode	197
21.5	Public-, Protected- und Private-Attribute	201
21.6	Destruktor	205
21.7	Klassenattribute	207
21.8	Statische Methoden	210
21.9	Klassenmethoden	211
21.10	Properties	213
21.11	Public-Attribute statt private Attribute	217
21.12	Vererbung	219
21.12.1	Oberbegriffe und Oberklassen	219
21.12.2	Ein einfaches Beispiel	219
21.12.3	Überladen, Überschreiben und Polymorphie	220
21.12.4	Vererbung in Python	223
21.13	Mehrfachvererbung	226
21.13.1	Einführung	226
21.13.2	Beispiel: CalendarClock	227
21.13.3	Diamand-Problem oder „deadly diamond of death“	234
21.13.4	super und MRO	236
21.14	Magische Methoden und Operatorüberladung	240
21.14.1	Einführung	240
21.14.2	Übersicht magische Methoden	241
21.14.3	Beispielklasse: Length	242
21.15	Standardklassen als Basisklassen	245
21.16	Aufgaben	246

Teil II	Weiterführende Themen	249
22	Tests und Fehler	251
22.1	Einführung	251
22.2	Modultests	253
22.3	Modultests unter Benutzung von <code>__name__</code>	254
22.4	doctest-Modul	256
22.5	Testgetriebene Entwicklung oder „Im Anfang war der Test“	259
22.6	unittest	260
22.7	Methoden der Klasse <code>TestCase</code>	262
22.8	Aufgaben	265
23	Systemprogrammierung	267
23.1	Systemprogrammierung	267
23.2	Häufig falsch verstanden: Shell	267
23.3	os-Modul	268
23.3.1	Vorbemerkungen	268
23.3.2	Umgebungsvariablen	269
23.3.3	Dateiverarbeitung auf niedrigerer Ebene	271
23.3.4	Die <code>exec</code> -„Familie“	276
23.3.5	Weitere Funktionen im Überblick	282
23.3.6	<code>os.path</code> - Arbeiten mit Pfaden	295
23.4	shutil-Modul	303
24	Forks	309
24.1	Fork	309
24.2	Fork in Python	309
25	Daten konservieren	313
25.1	Persistente Speicherung	313
25.2	Pickle-Modul	314
25.2.1	Daten „einpökeln“ mit <code>pickle.dump</code>	314
25.2.2	<code>pickle.load</code>	315
25.3	Ein persistentes Dictionary mit <code>shelve</code>	315
26	Reguläre Ausdrücke	319
26.1	Ursprünge und Verbreitung	319
26.2	Stringvergleiche	319
26.3	Überlappungen und Teilstrings	321
26.4	Das <code>re</code> -Modul	321

26.5	Matching-Problem.....	322
26.6	Syntax der regulären Ausdrücke	324
26.6.1	Beliebiges Zeichen	324
26.7	Zeichenauswahl.....	324
26.8	Endliche Automaten.....	325
26.9	Anfang und Ende eines Strings.....	326
26.10	Vordefinierte Zeichenklassen	328
26.11	Optionale Teile	330
26.12	Quantoren	330
26.13	Gruppierungen und Rückwärtsreferenzen	333
26.13.1	Match-Objekte.....	333
26.14	Umfangreiche Übung	335
26.15	Alles finden mit findall	337
26.16	Alternativen.....	338
26.17	Kompilierung von regulären Ausdrücken	339
26.18	Aufspalten eines Strings mit oder ohne regulären Ausdruck	340
26.18.1	split-Methode der String-Klasse.....	340
26.18.2	split-Methode des re-Moduls	342
26.18.3	Wörter filtern	343
26.19	Suchen und Ersetzen mit sub	344
26.20	Aufgaben	345
27	lambda, map, filter und reduce	347
27.1	lambda	347
27.2	map	350
27.3	Filtern von sequentiellen Datentypen mittels „filter“	352
27.4	reduce	352
27.5	Aufgaben	354
28	Listen-Abstraktion/List Comprehension	355
28.1	Die Alternative zu Lambda und Co.	355
28.2	Syntax.....	356
28.3	Weitere Beispiele	356
28.4	Die zugrunde liegende Idee	357
28.5	Anspruchsvolleres Beispiel	358
28.6	Mengen-Abstraktion	358
28.7	Rekursive Primzahlberechnung	359
28.8	Generatoren-Abstraktion	359
28.9	Aufgaben	360

29	Generatoren und Iteratoren	361
29.1	Einführung	361
29.2	Iteration in for-Schleifen	361
29.3	Generatoren	363
29.4	Beispiele	365
29.4.1	Permutationen	365
29.4.2	Variationen und Kombinationen	366
29.5	Generatoren zählen mit firstn und islice	367
29.6	send-Methode	368
29.7	Generator-Ausdrücke	369
29.8	Aufgaben	370
30	Memoisation	371
30.1	Bedeutung und Herkunft des Begriffes	371
30.2	Memoisation mit Dekorateur-Funktionen	372
30.3	Memoize in einer Class	373
30.4	Dekoratore in Python	373
30.5	Überprüfung von Argumenten durch Dekoratore	375
31	NumPy	377
31.1	Übersicht	377
31.2	Arrays in NumPy	379
31.3	Arrays flach machen	381
31.4	Arrays umdimensionieren	383
31.5	Arrays konkatenieren	384
31.6	Array, neue Dimension hinzufügen	386
31.7	Array mit Nullen und Einsen initialisieren	386
31.8	Matrizenarithmetik	387
31.9	Vektoraddition und Vektorsubtraktion	388
31.10	Matrix-Klasse	390
31.10.1	Matrix-Produkt	391
31.10.2	Eine einfache praktische Anwendung	392
31.11	Inverse Matrix	393
31.12	Kreuzprodukt / Vektorprodukt	394
31.13	Lineare Gleichungssysteme	394
31.14	Polynome	396
31.15	Aufgaben	397

Teil III	Umfassende Beispiele	399
32	Bruchklasse	401
32.1	Brüche à la 1001 Nacht	401
32.2	Zurück in die Gegenwart	402
32.3	Rechenregeln	405
32.3.1	Multiplikation von Brüchen	405
32.3.2	Division von Brüchen	406
32.3.3	Addition von Brüchen	407
32.3.4	Subtraktion von Brüchen	407
32.4	Integer plus Bruch	408
32.4.1	Die Bruchklasse im Überblick	409
32.5	Die Fraction-Klasse im fractions-Modul	411
33	Mastermind	413
33.1	Die Ursprünge des Spiels	413
33.2	Die Spielregeln	414
33.2.1	„Bulls and Cows“	414
33.2.2	Mastermind	414
33.3	Kombinatorikmodul	415
33.4	Mastermind in Python	416
34	Textklassifikation	421
34.1	Einführung in die Textklassifikation	421
34.2	Textklassifikation: Aufgabe	423
34.3	Naive-Bayes-Klassifikator	423
34.3.1	Definition	423
34.3.2	Bayes-Theorem	423
34.4	Formale Herleitung der Naive-Bayes-Klassifikation	424
34.5	Textklassifikation in Python	426
34.5.1	BagOfWords-Klasse	426
34.5.2	Document-Klasse	427
34.5.3	DocumentClass-Klasse	429
34.5.4	Pool-Klasse	430

Teil IV	Lösungen zu den Aufgaben	433
35	Lösungen zu den Aufgaben	435
35.1	Lösungen zu Kapitel 5 (Sequentielle Datentypen)	435
35.2	Lösungen zu Kapitel 6 (Dictionaries)	438
35.3	Lösungen zu Kapitel 9 (Verzweigungen)	439
35.4	Lösungen zu Kapitel 10 (Schleifen)	442
35.5	Lösungen zu Kapitel 11 (Dateien lesen und schreiben)	445
35.6	Lösungen zu Kapitel 16 (Listen und Tupel im Detail)	446
35.7	Lösungen zu Kapitel 14 (Funktionen)	449
35.8	Lösungen zu Kapitel 15 (Rekursive Funktionen)	453
35.9	Lösungen zu Kapitel 19 (Alles über Strings ...)	457
35.10	Lösungen zu Kapitel 21 (Objektorientierte Programmierung)	460
35.11	Lösungen zu Kapitel 22 (Tests und Fehler)	468
35.12	Lösungen zu Kapitel 26 (Reguläre Ausdrücke)	468
35.13	Lösungen zu Kapitel 27 (lambda, map, filter und reduce)	474
35.14	Lösungen zu Kapitel 28 (Listen-Abstraktion/List Comprehension)	475
35.15	Lösungen zu Kapitel 29 (Generatoren und Iteratoren)	476
35.16	Lösungen zu Kapitel 31 (NumPy)	479
	Stichwortverzeichnis	481

Vorwort

Bedingt durch den traumhaften Anstieg der Bedeutung von Python in der Wissenschaft und Wirtschaft in den letzten Jahren gibt es auch ein wachsendes Interesse an geeigneten Python-Büchern. Gerade für Programmieranfänger besteht die Schwierigkeit darin, ein geeignetes Buch zu finden. Der unüberlegte Griff ins Bücherregal führt schnell zum Kauf eines Buches, was zwar viele Vorzüge haben mag, aber für Anfängerinnen und Anfänger völlig ungeeignet ist.

Dies ist besonders schade, da es sich bei Python ja um eine einfache Programmiersprache handelt. Das war bereits beim anfänglichen Design der Sprache eines der wesentlichen Ziele ihres „Erfinders“ Guido van Rossum, als er Python Anfang der 1990er Jahre am Zentrum für Mathematik und Informatik (Centrum voor Wiskunde en Informatica) in Amsterdam entwarf. Python ist einfach, weil es mit erstaunlich wenigen Schlüsselwörtern auskommt und seine Syntax, also der Aufbau der Sprache, auf Übersichtlichkeit optimiert ist.

Auch im Bereich objektorientierte Programmierung ist Python sehr konsequent, in manchen Dingen sogar konsequenter als Java. So sind in Python alle Objekte, d.h. es gibt keine primitiven Typen. Selbst Integer und Float-Zahlen sind in Python als Klassen realisiert. Ein weiterer Unterschied zu Java besteht darin, dass in Python Mehrfachvererbung möglich ist.

Auch Programme in anderen Sprachen lassen sich einfach als Module in Python einbetten. So kann man beispielsweise zeitkritische Algorithmen in C programmieren und sie dann in Python einbinden.

Zusammenfassend kann man sagen, dass es sich bei Python um eine Programmiersprache handelt, die sich bestens zum Einstieg in die Programmierung eignet, aber auch die optimale Lösung für Spezialisten aus zahlreichen Problemfeldern ist. In diesem Sinne finden Sie in diesem Buch den idealen Einstieg.

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 Teilnehmern an meinen Python-Kursen, die mir geholfen haben, meine didaktischen und fachlichen Kenntnisse kontinuierlich zu verbessern. Ebenso möchte ich den Besuchern meiner Online-Tutorials unter www.python-kurs.eu und www.python-course.eu danken, vor allem denjenigen, die sich mit konstruktiven Anmerkungen bei mir gemeldet haben. Allen voran Stefan Günther, der mir auch wertvolle Anregungen zur Erstauflage des Buches lieferte.

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

Für die Hilfe zur zweiten – weitestgehend überarbeiteten – Auflage möchte ich im besonderen Maße Herrn Jan Lendertse und Herrn Vincent Bermel Dank sagen. Von ihnen kamen viele wertvolle Anregungen und Änderungsvorschläge.

Bernd Klein, Singen

10

Schleifen

■ 10.1 Übersicht

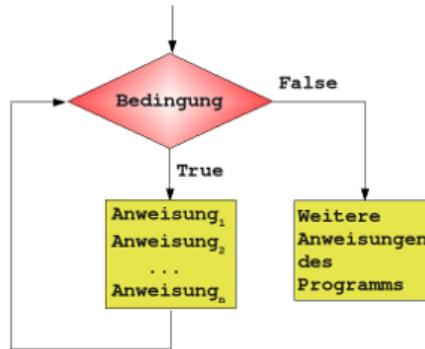
Schleifen werden benötigt, um einen Codeblock, also eine oder mehrere Python-Anweisungen, wiederholt auszuführen. Einen solchen Codeblock bezeichnet man auch als Schleifenkörper oder Body. In Python gibt es zwei Schleifentypen: die while-Schleife und die for-Schleife.

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 dem 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.



Bild 10.1 Karussell

■ 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 obigen 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 Schleifenrumpfes 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 weiter gegangen. Falls die Liste eine negative Zahl oder eine Null enthält, wird die Schleife komplett abgebrochen:

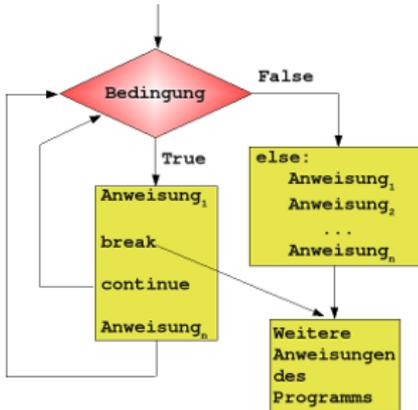
```
liste = eval(input("Liste mit positiven Zahlen eingeben: "))
n = len(liste)
i = 0
previous = None
erg = []
while i < n:
    current = liste[i]
    i += 1
    if current == previous:
        continue
    if current <= 0:
        print("Abbruch: Nicht-positive Zahl gefunden!")
        break
    erg.append(current)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

print(erg)
```

Zum besseren Verständnis des obigen Programmes 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-Programmierern¹ 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 = int(n * random.random()) + 1
```

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("New number: "))
    if guess > 0:
        if guess > to_be_guessed:
            print("Number too large")
        else:
            print("Number too small")
    else:
        print("Sorry, that you're giving up!")
        break
else:
    print("Congratulation. You made it!")
```

Die Ausgabe einer Spielsitzung könnte beispielsweise so aussehen:

```
$ python3 number_game.py
New number: 12
Number too small
New number: 15
Number too small
New number: 18
Number too large
New number: 17
Number too small
Congratulation. You made it!
$
```

■ 10.5 For-Schleife

Wie auch die while-Schleife ist die for-Schleife eine Kontrollstruktur, mit der eine Gruppe von Anweisungen (ein Block) wiederholt ausgeführt 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 lieb gewonnen haben, müssen wir Sie leider enttäuschen: Python kennt keine solche for-Schleife. Wohl-

gemerkt „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 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:

```
edibles = ["ham", "spam", "eggs", "nuts"]
for food in edibles:
    if food == "spam":
        print("No more spam please!")
        break
    print("Great, delicious " + food)
```

```

else:
    print("I am so glad: No spam!")
print("Finally, I finished stuffing myself")

```

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

```

$ python for.py
Great, delicious ham
No more spam please!
Finally, I finished stuffing myself
$

```

Wenn wir „spam“ aus der Liste der essbaren Dinge entfernen, erhalten wir folgende Ausgabe:

```

$ python for.py
Great, delicious ham
Great, delicious eggs
Great, delicious nuts
I am so glad: No spam!
Finally, I finished stuffing myself
$

```

Vielleicht ist unsere Abscheu vor dem Dosenfutter „spam“ nicht so groß, dass wir sofort aufhören zu essen. In diesem Fall kommt die `continue`-Anweisung ins Spiel. Im folgenden kleinen Skript benutzen wir `continue`, um mit dem nächsten Artikel der essbaren Artikel weiterzumachen. „`continue`“ schützt uns davor, „spam“ essen zu müssen:

```

edibles = ["ham", "spam", "eggs", "nuts"]
for food in edibles:
    if food == "spam":
        print("No more spam please!")
        continue
    print("Great, delicious " + food)
    # here can be the code for enjoying our food :-)
else:
    print("I am so glad: No spam!")
print("Finally, I finished stuffing myself")

```

Die Ausgabe sieht dann wie folgt aus:

```

$ python for.py
Great, delicious ham
No more spam please!
Great, delicious eggs
Great, delicious nuts
I am so glad: No spam!
Finally, I finished stuffing myself
$

```

■ 10.6 Aufgaben

Für die erste Aufgabe benötigen wir die römischen Zahlen. Für diejenigen, die sich nicht so ganz sicher sind mit römischen Zahlen, 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
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: [35.4 \(1. Aufgabe\)](#), Seite 442

**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 überqueren, 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:

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

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

Lösung: [35.4 \(2. Aufgabe\)](#), Seite 443

**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: [35.4 \(3. Aufgabe\)](#), Seite [444](#)

² Die Weizenkornlegende wurde wörtlich von Wikipedia entnommen.

Stichwortverzeichnis

^ bei regulären Ausdrücken 326
. Match eines beliebigen Zeichens 324
\$ bei regulären Ausdrücken 326
% Modulo-Operator 21
16-Bit-Unicode-Zeichen → Escape-Zeichen
32-Bit-Unicode-Zeichen → Escape-Zeichen

A

Abfragemethode → Getter
Abgeleitete Klasse → Unterklasse
abort 282
__abs__ 242
abspath 295
access 282
__add__ 240, 241, 464
– Beispiel 242
Allgemeine Klasse → Basisklasse
Anagramm
– als Beispiel einer Permutation 365
__and__ 241
Änderungsmethode → Setter
Anführungszeichen → Escape-Zeichen
Angestelltenklasse 219 → Personklasse
Ankerzeichen 328
Anweisungsblöcke 63
Anzahl der Elemente einer Liste 35
Anzahl der Elemente eines Tupels 35
Anzahl der Zeichen eines Strings 35
append 135, 137
Archivdatei erzeugen 306
Arität 120
Arrays 382
– flach machen → NumPy
– konkatenieren → NumPy
– ones() 386
– umdimensionieren → NumPy
– zeros() 386
ASCII-Zeichen hexadezimal →
Escape-Zeichen

ASCII-Zeichen oktal → Escape-Zeichen
Asimovsches Gesetz 207
assertAlmostEqual 263
assertCountEqual 263
AssertEqual 262
assertEqual 263
assertFalse 264
assertGreater 264
assertGreaterEqual 264
assertIn 264
AssertionError 261
assertIs 264
assertIsInstance 264
assertIsNone 264
assertIsNot 264
assertItemsEqual 264
assertLess 264
assertLessEqual 264
assertListEqual 265
assertNotRegexMatches 265
assertTrue 264
assertTupleEqual 265
assoziative Arrays 39
assoziatives Feld 39
atime 297
Attribute 189
– dynamische Erzeugung 190
– private 201
– protected 201
– public 201
AttributeError 225
Aufspalten von Zeichenketten 162
Augmented Assignment → Erweiterte
Zuweisung
augmented assignment 137
Ausgabe
– formatieren 87
– in Datei umleiten 90
– in Fehlerkanal umleiten 90

Auskellern 135
 Ausnahme
 – StopIteration 362
 Ausnahmebehandlung 179
 – except 179
 – finally 184
 – try 179
 Automatentheorie 319

B

Barry 251
 basename 296
 Bash 268
 Bash-Befehle ausführen 294
 Basisklasse 219
 Bayes-Theorem 423
 bedingte Anweisungen 66
 Bibliothek 150
 Bierdeckelarithmetik 247
 Bierdeckelnotation 247
 binäre Operatoren 241
 Binärsystem 247
 Binärzahlen 21
 Blöcke 63
 Boehm 251
 Boolesche Werte 23
 Bourne-Shell 268
 Bruch
 – kürzen 403
 – Kürzungszahl 403
 – Repräsentierung in Python 402
 – vollständig gekürzte Form 403
 Bruchklasse 401
 Bruchrechnen 402
 Bulls and Cows 414
 bztar-Datei → Archivdatei erzeugen

C

C3 superclass linearization 236
 Calendar-Klasse 227
 CalendarClock-Klasse 227
 Call-by-Reference 117
 Call-by-Value 117
 Caret-Zeichen 325
 Cast 27
 cast-Operator 61
 Casting 27
 Catullus 82
 CaveInt 464 → Neanderthal-Arithmetik
 center 173
 chdir 282
 chmod 283

Chomsky-Grammatik 125
 chown 283
 chroot 283
 clear → Dictionary
 CLI 267
 Clock-Klasse 227
 close 274
 Closure 372
 Codeblock 71
 combinatorics.py 415
 commonprefix 296
 complex 23, 242
 copy 304 → Dictionary
 copy2 304
 copy-Modul 107
 copyfile 304
 copyfileobj 304
 copymode 304
 copystat 304
 copytree 304
 count 139, 169
 ctime 297

D

data hiding → Geheimnisprinzip
 Datei 81
 – lesen 81
 – öffnen 81
 – schreiben 83
 – zum Schreiben öffnen 83
 Dateibaum
 – rekursiv durchwandern 295
 Dateibearbeitung mit os 271
 Dateideskriptor 274
 Dateigröße 299
 Daten konservieren 313
 Daten sichern mit Pickle 313
 Datenabstraktion 195
 Datenkapselung 188, 195
 Datenpersistenz 313
 Datentypen 19
 Deadly Diamond of Death 234
 Decorator → Dekorateur
 deepcopy 107
 Definition einer Variablen 20
 Dekorateur 372
 – Beispiel Fakultätsfunktion 374
 – Benutzung in Python 373
 – zur Implementierung eine Memoisation 374
 – zur Überprüfung von Funktionsargumenten 374

Dekorator
– staticmethod 211
__del__ 208
deleter 215
deriv 397
derivation 397
Destruktor 205
Dezimalpunkt 22
Dezimalsystem 247
Diamond-Problem 234
Dictionary 39, 45
– aus Listen erzeugen 49
– clear 44
– copy 44
– flache Kopie 45, 107
– get 43
– items 45
– keys 45
– nested Dictionaries 43
– pop 46
– popitem 46
– setdefault 46
– tiefe Kopie 45, 107
– update 47
– verschachtelte Dictionaries 43
Differenz 24
dirname 296
__div__ 241
Division
– ganzzahlige 21
__doc__ 112
Docstring 112
– property 215
doctest 256
Dokumentenklassifikation 421
Donald Michie → Memoisation
Drei-Finger-Regel 394
dup 276
Dynamische Attribute 207
dynamische Typdeklaration 25, 26

E

Eigenschaften 189
Eingabe 61
Eingabeaufforderung
– robust 180
Eingabeprompt 7, 61
Einheitsmatrix 393
Einkellern 135
Einrückungen 63
Einschubmethode → Hook-Methode
Elternklasse → Basisklasse

endliche Automaten 319, 325
__eq__ 242
erben → Vererbung
errare humanum est 251
erweiterte Zuweisungen 20, 241
Escape-Zeichen
– 16-Bit-Unicode-Zeichen 31
– 32-Bit-Unicode-Zeichen 31
– Anführungszeichen 31
– ASCII-Zeichen hexadezimal 31
– ASCII-Zeichen oktala 31
– Hochkomma 31
– Horizontaler Tabulator 31
– Rückschritt 31
– Seitenumbruch 31
– Unicode-Zeichen 31
– Vertikaler Tabulator 31
– Zeilenumbruch 31
escape-Zeichen 31
Euklidischer Algorithmus 403
eval 62
except 179
execl 281
execle 281
execlp 280
execlpe 281
execv 279
execve 280
execvp 277
execvpe 278
exists 297
expandvars 297
explizite Typumwandlung 27
extend 136
extsep 283

F

Fakultätsfunktion 125
– iterative Lösung 127
– rekursive Implementierung 126
– unter Benutzung eines Dekorateurs 374
False 23
Fehler
– Semantik 252
– Syntax 251
Fehlerarten 251
fib → Fibonacci-Modul
fiblist → Fibonacci-Modul
Fibonacci 128
Fibonacci-Folge → Fibonacci-Zahlen
– rekursive Berechnung 133

- Fibonacci-Modul 253
 - fib-Funktion 253
 - fiblist-Funktion 253
 - Fibonacci-Zahlen
 - formale mathematische Definition 128
 - Generator 370, 476
 - Modul zur Berechnung der Fibonacci-Zahlen 253
 - rekursive Berechnung mit Dekoratoren 372
 - rekursive Funktion 127
 - filter 347
 - finally 184
 - find 139, 169
 - Finite State Machine 326
 - firstn-Generator 367
 - flache Kopie 45, 107
 - __float__ 242
 - __floordiv__ 241
 - flüchtige Daten 313
 - Flussdiagramm 63
 - for-Schleife 75
 - optionaler else-Teil 76
 - StopIteration 362
 - Unterschied zur while-Schleife 76
 - Vergleich zu C 75
 - fork 284, 309
 - Forking 309
 - forkpty 284
 - formale Sprachen 319
 - format-Methode 92
 - Formatstring 91
 - Fraction-Klasse 411
 - fractions-Modul 411
 - fromkeys → Dictionary
 - Frosch
 - Aufgabe mit Summenbildung 79
 - Straßenüberquerung 79
 - frozensets 55
 - fully qualified 150
 - functools 352
 - Funktionen 109
 - Attribute 190
 - globale Variablen 114
 - lokale Variablen 114
 - Parameterübergabe 116
 - Rückgabewerte 113
 - statische Variablen 190
 - Überladen 220
 - Überschreiben 220
 - variadische 120
 - Zählen der Aufrufe 190
 - Funktionsabschluss 372
- G**
- Ganze Zahlen 21
 - ganzzahlige Division 21, 24
 - Gaußsche Summenformel 73
 - __ge__ 242
 - Geheimnisprinzip 195
 - Generator
 - all_colours() 415
 - Allgemein 361
 - CLU 361
 - Endlosschleife in 363
 - firstn 367
 - Icon 361
 - islice zur Ausgabe von unendlichen Generatoren 368
 - k-Permutationen 415
 - k-Permutationen 415
 - k_permutations() 415
 - pair_sum 370
 - pendulum 370
 - Permutationen 365, 415
 - permutations() 415
 - round_robin 370
 - send-Methode 368
 - Werte empfangen 368
 - yield 363
 - zur Erzeugung von Selektionen 366
 - Generator-Ausdrücke 369
 - Generator-Comprehension 359
 - Generatoren-Abstraktion 359
 - get → Dictionary
 - get_archive_formats 305
 - getatime 297, 298
 - getcwd 284, 302
 - getcwdb 284
 - getegid 284
 - getenv 269
 - getenvb 269
 - geteuid 284
 - get_exec_path 284
 - getgid 284
 - getgroups 284
 - getloadavg 285
 - getlogin 285
 - getmtime 298
 - getpgid 285
 - getpgrp 285
 - getpid 285
 - getppid 285
 - getresgid 285
 - getresuid 285
 - getsize ⇒ os.path

Getter 195
getter 213
getuid 285
get_unpack_formats 306
ggT 403
Gleichungssysteme 394
grafische Veranschaulichung von Matrixwerten
→ Hinton-Diagramm
größter gemeinsamer Teiler 403
Gruppierungen 333
__gt__ 242
GUI 267
gztar-Datei → Archivdatei erzeugen

H

Hash 39
__hex__ 242
Hexadezimalzahlen 22
Hinton-Diagramm 377
Hochkomma → Escape-Zeichen
Hooks → Hook-Methode
Hook-Methode 262
Horizontaler Tabulator → Escape-Zeichen

I

__iadd__ 241
__iand__ 241
id 103
Identitätsfunktion 103
__idiv__ 241
if-Anweisung 66
__ifloordiv__ 241
ignore_patterns 306
__ilshift__ 241
__imod__ 241
implizite Typumwandlung 27
import-Anweisung 150
__imul__ 241
in
– Dictionaries 42
index 139, 169
index-Funktionen 276
Inheritance → Vererbung
__init__ 240
__init__-Methode 193
__init__.py 155
inneres Produkt 389
input 42, 61
insert 140
Instanz 187, 189
Instanzattribute 190, 207
Instanzvariablen 191

int 21, 242
integ 397
Integer 21
interaktive Shell
– Befehlsstack 11
– Starten unter Linux 7
– Starten unter Windows 8
– Unterstrich 10
inverse Matrix 393
__invert__ 242
__ior__ 241
__ipow__ 241
__irshift__ 241
isabs 299
isalnum 174
isalpha 174
isdigit 174
isdir 300
isfile 300
isinstance 27
islice 368
– als Ersatz für firstn 368
islink 300
islower 174
ismount 300
is_palindrome 124
isspace 174
istitle 174
__isub__ 241
isupper 174
items → Dictionary
itemgetter 145
__iter__ 362
Iteration
– for-Schleife 361
Iteratoren 361
– for-Schleife 361
itertools 368
– islice 368
__ixor__ 241

J

join 168, 300
Junit 260

K

kanonischer Pfad 302
Keller 135
Kellerspeicher 135, 245
keys → Dictionary
KeyError 41
kill 285

- killpg 286
- Kindklasse → Unterklasse
- Klasse
 - Instanz 189
 - Kopf 188
 - Körper 188
 - minimale Klasse in Python 188
- Klassenattribute 207
- Klassenmethoden 211
- Klassenobjekt 190
- Kombination 366
- Kombinatorik-Modul 415
- Kommandozeilenparameter 119
- Kompilierung von regulären Ausdrücken 339
- Komplexe Zahlen 23
- Komponententest 253
- Konstruktor 193
- Kontrollstrukturen 66
- Kopieren
 - von Unterlisten 106
 - von verschachtelten Listen 103
- Kreuzprodukt 389, 394
- Kundenklasse → Personklasse
- Kürzen 403

- L**
- lambda 347
- Länge eines Vektors 389
- Länge von dictionaries 47
- __le__ 242
- len 47
- len-Funktion 35
- Leonardo von Pisa → Fibonacci-Zahlen
- Lesbia 82
- lexists 301
- Liber Abaci 128
- Lieferant → Personklasse
- linalg → NumPy
- lineare Algebra 394
- lineare Gleichungssysteme 394
- linesep 286
- link 286
- Linux 268
 - Python unter 7
- list
 - append 135
 - augmented assignment 137
 - extend 136
 - find 139
 - index 139
 - Packing 141
 - pop 135
 - sort 143
 - Unpacking 141
- List Comprehension 355
 - Kreuzprodukt 357
 - pythagoreisches Tripel 356
 - Sieb des Eratosthenes 358
 - Syntax 356
 - Vergleich mit konventionellem Code 356
 - Wandlung Celsius in Fahrenheit 356
 - Weitere Beispiele 356
 - zugrundeliegende Idee 357
- list:count 139
- list:insert 140
- list:remove 138
- list-Klasse 187
- listdir 286
- list_iterator 362
- Listen-Abstraktion 355
 - Kreuzprodukt 357
 - pythagoreisches Tripel 356
 - Sieb des Eratosthenes 358
 - Syntax 356
 - Vergleich mit konventionellem Code 356
 - Wandlung Celsius in Fahrenheit 356
 - Weitere Beispiele 356
 - zugrundeliegende Idee 357
- ljust 173
- Löffelsprache 123
- __long__ 242
- Lösung von Gleichungssystemen 394
- lower 172
- lseek 274
- __lshift__ 241
- lstat 286
- __lt__ 242

- M**
- Magische Methoden 240
 - __add__ 240
 - binäre Operatoren 241
 - __div__ 241
 - Erweiterte Zuweisungen 241
 - __floordiv__ 241
 - __init__ 193, 240
 - __mod__ 241
 - __mul__ 241
 - __new__ 240
 - __pow__ 241
 - __repr__ 197, 240
 - __str__ 197
 - __sub__ 241
 - unäre Operatoren 242

- Vergleichsoperatoren 242
 - Magische Operatoren
 - Beispielklasse Length 242
 - `__main__` 189, 253
 - major 286
 - make_archive 306
 - makedev 287
 - makedirs 287
 - map 347, 350
 - Mapping 39
 - Marvin 462
 - Mastermind 413
 - Match eines beliebigen Zeichens 324
 - Match-Objekte 333
 - Matching-Problem 322
 - Over Matching 322
 - Under Matching 322
 - math-Modul 150
 - MATLAB 377
 - Matrix-Klasse → NumPy
 - Matrix-Produkt → NumPy
 - Matrizenaddition 387
 - Matrizenmultiplikation 387, 391
 - Matrizensubtraktion 387
 - maxsplit
 - split 162
 - Mehrfachschnittstellenvererbung 226
 - Mehrfachvererbung 226
 - Beispiel CalendarClock 227
 - Memoisation 131, 371
 - Memoization → Memoisation
 - Memoize 373
 - memoize-Funktion 372
 - Mengen 53
 - add 55
 - clear 55
 - copy 56
 - difference 56
 - difference_update 57
 - isdisjoint 58
 - issubset 58
 - issuperset 59
 - Obermenge 59
 - pop 59
 - remove 57
 - Teilmenge 58
 - Mengen-Abstraktion 358
 - Mengenlehre 53
 - Method Resolution Order 236, 239
 - Methoden 188, 191
 - `__init__` 193
 - Overloading → Überladen
 - Overwriting → Überschreiben
 - Parameter self 192
 - Überladen 224
 - Überschreiben 224
 - Unterschiede zur Funktion 191
 - minor 286
 - mkdir 287
 - mkfifo 288
 - `__mod__` 241
 - modulare Programmierung 149
 - modulares Design 149
 - Modularisierung 149
 - Module 149
 - `dir()` 153
 - Dokumentation 154
 - dynamisch geladene C-Module 151
 - eigene Module schreiben 153
 - Inhalt 153
 - lokal 150
 - math 150
 - Modulararten 151
 - Namensraum 150
 - re 321
 - Suchpfad 152
 - Modulo-Division 24
 - Modulo-Operator 21
 - Modultest 253
 - unter Benutzung von `__name__` 253
 - Monty Python 40
 - move 307
 - MRO 236, 239
 - mtime 297
 - `__mul__` 241, 464
- ## N
- Naive-Bayes-Klassifikator 423
 - `__name__` 189, 253
 - Namensraum
 - umbenennen 151
 - `__ne__` 242
 - Neanderthal-Arithmetik 247
 - Nebeneffekte 118
 - `__neg__` 242
 - Nested Dictionaries 43
 - `__new__` 240
 - next 362
 - nice 288
 - Noam Chomsky 125
 - None 110
 - normcase 301
 - normpath 302
 - Nullstellen berechnen 397

- Numarray → NumPy
- Numeric → NumPy
- Numeric Python → NumPy
- NumPy 377
 - Arrays 379
 - Array neue Dimension hinzufügen 386
 - Arrays flach machen 381
 - Arrays konkatenieren 384
 - Array mit Nullen und Einsen initialisieren 386
 - Arrays umdimensionieren 382
 - deriv 397
 - Drei-Finger-Regel 394
 - Einheitsmatrix 393
 - eye 393
 - Gleichungssysteme 394
 - inneres Produkt 389
 - integ 397
 - inv 393
 - inverse Matrix 393
 - Kreuzprodukt 389, 394
 - Länge eines Vektors 389
 - linalg 394
 - lineare Algebra 394
 - lineare Gleichungssysteme 394
 - Lösung von Gleichungssystemen 394
 - Matrix-Klasse 390
 - Matrix-Produkt 391
 - Matrixarithmetik 387
 - Matrizenaddition 387
 - Matrizenmultiplikation 387, 391
 - Matrizensubtraktion 387
 - Nullstellen 395
 - Nullstellen berechnen 397
 - ones() 386
 - ones_like() 387
 - polynomial-Paket 395
 - Punktprodukt 389
 - Rechtssystem 394
 - roots 397
 - Skalarprodukt 389
 - Vektoraddition 388
 - Vektorprodukt 394
 - Vektorsubtraktion 388
 - Winkel zwischen Vektoren 390
 - zeros() 386
 - zeros_like() 387
- O**
- Oberklasse 219
- object 219
- Objekt 187
- Objektorientierte Programmiersprache 185
- Objektorientierte Programmierung 185
- __oct__ 242
- Oktalzahlen 22
- ones_like() 387
- OOP 185
- open 272
- open() 81
- openpty 274
- operator-Modul 145
 - itemgetter 145
- Operatoren überladen
 - binäre Operatoren 241
 - erweiterte Zuweisungen 241
 - unäre Operatoren 242
 - Vergleichsoperatoren 242
- Operatorüberladung 240
- __or__ 241
- os
 - abort 282
 - access 282
 - chdir 282
 - chmod 283
 - chown 283
 - chroot 283
 - close 274
 - dup 276
 - execl 281
 - execlp 281
 - execlpe 281
 - execv 279
 - execve 280
 - execvp 277
 - execvpe 278
 - extsep 283
 - fork 284
 - forkpty 284
 - getcwd 284
 - getcwdb 284
 - getegid 284
 - getenv 269
 - getenvb 269
 - geteuid 284
 - get_exec_path 284
 - getgid 284
 - getgroups 284
 - getloadavg 285
 - getlogin 285
 - getpgid 285
 - getpgrp 285
 - getpid 285

- getppid 285
- getresgid 285
- getresuid 285
- getuid 285
- kill 285
- killpg 286
- linesep 286
- link 286
- listdir 286
- lseek 274
- lstat 286
- major 286
- makedev 287
- mkdir 287
- minor 286
- mkdir 287
- mkfifo 288
- nice 288
- open 272
- openpty 274
- popen 288
- putenv 270
- read 274
- readlink 290
- remove 290
- removedirs 290
- rename 291
- renames 291
- rmdir 291
- sep 296
- setegid 292
- seteuid 292
- setgid 292
- setgroups 285
- setpgid 292
- setpgrp 292
- setregid 292
- setresgid 292
- setresuid 292
- setreuid 292
- setsid 292
- setuid 292
- stat 292
- stat_float_times 293
- strerror 293
- supports_bytes_environ 270
- symlink 293
- sysconf 294
- system 294
- times 294
- umask 294
- uname 294

- unlink 294
- unsetenv 270
- urandom 294
- utime 294
- wait 295
- wait3 295
- waitpid 295
- walk 295
- write 271
- os-Modul 268, 282
- os.path 295
- os.python
 - abspath 295
 - basename 296
 - commonprefix 296
 - dirname 296
 - exists 297
 - expandvars 297
 - getatime 297, 298
 - getmtime 298
 - getsize 299
 - isabs 299
 - isdir 300
 - isfile 300
 - islink 300
 - ismount 300
 - join 300
 - lexists 301
 - normcase 301
 - normpath 302
 - realpath 302
 - relpath 302
 - samefile 302
 - split 303
 - splitdrive 303
 - splitext 303
- Over Matching 322
- Overloading 220→ Überladen
- Overwriting 220→ Überschreiben

P

- Packing 141
- Paket 155
- Paketkonzept 155
- Palindrome 124
- Parameter 111
 - beliebige Anzahl 120
 - optional 111
 - Schlüsselwort 111
- Parameterliste 110
- Parameterübergabe 116
- partition 168

- Pascalsches Dreieck 132
- Fibonacci-Zahlen 133
- peek 135
- Permutationen 365
- mit Wiederholungen 365
- ohne Wiederholungen 365
- Permutations-Generator 365
- Persistente Daten 313
- Personenklasse 219
- Personklasse → Vererbung
- Pfadname
- absolut 299
- relativ 299
- pickle-Modul 313
- Platzhalter 92
- Polymorphismus 222
- Polynome 395
- pop 135, 245 → Dictionary
- popen 288
- popitem → Dictionary
- __pos__ 242
- Potenzieren 24
- __pow__ 241
- print 88
- Anweisung 88
- Ausgabe in Fehlerkanal 90
- end 89
- file 90
- Funktion 88
- sep 89
- sys.stderr 90
- Umlenkung in Datei 90
- Unterschiede zu Python 2 88
- printf 90
- private Attribute 201
- lesender und schreibender Zugriff mittels
 Methoden 213
- Produkt 24
- Programmablaufplan 63
- Programmiersprache
- Unterschied zu Skriptsprache 15
- Properties 213
- property
- deleter 215
- mit Dekoratoren 216
- protected Attribute 201
- public Attribute 201
- Punktprodukt 389
- push 135, 245
- putenv 270

Q

- Quantoren 330
- Quotient 24

R

- __radd__ 244, 408
- random-Methode
- sample 367
- raw-Strings 31
- read 274
- readlink 290
- re-Modul 321
- realpath 302
- Rechtssystem 394
- reduce 352
- Regeln zur Interpretation von römischen
 Zahlen 442
- register_archive_format 307
- register_unpack_format 307
- reguläre Ausdrücke 319
- Alternativen 338
- Anfang eines Strings matchen 326
- beliebiges Zeichen matchen 324
- compile 339
- Ende eines Strings matchen 326
- Kompilierung 339
- optionale Teile 330
- Quantoren 330
- Teilstringsuche 321
- Überlappungen 321
- vordefinierte Zeichenklassen 328
- Wiederholungen von Teilausdrücken 330
- Zeichenauswahl 324
- reguläre Auswahl
- Caret-Zeichen, Zeichenklasse 325
- reguläre Mengen 319
- reguläre Sprachen 319
- Rekursion 125
- Beispiel aus der natürlichen Sprache 125
- rekursiv → Rekursion
- rekursive Funktion 125, 126
- relpath 302
- remove 138, 290
- removedirs 290
- rename 291
- renames 291
- replace 171
- __repr__ 197, 240
- Restdivision 24
- return 110
- rfind 169
- rindex 169

rjust 173
rmdir 291
rmtree 308
Robot-Klasse 461
Roboter → Roboterklasse
Roboter Gesetze 207
Roboterklasse 188, 246
römische Zahlen 78
roots 397
Rossum, Guido van 347
round_robin 370
__rshift__ 241
rsplit 165
– Folge von Trennzeichen 167
rstrip 172
Rückgabewerte 113
Rückschritt → Escape-Zeichen
Rückwärtsreferenzen 333

S

samefile 302
sample 367
Schachbrett mit Weizenkörnern 79
Schaltjahr Berechnung 69, 229
Schaltjahre 69
Schleifen 66, 71
– Endekriterium 71
– for-Schleife 75
– Schleifendurchlauf 71
– Schleifenkopf 71
– Schleifenkörper 71
– Schleifenzähler 71
– while-Schleife 72
Schleifenzähler 71
Schlüssel
– Dictionary 39
– zulässige Typen 43
Schlüsselwortparameter 111
Scientific Python → SciPy
SciPy 377
Seiteneffekte 118
Seitenumbruch → Escape-Zeichen
Sekunden
– Additon zu Uhrzeiten 70
semantische Fehler 252
send-Methode 368
sep 296
Sequentielle Datentypen 29
Sequenz 29
set 53
set comprehension 358
setdefault → Dictionary

setgid 292
seteuid 292
setgid 292
setgroups 285
setpgid 292
setpgrp 292
setregid 292
setresgid 292
setresuid 292
setreuid 292
sets
– add 55
– clear 55
– copy 56
– difference 56
– difference_update 57
– discard 57
– intersection 58
– isdisjoint 58
– issubset 58
– issuperset 59
– Operationen auf sets 55
– pop 59
– remove 57
setsid 292
Setter 195
setter 213
setuid 292
setUp-Methode 262
Shell 267
– Bash 268
– Bourne 268
– C-Shell 268
– CLI 267
– GUI 267
– Herkunft und Bedeutung des Begriffes 267
Shell-Skripte ausführen 294
shelve-Modul 315
Shihram 79
shutil
– copy 304
– copy2 304
– copyfile 304
– copyfileobj 304
– copymode 304
– copystat 304
– copytree 304
– get_archive_formats 305
– get_unpack_formats 306
– ignore_patterns 306
– make_archive 306
– move 307

- register_archive_format 307
- register_unpack_format 307
- rmtree 308
- unpack_archive 308
- unregister_archive 308
- unregister_unpack_format 308
- shutil-Modul 303
- Sieb des Eratosthenes 358
 - Rekursive Berechnung 133
 - Rekursive Funktion mit Mengen-Abstraktion 359
- Simula 67 185
- Skalarprodukt 389
- Skriptsprache
 - Unterschied zu Programmiersprache 15
- sort 143
 - eigene Sortierfunktionen 144
 - reverse 144
 - Umkehrung der Sortierreihenfolge 144
- sorted 143
- spezialisierte Klasse → Unterklasse
- splICE 248
- split 162, 303
 - Folge von Trennzeichen 167
 - maxsplit 162
- splitdrive 303
- splittext 303
- splitlines 168
- Sprachfamilie 319
- sprintf 90
- Stack 135
 - Stapelspeicher 135
- Standardausnahmebehandlung 181
- Standardbibliothek 150
- Standardklassen als Basisklassen 245
- Stapelspeicher 135, 245
- stat 292
- stat_float_times 293
- staticmethod 210
- Statische Attribute 207
- Statische Methoden 210
- statische Typdeklaration 25
- Stelligkeit 120
- Stephen Cole Kleene 319
- StopIteration 362
- str 23, 197, 403
- strerror 293
- Strings
 - escape-Zeichen 31
 - raw 31
 - Suchen und Ersetzen 171
- String-Tests 174
 - Stringinterpolation 91
 - Stringmethoden
 - Alles in Großbuchstaben 172
 - Alles in Kleinbuchstaben 172
 - capitalize 172
 - center 173
 - count 169
 - find 169
 - index 169
 - isalnum 174
 - isalpha 174
 - isdigit 174
 - islower 174
 - isspace 174
 - istitle 174
 - isupper 174
 - ljust 173
 - lower 172
 - replace 171
 - rfind 169
 - rindex 169
 - rjust 173
 - rstrip 172
 - String-Tests 174
 - strip 172
 - title 172
 - upper 172
 - zfill 173
 - Stringmodulo 91
 - Strings 23
 - strip 172
 - Strukturierungselement 109
 - __sub__ 241, 464
 - Subklasse → Unterklasse
 - Suchen und Ersetzen 171
 - Suchen von Teilstrings 169
 - sum 72
 - Summe 24
 - Berechnung mit while-Schleife 72
 - SUnit 260
 - super 236
 - Superklasse → Basisklasse
 - supports_bytes_environ 270
 - symlink 293, 300
 - syntaktische Fehler 251
 - Syntax 251
 - Fehler 251
 - Syntaxfehler 251
 - sysconf 294
 - system 294
 - Systemprogrammierung 267

T

tar-Datei → Archivdatei erzeugen
TDD → test-driven development
tearDown-Methode 262
Tests 251
test first development 260
test-driven development 260
TestCase 262
– Methoden 262
– setUp-Methode 262
– tearDown-Methode 262
testCase 260
Testgesteuerte Entwicklung 260
Testgetriebene Entwicklung 259, 260
Testmethoden 262
Textklassifikation 421
Textverarbeitung 161
Theoretische Informatik 319
this
– statt self 192
tiefe Kopie 45, 107
times 294
Transiente Daten 313
True 23
__truediv__ 464
try 179
Tupel
– leere 140
– mit einem Element 141
– Packing 141
– Unpacking 141
type 27
type conversion 27
TypeError 255
– unhashable type 43
Typumwandlung 27
– explizit 27
– implizit 27
Typverletzung 26

U

Überladen 220, 224
Überlappungen 321
Überschreiben 220, 224
umask 294
Umgebungsvariablen 269
uname 294
unäre Operatoren 242
Unärsystem 247
Unary System → Unärsystem
Under Matching 322
unhashable type 43

Unicode-Zeichen → Escape-Zeichen
unittest 260 → Modultest
Unix 268
unlink 294
unpack_archive 308
Unpacking 141
unregister_archive_format 308
unregister_unpack_format 308
unsetenv 270
Unterklasse 219
Unterstrich
– Bedeutung in der interaktiven Shell 10
update → Dictionary
upper 172
urandom 294
utime 294

V

ValueError 180
Variable Anzahl von Parametern 120
Variablen 19
variadische Funktion 120
Variation 366
Vektoraddition 388
Vektorprodukt 394
Vektorsubtraktion 388
Vererbung 219
– Beispiel Angestelltenklasse, die von Person erbt 219
– Beispiel Kundenklasse, die von Person erbt 219
– Beispiel Lieferantenklasse, die von Person erbt 219
– Beispiel Personenklasse 219
Vererbungsbaum 226
Vergleichsoperatoren 67, 242
verschachtelte Dictionaries 43
Vertikaler Tabulator → Escape-Zeichen
Verzeichnis löschen 291
Verzweigungen 66
Vollständige Induktion 125

W

wait 295
wait3 295
waitpid 295
walk 295
Weizenkornaufgabe 79
while-Schleife 72
– optionaler else-Teil 74
Winkel zwischen Vektoren 390
write 271

X

`__xor__` 241

Y

`yield` 363

– im Vergleich zur `return`-Anweisung 363

Z

Zahlenratespiel 74

Zeichenauswahl 324

Zeichenkette 23

Zeilenumbruch → Escape-Zeichen

Zeitrechnung 70

`zeros_like()` 387

`zfill` 173

zip-Datei → Archivdatei erzeugen

zip-Funktion 47

Zufallspermutation 415

Zugriffsmethoden 195

Zustandsautomat 326

Zustandsmaschine 326

Zuweisung 20