

HANSER



Leseprobe

zu

Einführung in Python 3

von Bernd Klein

Print-ISBN: 978-3-446-46379-0

E-Book-ISBN: 978-3-446-46556-5

E-Pub-ISBN: 978-3-446-46467-4

Weitere Informationen und Bestellungen unter

<https://www.hanser-kundencenter.de/fachbuch/artikel/9783446463790>

sowie im Buchhandel

© Carl Hanser Verlag, München

Inhalt

Vorwort	XIX
Danksagung	XX
Teil I: Einleitung	1
1 Einleitung	3
1.1 Einfach und schnell zu lernen	3
1.2 Geschichte von Python	3
1.3 Zen von Python	4
1.4 Zielgruppe des Buches	5
1.5 Aufbau des Buches	6
1.6 Programmieren lernen „interaktiv“	7
1.7 Download der Beispiele und Hilfe	8
1.8 Anregungen und Kritik	8
Teil II: Grundlagen	9
2 Kommandos und Programme	11
2.1 Erste Schritte mit Python	11
2.1.1 Linux	11
2.1.2 Windows	12
2.1.3 macOS	13
2.2 Herkunft und Bedeutung des Begriffes interaktive Shell	13
2.2.1 Erste Schritte in der interaktiven Shell	14
2.3 Verlassen der Python-Shell	15
2.4 Benutzung von Variablen	15
2.5 Mehrzeilige Anweisungen in der interaktiven Shell	16
2.6 Programme schreiben oder schnell mal der Welt “Hallo” sagen	17

3	Bytecode und Maschinencode	21
3.1	Einführung.....	21
3.2	Unterschied zwischen Programmier- und Skriptsprachen	21
3.3	Interpreter- oder Compilersprache.....	21
4	Datentypen und Variablen	25
4.1	Einführung.....	25
4.2	Variablennamen	28
	4.2.1 Gültige Variablennamen	28
	4.2.2 Konventionen für Variablennamen	29
4.3	Datentypen	29
	4.3.1 Ganze Zahlen	29
	4.3.2 Fließkommazahlen.....	31
	4.3.3 Zeichenketten.....	31
	4.3.4 Boolesche Werte	31
	4.3.5 Komplexe Zahlen	32
	4.3.6 Operatoren	32
4.4	Statische und dynamische Typdeklaration	34
4.5	Typumwandlung	35
4.6	Datentyp ermitteln	36
5	Sequentielle Datentypen	39
5.1	Übersicht.....	39
	5.1.1 Zeichenketten oder Strings	40
	5.1.2 Listen.....	42
	5.1.3 Tupel	42
5.2	Indizierung von sequentiellen Datentypen	43
5.3	Teilbereichsoperator.....	44
5.4	Die len-Funktion	47
5.5	Aufgaben	47
6	Listen und Tupel im Detail.....	49
6.1	Virtueller Einkaufsbummel.....	49
6.2	Stapelspeicher/Stacks	51
6.3	Stapelverarbeitung in Python: pop und append.....	52
6.4	extend	52
6.5	Module importieren	53
6.6	,+'-Operator oder append	55
6.7	Entfernen eines Wertes.....	56

6.8	Prüfen, ob ein Element in Liste enthalten ist	57
6.9	Finden der Position eines Elementes	57
6.10	Einfügen von Elementen	57
6.11	Besonderheiten bei Tupel	58
6.11.1	Leere Tupel	58
6.11.2	1-Tupel	58
6.11.3	Mehrfachzuweisungen, Packing und Unpacking	59
6.12	Die veränderliche Unveränderliche	60
6.13	Aufgaben	60
7	Verzweigungen	63
7.1	Anweisungsblöcke und Einrückungen	63
7.2	Bedingte Anweisungen in Python	66
7.2.1	Einfachste if-Anweisung	66
7.2.2	if-Anweisung mit else-Zweig	67
7.2.3	elif-Zweige	67
7.3	Vergleichsoperatoren	68
7.4	Zusammengesetzte Bedingungen	68
7.5	Wahr oder falsch: Bedingungen in Verzweigungen	69
7.6	Aufgaben	70
8	Schleifen	71
8.1	Übersicht	71
8.2	while-Schleife	72
8.3	break und continue	73
8.4	Die Alternative im Erfolgsfall: else	74
8.5	Iterierbare Objekte (iterables)	76
8.6	For-Schleife	77
8.7	Aufgaben	80
9	Dictionaries	83
9.1	Definition und Benutzung	83
9.2	Fehlerfreie Zugriffe auf Dictionaries	86
9.3	Einfachere Definition von Dictionaries	88
9.4	Zulässige Typen für Schlüssel und Werte	88
9.5	Verschachtelte Dictionaries	89
9.6	Dictionaries in Listen wandeln	89
9.7	Weitere Methoden auf Dictionaries	90
9.8	Operatoren	92

9.9	Die zip-Funktion	93
9.10	Dictionaries aus Listen erzeugen	95
9.11	Aufgaben	95
10	Mengen	99
10.1	Übersicht	99
10.2	Mengen in Python	99
10.2.1	Sets erzeugen	100
10.2.2	Mengen von unveränderlichen Elementen	100
10.3	Frozensets	101
10.4	Operationen auf „set“-Objekten	101
10.4.1	add(element)	101
10.4.2	clear()	101
10.4.3	copy	102
10.4.4	difference()	102
10.4.5	difference_update()	102
10.4.6	discard(el)	103
10.4.7	remove(el)	103
10.4.8	intersection(s)	103
10.4.9	union(s)	104
10.4.10	isdisjoint()	104
10.4.11	issubset()	104
10.4.12	issuperset()	105
10.4.13	pop()	105
10.5	Erweiterte Zuweisungsoperatoren für Mengen	106
11	Eingaben	107
11.1	Eingabe mittels input	107
12	Dateien lesen und schreiben	109
12.1	Dateien	109
12.2	Text aus einer Datei lesen	109
12.3	Schreiben in eine Datei	111
12.4	In einem Rutsch lesen: readlines und read	111
12.5	with-Anweisung	112
12.6	Aufgaben	113

13	Formatierte Ausgabe und Strings formatieren	115
13.1	Wege, die Ausgabe zu formatieren	115
13.2	print-Funktion	116
13.3	Notwendigkeit	118
13.4	Formatierte Stringlitterale / f-Strings	118
13.5	Die String-Methode „format“	123
13.6	Stringmodulo-Operator oder Formatierung à la C	125
13.7	Benutzung von Dictionaries beim Aufruf der „format“-Methode	129
13.8	Benutzung von lokalen Variablen in „format“	130
13.9	Weitere String-Methoden zum Formatieren	131
14	Referenzen, flaches und tiefes Kopieren	133
14.1	Einführung	133
14.2	Swen und Sarah	133
14.3	Variablen sind Referenzen	134
14.4	Kopieren einer Liste	135
14.5	Flache Kopien	136
14.6	Kopieren mit deepcopy	138
14.7	Problemverhinderung	139
14.8	Deepcopy für Dictionaries	139
15	Funktionen	141
15.1	Allgemein	141
15.2	Funktionen	141
15.3	Docstring	143
15.4	Standardwerte für Funktionen	145
15.5	Schlüsselwortparameter	146
15.6	Funktionen ohne oder mit leerer return-Anweisung	146
15.7	Mehrere Rückgabewerte	147
15.8	Parameterübergabe im Detail	148
15.9	Effekte bei veränderlichen Objekten	150
15.10	Kommandozeilenparameter	151
15.11	Variable Anzahl von Parametern / Variadische Funktionen	152
15.12	* in Funktionsaufrufen	154
15.13	Beliebige Schlüsselwortparameter	155
15.14	Doppeltes Sternchen im Funktionsaufruf	155
15.15	Aufgaben	155

16	Wertebereich von Variablen	159
16.1	Einführung	159
16.2	Globale und lokale Variablen in Funktionen	159
16.3	nonlocal-Variablen	162
17	Rekursive Funktionen	167
17.1	Definition und Herkunft des Begriffs	167
17.2	Definition der Rekursion	168
17.3	Rekursive Funktionen in Python	168
17.4	Die Tücken der Rekursion	169
17.5	Fibonacci-Folge in Python	170
17.6	Aufgaben	174
18	Sortieren	177
18.1	Sortieren von Listen	177
18.1.1	„sort“ und „sorted“	177
18.1.2	Umkehrung der Sortierreihenfolge	178
18.1.3	Eigene Sortierfunktionen	178
18.2	Aufgaben	181
19	Modularisierung	183
19.1	Module	183
19.1.1	Namensräume von Modulen	184
19.1.2	Namensräume umbenennen	185
19.1.3	Modularten	185
19.1.4	Suchpfad für Module	186
19.1.5	Inhalt eines Moduls	187
19.1.6	Eigene Module	187
19.1.7	Dokumentation für eigene Module	188
19.2	Pakete	189
19.2.1	Einfaches Paket erzeugen	190
19.2.2	Komplexeres Paket	191
19.2.3	Komplettes Paket importieren	193
20	Alles über Strings	197
20.1	... fast alles	197
20.2	Aufspalten von Zeichenketten	198
20.2.1	split	199
20.2.2	rsplit	201

20.2.3	Folge von Trennzeichen	202
20.2.4	splitlines	203
20.2.5	partition und rpartition	204
20.3	Zusammenfügen von Stringlisten mit join	204
20.4	Suchen von Teilstrings	205
20.4.1	„in“ oder „not in“	205
20.4.2	s.find(substring[, start[, end]])	205
20.4.3	s.rfind(substring[, start[, end]])	206
20.4.4	s.index(substring[, start[, end]])	206
20.4.5	s.rindex(substring[, start[, end]])	206
20.4.6	s.count(substring[, start[, end]])	206
20.5	Suchen und Ersetzen	207
20.6	Nur noch Kleinbuchstaben oder Großbuchstaben	207
20.7	capitalize und title	208
20.8	Stripping Strings	208
20.9	Strings ausrichten	209
20.10	String-Tests	209
20.11	Aufgaben	211
21	Ausnahmebehandlung	215
21.1	Abfangen mehrerer Ausnahmen	217
21.2	except mit mehrfachen Ausnahmen	218
21.3	Die optionale else-Klausel	218
21.4	Fehlerinformationen über sys.exc_info	219
21.5	Exceptions generieren	220
21.6	Finalisierungsaktion	220
	Teil III: Objektorientierte Programmierung	223
22	Grundlegende Aspekte	225
22.1	Bibliotheksvergleich	225
22.2	Objekte und Instanzen einer Klasse	227
22.3	Kapselung von Daten und Methoden	228
22.4	Eine minimale Klasse in Python	228
22.5	Eigenschaften und Attribute	229
22.6	Methoden	232
22.7	Instanzvariablen	232
22.8	Die __init__-Methode	233

22.9	Destruktor	234
22.10	Datenkapselung, Datenabstraktion und Geheimnisprinzip	236
	22.10.1 Definitionen.....	236
	22.10.2 Zugriffsmethoden	238
	22.10.3 Properties.....	239
	22.10.4 Public-, Protected- und Private-Attribute.....	240
	22.10.5 Weitere Möglichkeiten der Properties	242
	22.10.6 Properties mit Dekoratoren	245
22.11	Stringausgaben mit str und repr	246
22.12	Klassenattribute	251
22.13	Statische Methoden	253
	22.13.1 Einleitendes Beispiel.....	254
	22.13.2 Getter und Setter für private Klassenattribute	255
22.14	Public-Attribute statt private Attribute	256
22.15	Magische Methoden und Operatorüberladung.....	258
	22.15.1 Einführung	258
	22.15.2 Übersicht magische Methoden.....	259
	22.15.3 Beispielklasse: Length	261
23	Bruchklasse	265
23.1	Brüche à la 1001 Nacht	265
23.2	Zurück in die Gegenwart.....	266
23.3	Rechenregeln	268
	23.3.1 Multiplikation von Brüchen	268
	23.3.2 Division von Brüchen.....	269
	23.3.3 Addition von Brüchen	270
	23.3.4 Subtraktion von Brüchen.....	270
	23.3.5 Vergleichsoperatoren	270
23.4	Integer plus Bruch	271
	23.4.1 Die Bruchklasse im Überblick	272
23.5	Fraction-Klasse.....	274
24	Aufrufbare Objekte	275
24.1	Einführung.....	275
	24.1.1 Die callable-Funktion.....	275
	24.1.2 Klassen statt Funktionen	276

25	Vererbung	279
25.1	Oberbegriffe und Oberklassen	279
25.2	Ein einfaches Beispiel	280
25.3	Überladen, Überschreiben und Polymorphie	281
25.4	Vererbung in Python.....	284
25.5	Klassenmethoden.....	287
25.6	Standardklassen als Basisklassen	289
26	Mehrfachvererbung	291
26.1	Einführung.....	291
26.2	Beispiel: KalenderUndUhr.....	292
26.3	Diamand-Problem oder „deadly diamond of death“	300
26.4	super und MRO	302
26.4.1	Ein einfaches Beispiel.....	302
26.4.2	Ein umfangreicheres Beispiel.....	305
27	Slots	307
27.1	Erzeugung von dynamischen Attributen verhindern	307
28	Dynamische Erzeugung von Klassen	309
28.1	Beziehung zwischen „class“ und „type“	309
29	Metaklassen	313
29.1	Motivation	313
29.2	Definition	318
29.3	Definition von Metaklassen in Python	318
29.4	Singletons mit Metaklassen erstellen	321
29.5	Beispiel: Methodenaufrufe zählen.....	322
29.5.1	Einführung	322
29.5.2	Vorbereitungen	322
29.5.3	Ein Dekorateur, um Funktionsaufrufe zu zählen	323
29.5.4	Die Metaklasse „Aufrufzähler“	324
30	Abstrakte Klassen	327
31	Aufgaben zur Objektorientierung	331

Teil IV: Funktionale Programmierung	335
32 Begriffsbestimmung	337
33 lambda, map, filter und reduce	339
33.1 lambda	339
33.2 map	341
33.3 Filtern von sequentiellen Datentypen mittels „filter“	344
33.4 reduce	344
33.5 Aufgaben	346
34 Listen-Abstraktion/List Comprehension	347
34.1 Die Alternative zu Lambda und Co.	347
34.2 Syntax	348
34.3 Weitere Beispiele	348
34.4 Die zugrunde liegende Idee	349
34.5 Anspruchsvolleres Beispiel	349
34.6 Mengen-Abstraktion	350
34.7 Rekursive Primzahlberechnung	351
34.8 Generatoren-Abstraktion	351
34.9 Aufgaben	352
35 Generatoren und Iteratoren	353
35.1 Einführung	353
35.2 Iteration in for-Schleifen	353
35.3 Generatoren	355
35.4 Endlos-Generatoren zähmen mit firstn und islice	359
35.5 Sinnvollere Beispiele	360
35.6 Beispiele aus der Kombinatorik	361
35.6.1 Permutationen	361
35.6.2 Variationen und Kombinationen	362
35.7 Generator-Ausdrücke	364
35.8 return-Anweisungen in Generatoren	365
35.9 send-Methode	366
35.10 Die close-Methode	370
35.11 Die throw-Methode	371
35.12 Dekoration von Generatoren	375
35.13 yield from	376
35.14 Aufgaben	378

36	Dekoratore	381
36.1	Einführung Dekoratore	381
36.1.1	Verschachtelte Funktionen	382
36.1.2	Funktionen als Parameter	384
36.1.3	Funktionen als Rückgabewert	385
36.1.4	Fabrikfunktionen	386
36.2	Ein einfacher Dekorateur	388
36.3	@-Syntax für Dekoratore	389
36.4	Anwendungsfälle für Dekoratore	392
36.4.1	Überprüfung von Argumenten durch Dekoratore	392
36.4.2	Funktionsaufrufe mit einem Dekorateur zählen	393
36.5	Dekoratore mit Parametern	395
36.6	Benutzung von Wraps aus functools	396
36.7	Eine Klasse als Dekorateur benutzen	398
36.8	Memoisation	399
36.8.1	Bedeutung und Herkunft des Begriffs	399
36.8.2	Memoisation mit Dekorateurfunktionen	399
36.8.3	Memoisation mit einer Klasse	400
36.8.4	Memoisation mit functools.lru_cache	401
Teil V: Weiterführende Themen		405
37	Tests und Fehler	407
37.1	Einführung	407
37.2	Modultests	409
37.3	Modultests unter Benutzung von <code>__name__</code>	410
37.4	doctest-Modul	412
37.5	Testgetriebene Entwicklung oder „Im Anfang war der Test“	415
37.6	unittest	417
37.7	Methoden der Klasse TestCase	419
37.8	Aufgaben	420
38	Daten konservieren	423
38.1	Persistente Speicherung	423
38.2	Pickle-Modul	424
38.2.1	Daten „einpökeln“ mit <code>pickle.dump</code>	424
38.2.2	<code>pickle.load</code>	425
38.3	Ein persistentes Dictionary mit <code>shelve</code>	425

39	Reguläre Ausdrücke	429
39.1	Ursprünge und Verbreitung	429
39.2	Stringvergleiche	429
39.3	Überlappungen und Teilstrings	431
39.4	Das re-Modul	431
39.5	Matching-Problem	432
39.6	Syntax der regulären Ausdrücke	434
39.6.1	Beliebiges Zeichen	434
39.7	Zeichenauswahl	435
39.8	Endliche Automaten	436
39.9	Anfang und Ende eines Strings	436
39.10	Vordefinierte Zeichenklassen	438
39.11	Optionale Teile	440
39.12	Quantoren	441
39.13	Gruppierungen und Rückwärtsreferenzen	443
39.13.1	Match-Objekte	443
39.14	Iteration über Matches mit finditer	446
39.15	Umfangreiche Übung	446
39.16	Alles finden mit findall	448
39.17	Alternativen	449
39.18	Compilierung von regulären Ausdrücken	450
39.19	Aufspalten eines Strings mit oder ohne regulären Ausdruck	450
39.19.1	split-Methode der String-Klasse	450
39.19.2	split-Methode des re-Moduls	452
39.19.3	Wörter filtern	454
39.20	Suchen und Ersetzen mit sub	455
39.21	Aufgaben	455
40	Typanmerkungen	459
40.1	Einführung	459
40.2	Einfaches Beispiel	460
40.3	Variablenanmerkungen	461
40.4	Listenbeispiele	462
40.5	Listen mit homogenem Typ	463
40.5.1	Version 3.6 bis 3.8	463
40.5.2	Python 3.9 und später	464

41	Systemprogrammierung	467
41.1	Einleitung	467
41.2	Häufig falsch verstanden: Shell	467
41.3	os-Modul	468
41.3.1	Vorbemerkungen	468
41.3.2	Umgebungsvariablen	469
41.3.3	Dateiverarbeitung auf niedrigerer Ebene	471
41.3.4	Weitere Funktionen im Überblick	476
41.3.5	os.path – Arbeiten mit Pfaden	490
41.4	shutil-Modul	498
41.5	glob-Modul	503
42	Forks	505
42.1	Fork	505
42.2	Fork in Python	505
Teil VI: Lösungen zu den Aufgaben		509
43	Lösungen zu den Aufgaben	511
43.1	Lösungen zu Kapitel 5 (Sequentielle Datentypen)	511
43.2	Lösungen zu Kapitel 6 (Listen und Tupel im Detail)	514
43.3	Lösungen zu Kapitel 7 (Verzweigungen)	516
43.4	Lösungen zu Kapitel 8 (Schleifen)	518
43.5	Lösungen zu Kapitel 9 (Dictionaries)	521
43.6	Lösungen zu Kapitel 12 (Dateien lesen und schreiben)	523
43.7	Lösungen zu Kapitel 15 (Funktionen)	524
43.8	Lösungen zu Kapitel 17 (Rekursive Funktionen)	529
43.9	Lösungen zu Kapitel 18 (Sortieren)	534
43.10	Lösungen zu Kapitel 20 (Alles über Strings ...)	537
43.11	Lösungen zu den Kapiteln 22 bis 31 (Aufgaben zur Objektorientierung)	540
43.12	Lösungen zu Kapitel 33 (lambda, map, filter und reduce)	554
43.13	Lösungen zu Kapitel 34 (Listen-Abstraktion/List Comprehension)	555
43.14	Lösungen zu Kapitel 35 (Generatoren und Iteratoren)	556
43.15	Lösungen zu Kapitel 37 (Tests und Fehler)	560
43.16	Lösungen zu Kapitel 39 (Reguläre Ausdrücke)	560
Stichwortverzeichnis		567



Vorwort

Ist es wirklich so, dass Vorworte – 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 seiner Bedeutung in Wissenschaft, Forschung und Wirtschaft in den letzten Jahren. Dies spiegelt sich auch in den Rankings von Programmiersprachen wider, die von verschiedenen Stellen durchgeführt werden. PYPL zählt wohl zu den bekanntesten und anerkanntesten unter diesen Webrankings. Python führt im Dezember 2020 mit 30,34 % den Index von PYPL an, gefolgt von Java mit nur 17,23 %.¹

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, der Psychologie und vielen anderen Bereichen. Weil bedeutende Firmen wie Google, Facebook, Instagram, Spotify, Quora, Netflix und Dropbox Python benutzen und unterstützen, wird Python auch kontinuierlich weiterentwickelt.

Dieses Buch erscheint nun in der vierten, völlig 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, Dekorateur, Systemprogrammierung, Threads, Forks, Ausnahmebehandlungen und Modultests. Auf über hundert Seiten wird anschaulich und mit zahlreichen Beispielen auf die vielfältigen Aspekte der Objektorientierung eingegangen.

Brigitte Bauer-Schiewek, Lektorin

¹ Im Vorwort zur 3. Auflage stand hier: 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.

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, wie bereits bei allen vorigen Auflagen auch, 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.

Mein besonderer Dank für die vierte Auflage gilt zum einen Herrn Tobias Habermann und zum anderen Herrn Dr. Konrad Wienands. Herr Habermann hat dafür gesorgt, dass alle Python-Beispiele dieser Auflage automatisch mittels „Pythontex“ getestet und ausgeführt werden. Herr Dr. Wienands ist mit großem Sachverstand in die Rolle eines potenziellen Anfängers geschlüpft und hat mir viele wertvolle Hinweise gegeben, wo es möglicherweise offene Fragen oder Probleme bei Neulingen geben könnte. Außerdem hat er auch kleine Unstimmigkeiten gefunden.

Vergessen darf ich auch nicht all diejenigen, die die Exemplare der vorigen Auflagen gekauft und damit erst den Erfolg des Buches ermöglicht hatten. Ebenso danke ich all denjenigen, die mich in E-Mails auf kleine Fehler oder Unstimmigkeiten hingewiesen haben, die ich aber hier leider nicht alle namentlich erwähnen kann.

Zuletzt danke ich auch ganz herzlich dem Hanser Verlag, der dieses Buch – nun auch in der vierten Auflage – ermöglicht. Vor allem danke ich Frau Brigitte Bauer-Schiewek und Frau Kristin Rothe, 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

4

Datentypen und Variablen

■ 4.1 Einführung

Sie glauben, weil Sie bereits in Sprachen wie C und C++ programmiert haben, dass Sie bereits genug über Datentypen und Variablen wissen? Vorsicht ist geboten! Sie wissen sicherlich viel, aber wahrscheinlich nicht genug, wenn es um Python geht. Deshalb lohnt es sich auf jeden Fall, hier weiterzulesen. Es gibt gravierende Unterschiede zwischen Python und anderen Programmiersprachen in der Art, wie Variablen behandelt werden. Vertraute Datentypen wie Ganzzahlen (Integer), Fließkommazahlen (floating point numbers) und Strings (Zeichenketten) sind zwar in Python vorhanden, aber auch hier gibt es wesentliche Unterschiede zu C/C++ und Java. Dieses Kapitel ist also sowohl für Anfänger als auch für fortgeschrittene Programmierende zu empfehlen.

Eine Variable ist ein Name bzw. ein Bezeichner, mit dem man auf ein Objekt zugreifen kann.¹ Ein Objekt steht zwar an einem bestimmten Speicherplatz, aber in Python spricht und denkt man nicht in physikalischen Speicherplätzen. Eine Variable ist nicht einem festen Objekt oder Speicherplatz zugeordnet. Während des Programmablaufs können einem Variablennamen neue beliebige Objekte zugewiesen werden, d.h. die Variable referenziert nach jeder Zuweisung in den meisten Fällen einen neuen Speicherort. Diese Objekte können beliebige Datentypen sein. Eine Variable referenziert also ein Objekt in Python.

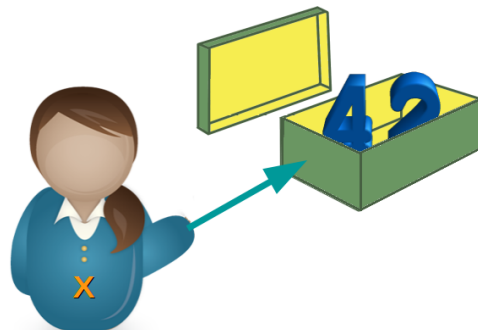


Bild 4.1 Variablen sind Referenzen auf Objekte.

¹ In den meisten Programmiersprachen ist es so, dass eine Variable einen festen Speicherplatz bezeichnet, in dem Werte eines bestimmten Datentyps abgelegt werden können. Während des Programmlaufs kann sich der Wert der Variable ändern, aber die Wertänderungen müssen vom gleichen Typ sein. Also kann man nicht in einer Variable zu einem bestimmten Zeitpunkt eine Integer-Zahl gespeichert haben und dann diesen Wert durch eine Fließkommazahl überschreiben. Ebenso ist der Speicherort der Variablen während des gesamten Laufs konstant, kann also nicht mehr geändert werden.

Im Folgenden kreieren wir eine Variable `x` in der interaktiven Python-Shell, indem wir ihr einfach den Wert 42 unter Verwendung eines Gleichheitszeichens zuweisen:

```
>>> x = 42
```

Man bezeichnet dies als Zuweisung oder auch als Definition einer Variablen. Genau genommen müsste man sagen, dass wir ein Integer-Objekt „42“ erzeugen und es mit dem Namen `x` referenzieren.

Anmerkung: Obige Anweisung darf man nicht als mathematisches Gleichheitszeichen sehen, sondern als „der Variablen `x` wird der Wert 42 zugewiesen“, d.h. der Inhalt von `x` ist nach der Zuweisung 42. Man kann die Anweisung also wie folgt „lesen“: „`x` soll sein 42“.

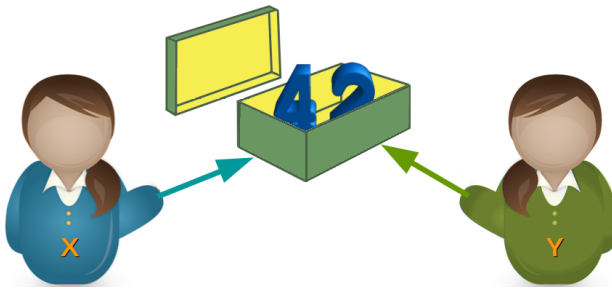
Wir können nun diese Variable zum Beispiel benutzen, indem wir ihren Wert mit der Funktion `print` ausgeben lassen:

```
>>> print("Wert von x: ", x)
Wert von x: 42
```

Wir können sie aber auch auf der rechten Seite einer Zuweisung verwenden:

```
>>> y = x
```

Beide Variablen zeigen nun auf das gleiche Objekt, d.h. das `int`-Objekt 42:



Woher wissen wir oder noch besser, wie kann man beweisen, dass beide Variablen auf das gleiche Objekt zeigen? Dazu bietet sich die `id`-Funktion an. Dies ist eine Funktion, die für ein Objekt die Identität eines Objektes zurückgibt. Erhält man für zwei Variablennamen die gleiche Identität, so weiß man, dass sie das gleiche Objekt referenzieren. Wir benutzen nun `id`, um die obige Behauptung zu beweisen:

```
>>> x = 42
>>> id(x)
9786176
>>> y = x
>>> id(y)
9786176
```

Der Ergebniswert von `id` interessiert in diesem Fall nicht. Wir möchten nur wissen, ob wir für beide Variablen die gleichen Werte erhalten:

```
>>> x = 42
>>> y = x
>>> id(x) == id(y)
True
```

Mit dem Ausdruck `id(x) == id(y)` können wir also prüfen, ob die beiden Variablen das gleiche Objekt referenzieren. Für einen solchen Test bietet Python auch den Operator `is`. Liefert `is` `True`, handelt es sich um das gleiche Objekt:

```
>>> pi1 = 3.141592653589793
>>> pi2 = 3.141592653589793
>>> pi1 is pi2 # the same as id(pi1) == id(pi2)
False

>>> pi1 = 3.141592653589793
>>> pi2 = pi1
>>> pi1 is pi2
True
```

Den Operator `is` darf man keinesfalls mit dem Gleichheitsoperator `==` verwechseln. Mit `==` kann man überprüfen, ob Objekte gleich sind, was aber nicht impliziert, dass die Objekte identisch sind.² Ein Beispiel aus dem täglichen Leben kann dies illustrieren. Wenn jemand sagt: „Das selbe Auto ist schon wieder vorbeigefahren.“, dann meint diese Person, dass es sich physikalisch um das gleiche Auto handelt, also gleiches Nummernschild beispielsweise. Wenn jemand sagt „Das ist das gleiche Auto, wie ich es habe!“, dann meint die Person nur, dass sie das gleiche Modell besitzt. Im folgenden Code sehen wir, dass `s1` und `s2` sowohl gleich sind als auch das identische Objekt referenzieren:

```
>>> s1 = "Der Unterschied zwischen 'das Gleiche' und 'dasselbe'"
>>> s2 = s1
>>> s1 == s2
True
>>> s1 is s2
True
```

Nun sehen wir, dass Gleichheit nicht immer Identität impliziert:

```
>>> s2 = "Der Unterschied zwischen 'das Gleiche' und 'dasselbe'"
>>> s1 is s2
False
>>> s1 == s2
True
```

Bisher stand meist nur eine einzelne Variable rechts vom Gleichheitszeichen. Auf der rechten Seite einer Zuweisung kann aber auch ein beliebiger, zu berechnender Ausdruck stehen, dessen Ergebnis dann der Variablen auf der linken Seite zugewiesen wird. Ein solcher Ausdruck kann auch andere Variablennamen enthalten. Diesen muss jedoch zuvor ein Wert zugewiesen worden sein. Das bedeutet, dass anschließend die Variable `y` auf das Ergebnis der Addition des alten `y`-Wertes und 36 zeigt.

```
>>> a = 3
>>> b = 1
>>> c = a + b
```

² In Gedanken kann man in den folgenden Beispielen auch den Begriff „Objekt“ durch „Wert“ ersetzen. Wir wollten von Anfang an die korrekten Begriffe, also „Objekt“, verwenden.

```
>>> c
4
```

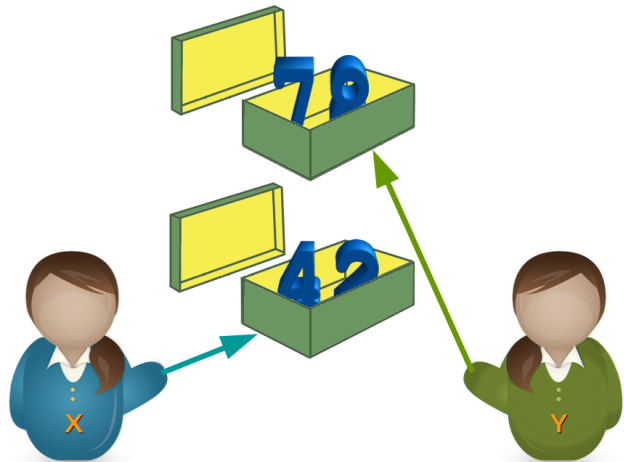
Wie man im nächsten Beispiel sehen kann, ist es möglich, eine Variable gleichzeitig auf der linken und rechten Seite einer Zuweisung zu benutzen.

```
>>> x = 42
>>> y = x
>>> y = y + 36
>>> y
78
```

Für die Schreibweise „ $y = y + 36$ “ verwendet man üblicherweise die sogenannte erweiterte Zuweisung „ $y += 36$ “.

Vom Ergebnis her gesehen sind die beiden Möglichkeiten identisch. Allerdings gibt es einen Implementierungsunterschied. Bei der erweiterten Zuweisung muss die Referenz der Variablen y nur einmal ausgewertet werden.³

Erweiterte Zuweisungen gibt es auch für die meisten anderen Operatoren wie „-“, „*“, „/“, „**“, „//“ (ganzzahlige Division) und „%“ (Modulo).



■ 4.2 Variablennamen

Wenn wir uns beim Programmieren Variablennamen ausdenken, muss man zweierlei beachten: Zum einen muss der Name den syntaktischen Anforderungen der Programmiersprache genügen, und zum anderen muss er den üblichen Gepflogenheiten und Konventionen entsprechen.

4.2.1 Gültige Variablennamen

Variablennamen müssen mit einem Buchstaben oder Unterstrich „_“ beginnen. Die folgenden Zeichen dürfen sich aus einer beliebigen Folge von Buchstaben, Ziffern und dem

³ Bei Datentypen wie Listen kann es zu extremen Laufzeitproblemen kommen, wenn man keine erweiterte Zuweisung verwendet. Darauf gehen wir jedoch erst später ein.

Unterstrich zusammensetzen. Variablennamen sind case sensitive.⁴ Dies bedeutet, dass Python zwischen Groß- und Kleinschreibung unterscheidet:

```
>>> person_42 = "Marvin"
>>> Person_42 = "Arthur"
>>> print(person_42)
Marvin
>>> print(Person_42)
Arthur
```

Zu den gültigen Buchstaben gehören aber nicht nur „lateinische“ Buchstaben, sondern auch alle anderen Unicode-Buchstaben, wie beispielsweise kyrillische, griechische und so weiter.

4.2.2 Konventionen für Variablennamen

Man bevorzugt in der Python-Community Kleinschreibung bei Variablennamen, z.B. `minimum` statt `Minimum`. Außerdem werden Variablennamen, die aus mehreren Wörtern bestehen, mittels Unterstrich (`_`) separiert, also beispielsweise `minimale_breite`. Variablennamen mit Binnenvorsalien – bei Programmierenden besser als „camel case“ oder „CamelCase“ bekannt – sind in der Python-Welt nicht beliebt: `MinimaleBreite` oder `minimaleBreite`.

Ansonsten zeichnet sich ein guter Programmierstil dadurch aus, dass man möglichst sprechende Variablennamen verwendet. Also beispielsweise `aktueller_kontostand` oder `maximale_beschleunigung` statt nichtssagender Namen wie `ak` oder `mb`.

■ 4.3 Datentypen

4.3.1 Ganze Zahlen

Ganze Zahlen werden in der Informatik üblicherweise als Integer bezeichnet. Es handelt sich dabei um die Zahlen ... -3, -2, -1, 0, 1, 2, 3, ...

Während Integer-Zahlen in den meisten Programmiersprachen durch einen Maximalwert begrenzt sind, sind sie in Python unbegrenzt, d.h. sie können beliebig groß bzw. beliebig klein werden. Die Variable `unsigned_long_max` bezeichnet in der folgenden interaktiven Python-Session den maximalen Integer-Wert in C für „unsigned long“. Ein Wert, der in C für diesen Datentyp nicht überschritten werden kann. Wir sehen, dass wir in Python diesen Wert sogar ohne Fehler beispielsweise quadrieren können.

```
>>> unsigned_long_max = 4294967295
>>> unsigned_long_max * unsigned_long_max
18446744065119617025
>>> x = 7876473829887890878787823666656543423341430089091000654
```

⁴ Für diesen Begriff gibt es keine deutsche Übersetzung.

```
>>> x * x
62038839992908819781348558921208782189916065115252466662224142028830j
└─ 910197615930689481485927796837531028427716
```

Bei Integer-Zahlen muss man beachten, dass sie nicht mit einer 0 beginnen dürfen, wenn es sich um eine Zahl im Dezimalsystem handeln soll:⁵

```
>>> 42
42
>>> 042
File "<stdin>", line 1
    042
      ^
```

SyntaxError: leading zeros in decimal integer literals are not permitted; use an 0o prefix for octal integers

Die führende Null wird benötigt, wenn man Binär-, Oktal- oder Hexadezimalzahlen schreiben möchte. Literale für Binärzahlen beginnen mit der Sequenz `0b` oder `0B` und werden dann von der eigentlichen Binärzahl gefolgt. Die Binärzahl `101010` schreibt man also wie folgt:

```
>>> x = 0b101010
>>> x
42
```

Man sieht, dass dies keine Auswirkung auf die Interndarstellung der Zahl hat.

Literale für Oktalzahlen beginnen mit der Sequenz `0o` oder `0O` und werden dann von der eigentlichen Oktalzahl gefolgt:

```
>>> x = 0o10
>>> x
8
```

Literale für Hexadezimalzahlen beginnen mit der Sequenz `0x` oder `0X` und werden dann von der eigentlichen Hexzahl gefolgt:

```
>>> x = 0x10
>>> x
16
>>> x = 0x1A
>>> x
26
```

Mit den Funktionen `hex`, `bin`, `oct` kann man eine Integer-Zahl in einen String wandeln, der der Stringdarstellung der Zahl in der entsprechenden Basis entspricht:⁶

```
>>> x = hex(19)
>>> x
'0x13'
>>> type(x)
```

⁵ Die Fehlermeldung im folgenden Beispiel gilt ab Python 3.8. Vorher wurde der Fehler "SyntaxError: invalid token" erhoben.

⁶ Strings, also Zeichenketten, haben wir noch nicht eingeführt. Diese Umwandlung haben wir hier nur der Vollständigkeit wegen eingeführt!

```
<class 'str'>
>>> bin(65)
'0b1000001'
>>> oct(65)
'0o101'
>>> oct(0b101101)
'0o55'
```

4.3.2 Fließkommazahlen

Fließkommazahlen⁷ entsprechen dem Datentyp `float` in Python. Es handelt sich dabei um Zahlen der Art 2.34, 27.87878 oder auch 3.15e2:

```
>>> x = 2.34
>>> y = 3.14e2
>>> y
314.0
```

Achtung: In deutschsprachigen Ländern werden die Nachkommastellen im täglichen Leben mit einem Komma eingeleitet. So schreibt man beispielsweise 1,99, wenn ein Artikel 1,99 € kostet. In Python und in anderen Programmiersprachen wird statt des Kommas immer ein Punkt, der sogenannte Dezimalpunkt, benutzt! Wie man aus obigem Beispiel ersehen kann, entspricht die Zahl „3.14e2“ dem Ausdruck $3.14 \cdot 10^2$.

4.3.3 Zeichenketten

Wir werden im nachfolgenden Kapitel weiter auf Strings eingehen.

4.3.4 Boolesche Werte

Bei den booleschen Werten handelt es sich um einen Datentyp, der nur die zwei Werte `True`⁸ (wahr) oder `False` (falsch) annehmen kann. Allerdings entspricht das im Prinzip den numerischen Werten 1 für `True` und 0 für `False`. Damit lässt sich mit booleschen Werten auch „numerisch“ rechnen, auch wenn das nicht immer gutem Programmierstil entspricht:

```
>>> x = True
>>> x * 4
4
```

Für zwei boolesche Variablen `x` und `y` gilt Folgendes:

⁷ Sie werden manchmal auch als Gleitpunktzahlen bezeichnet.

⁸ Ein beliebter Anfangsfehler besteht darin, dass man nicht auf die Großschreibung der beiden Werte achtet!

Tabelle 4.1 Logisches UND und ODER

Operator	Erklärung
not y	Negierung von y
x and y	Logisches UND von x und y
x or y	Logisches ODER von x und y

Beispielanwendungen:

```
>>> x = True
>>> not x
False
>>> y = False
>>> x and y
False
>>> x or y
True
>>> x and not y
True
```

4.3.5 Komplexe Zahlen

Python bietet einen Datentyp `complex` für komplexe Zahlen, den man bei den meisten Programmiersprachen vergeblich sucht. Allerdings würden ihn die meisten Anwender von Python wohl auch nicht vermissen. Wenn Sie komplexe Zahlen nicht kennen oder nichts mit ihnen zu tun haben, können Sie das Folgende gerne überspringen. Mathematisch gesehen erweitern die komplexen Zahlen die reellen Zahlen derart, dass die Gleichung $x^2 + 1 = 0$ lösbar wird. In der Mathematik werden komplexe Zahlen meist in der Form $a + b \cdot i$ dargestellt, wobei a und b reelle Zahlen sind und i die imaginäre Einheit ist. In Python benutzt man, der Konvention in der Elektrotechnik folgend, ein „j“ als imaginäre Einheit.

```
>>> x = 3 + 4j
>>> y = 2 - 4.5j
>>> x + y
(5-0.5j)
>>> x * y
(24-5.5j)
```

4.3.6 Operatoren

Eine Übersicht über die gängigsten Operatoren bietet die Tabelle 4.2. Zwei von diesen Operatoren werden wir nun im Detail vorstellen, nämlich „//“ und „%“:

„//“ bezeichnet die ganzzahlige Division, d.h. $11 // 4$ ergibt nicht den exakten Float-Wert 2.75, sondern den Integer-Wert 2, d.h. es wird immer auf die nächste ganzzahlige Inte-

Tabelle 4.2 Erklärung zu Operatoren

Operator	Erklärung
<code>x + y</code>	Summe von x und y
<code>x - y</code>	Differenz von x und y
<code>x * y</code>	Produkt von x und y
<code>x / y</code>	Quotient von x und y
<code>x // y</code>	Ganzzahlige Division
<code>x % y</code>	Modulo- oder Restdivision
<code>abs(x)</code>	Betrag von x
<code>x ** y</code>	Potenzieren, also x hoch y

ger „abgerundet“. Dies gilt allerdings nur für den Fall, dass beide Operanden ganze Zahlen (`int`) sind! Ist mindestens einer der Operanden eine `float`-Zahl, wird auch auf die nächste ganzzahlige Integer-Zahl abgerundet, aber dann das Ergebnis in `float` gewandelt:

```
>>> 8 // 3
2
>>> 11 // 3
3
>>> 12 // 3
4
>>> 12.0 // 3
4.0
```

Mit `%` lässt sich der Rest bei der Integerdivision bestimmen: So ergibt `11 % 4` den Wert 3 und `10 % 4` den Wert 2.

```
>>> 8 % 3
2
>>> 9 % 3
0
>>> 8.0 % 3
2.0
```

Es gilt folgender Zusammenhang zwischen der ganzzahligen Division und dem Modulo-Operator:

```
>>> x = 24
>>> y = 7
>>> x == (x // y) * y + (x % y)
True
```

■ 4.4 Statische und dynamische Typdeklaration

Wer Erfahrungen mit C, C++, Java oder ähnlichen Sprachen hat, hat dort gelernt, dass man einer Variablen einen Typ zuordnen muss, bevor man sie verwenden darf. Der Datentyp muss im Programm festgelegt werden – und zwar, bevor die Variable zum ersten Mal benutzt oder definiert wird. Dies sieht dann beispielsweise in einem C-Programm so aus:

```
int i, j;
float x;
x = i / 3.0 + 5.8;
```

Während des Programmlaufs können sich dann die Werte für *i*, *j* und *x* ändern, aber ihre Typen sind für die Dauer des Programmlaufs auf `int` im Falle von *i* und *j* und `float` für die Variable *x* festgelegt. Dies bezeichnet man als „statische Typdeklaration“, da bereits der Compiler den Typ festlegt und während des Programmablaufs keine Änderungen des Typs mehr vorgenommen werden können.

In Python sieht dies anders aus, wie wir bereits die ganze Zeit gesehen haben. Zunächst einmal bezeichnen Variablen in Python keinen bestimmten Typ, und deshalb benötigt man in Python keine Typdeklaration. Benötigt man im Programm beispielsweise eine Variable *i* mit dem Wert 42 und dem Typ Integer, so erreicht man dies einfach mit der Anweisung „`i = 42`“.

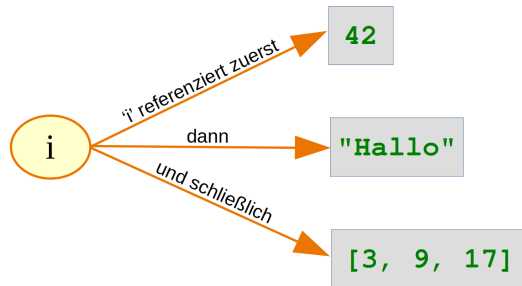


Bild 4.2 ‚i‘ während der Laufzeit

Damit hat man automatisch ein Objekt vom Typ Integer angelegt, welches dann von der Variablen *i* referenziert wird. Man kann dann im weiteren Verlauf des Programms der Variablen *i* auch andere Objekte mit anderen Datentypen zuweisen:

```
>>> i = 42
>>> i = "Hallo"
>>> i = [3, 9, 17]
```

Dennoch ordnet Python in jedem Fall einen speziellen Typ oder genauer gesagt eine spezielle Klasse der Variablen zu. Dieser Datentyp wird aber automatisch von Python erkannt. Diese automatische Typzuordnung, die auch während der Laufzeit erfolgen kann, bezeichnet man als „dynamische Typdeklaration“. Mit der Funktion `type` können wir uns den jeweiligen Typ ausgeben lassen:

```
>>> i = 42
>>> type(i)
<class 'int'>
>>> i = "Hallo"
>>> type(i)
<class 'str'>
```

```
>>> i = [3, 9, 17]
>>> type(i)
<class 'list'>
```

Während sich bei statischen Typdeklarationen, also bei Sprachen wie C und C++, nur der Wert, aber nicht der Typ einer Variablen während eines Laufs ändert, kann sich bei dynamischer Typdeklaration, also in Python, sowohl der Wert als auch der Typ einer Variablen ändern.

Aber Python achtet auf Typverletzungen zur Laufzeit eines Programms. Man spricht von einer Typverletzung⁹, wenn Datentypen beispielsweise aufgrund fehlender Zuweisungskompatibilität nicht regelgemäß verwendet werden. In Python kann es nur bei Ausdrücken zu Problemen kommen, da man ja einer Variablen einen beliebigen Typ zuordnen kann. Eine Typverletzung liegt beispielsweise vor, wenn man eine Variable vom Typ `int` zu einer Variablen vom Typ `str` addieren will. Es wird in diesem Fall ein `TypeError` generiert:

```
>>> x = "Ich bin ein String"
>>> y = 42
>>> z = x + y
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

Allerdings kann man Integer- und Float-Werte – auch wenn es sich um unterschiedliche Datentypen handelt – in einem Ausdruck mischen. Der Wert des Ausdrucks wird dann ein Float.

```
>>> x = 12
>>> y = 3.5
>>> z = x * y
>>> z
42.0
>>> type(x)
<class 'int'>
>>> type(y)
<class 'float'>
>>> type(z)
<class 'float'>
```

■ 4.5 Typumwandlung

In der Programmierung bezeichnet man die Umwandlung eines Datentyps in einen anderen als Typumwandlung.¹⁰ Man benötigt Typumwandlungen beispielsweise, wenn man Strings und numerische Werte mit dem Stringoperator „+“ konkatenieren möchte.

⁹ engl. type conflict

¹⁰ engl. type conversion oder type cast

```
>>> first_name = "Henry"
>>> last_name = "Miller"
>>> age = 20
>>> print(first_name + " " + last_name + ": " + str(age))
Henry Miller: 20
```

In dem Beispiel haben wir den Wert von `age` explizit mit der Funktion `str` in einen String gewandelt. Man bezeichnet dies als explizite Typumwandlung. Hätten wir in obigem Beispiel nicht den Integer-Wert `age` in einen String gewandelt, hätte Python einen `TypeError` generiert:

```
>>> print(first_name + " " + last_name + ": " + age)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

Der Text der Fehlermeldung besagt, dass Python keine implizite Typumwandlung von `int` nach `str` vornehmen kann. Prinzipiell unterstützt Python keine impliziten Typumwandlungen, wie sie in Perl oder PHP möglich sind. Dennoch gibt es Ausnahmen so wie unser Beispiel, in dem wir Integer- und Float-Werte in einem Ausdruck gemischt hatten. Dort wurde der Integer-Wert implizit in einen Float-Wert gewandelt.

■ 4.6 Datentyp ermitteln

In vielen Anwendungen muss man für ein Objekt bestimmen, um welchen Typ bzw. um welche Klasse es sich handelt. Dafür bietet sich meistens die eingebaute Funktion `type` an. Man übergibt ihr als Argument einen Variablennamen und erhält den Typ des Objekts zurück, auf das die Variable zeigt:

```
>>> l = [3, 5, 4]
>>> type(l)
<class 'list'>
>>> x = 4
>>> type(x)
<class 'int'>
>>> x = 4.5
>>> type(x)
<class 'float'>
>>> x = "Ein String"
>>> type(x)
<class 'str'>
```

Als Alternative zur Funktion `type` gibt es noch die eingebaute Funktion `isinstance`, die einen Wahrheitswert „True“ oder „False“ zurückgibt.

```
isinstance(object, ct)
```

`object` ist das Objekt, das geprüft werden soll, und `ct` entspricht der Klasse oder dem Typ, auf den geprüft werden soll. Im folgenden Beispiel prüfen wir, ob es sich bei dem Objekt `s` um ein String handelt:

```
>>> s = "Ich bin ein String"
>>> isinstance(s, str)
True
```

In Programmen kommt es häufig vor, dass wir für ein Objekt wissen wollen, ob es sich um einen von vielen Typen handelt. Also zum Beispiel die Frage, ob eine Variable ein Integer oder ein Float ist. Dies kann man mit der Verknüpfung `or` lösen:

```
>>> x = 4
>>> isinstance(x, int) or isinstance(x, float)
True
>>> x = 4.8
>>> isinstance(x, int) or isinstance(x, float)
True
```

Allerdings bietet `isinstance` hierfür eine bequemere Möglichkeit. Statt eines Typs gibt man als zweites Argument ein Tupel von Typen an:

```
>>> x = 4.8
>>> isinstance(x, (int, float))
True
>>> x = (89, 123, 898)
>>> isinstance(x, (list, tuple))
True
>>> isinstance(x, (int, float))
False
```

Möchte man in einem Programm auf einen bestimmten Typ prüfen, so kann man obiges Vorgehen wählen, aber häufig lässt sich dies eleganter mit Ausnahmehandlungen lösen, die wir in Kapitel [Ausnahmebehandlung](#), Seite 215 behandeln.



Stichwortverzeichnis

- ^ bei regulären Ausdrücken 436
- . Match eines beliebigen Zeichens 434
- \$ bei regulären Ausdrücken 436
- % Modulo-Operator 28
- 16-Bit-Unicode-Zeichen → Escape-Zeichen
- 32-Bit-Unicode-Zeichen → Escape-Zeichen

- ABC 3
- Abfragemethode → Getter
- Abgeleitete Klasse → Unterklasse
- abort 476
- __abs__ 260
- abspath 490
- Abstrakte Klassen 327
- Abstrakte Methoden 327
- access 477
- __add__ 258, 259, 544
 - Beispiel 261
- all 76, 392
- Allgemeine Klasse → Basisklasse
- Amoeba 3
- Anagramm
 - als Beispiel einer Permutation 361
- and 68, 259
- Änderungsmethode → Setter
- Anführungszeichen → Escape-Zeichen
- Angestelltenklasse 280 → Personenklasse
- Ankerzeichen 438
- Anweisungsblöcke 63
- any 76
- Anzahl der Elemente einer Liste 47
- Anzahl der Elemente eines Tupels 47
- Anzahl der Zeichen eines Strings 47
- append 51, 55
- Archivdatei erzeugen 500
- Arität 152
- ASCII-Zeichen hexadezimal →
 - Escape-Zeichen
- ASCII-Zeichen oktal → Escape-Zeichen

- Asimovsche Gesetze 252
- assertAlmostEqual 421
- assertCountEqual 421
- AssertEqual 419
- assertEqual 421
- assertFalse 421
- assertGreater 421
- assertGreaterEqual 421
- assertIn 422
- assertIs 422
- assertIsInstance 422
- assertIsNone 422
- assertIsNot 422
- assertItemsEqual 422
- assertLess 422
- assertLessEqual 422
- assertListEqual 422
- assertNotRegexpMatches 422
- assertTrue 421
- assertTupleEqual 422
- AssertionError 418
- assoziative Arrays 83
- assoziatives Feld 83
- atime 492
- Attribute 229
 - dynamische Erzeugung 231
 - private 240
 - protected 240
 - public 240
- AttributeError 286
- aufrufbare Objekte 275
- Aufspalten von Zeichenketten 198
- Augmented Assignment → erweiterte Zuweisungen
- augmented assignment 55
- Ausdruck 27
- Ausgabe
 - formatieren 115

- in Datei umleiten 117
- in Fehlerkanal umleiten 117
- Auskellern 51
- Ausnahme
 - StopIteration 354
- Ausnahmebehandlung 215
 - Abfangen mehrerer Ausnahmen 217
 - except 215
 - Exception 215
 - finally 220
 - try 215
- Automatentheorie 429

- basename 490
- Bash 468
- Bash-Befehle ausführen 488
- Basisklasse 279
- bedingte Anweisungen 66
- Bibliothek 184
- Bierdeckelarithmetik 332
- Bierdeckelnotation 332
- binäre Operatoren 259
- Binärzahlen 30
- Blöcke 63
- Boehm, Barry W. 407
- Boolesche Werte 31
- Bourne-Shell 468
- break 73, 74, 78
- Bruch
 - kürzen 267
 - Kürzungszahl 267
 - Repräsentierung in Python 266
 - vollständig gekürzte Form 267
- Bruchklasse 265
- Bruchrechnen 266
- bztar-Datei → Archivdatei erzeugen

- C3 superclass linearization 302
- __call__ 276, 398
- callable 275, 398
- Call-by-Reference 149
- Call-by-Value 149
- Caret-Zeichen 435
- Cast 35
- cast-Operator 107
- Casting 35
- Catullus 110
- CaveInt 544→ Höhlenmenschen-Arithmetik
- center 209
- chdir 477
- chmod 477
- Chomsky, Noam 167
- Chomsky-Grammatik 167
- chown 477
- chroot 478
- Church, Alonzo 337
- classmethod 288
- clear → Dictionary
- CLI 467
- close 474
- close-Methode
 - Generator 370
- Closure 400
- Codeblock 71, 459
- commonprefix 491
- complex 32, 260
- continue 73, 78
- copy 498→ Dictionary
- copy2 498
- copy-Modul 138
- copyfile 498
- copyfileobj 498
- copymode 499
- copystat 499
- copytree 499
- cos 53
- count 57, 205
- ctime 492

- data hiding → Geheimnisprinzip
- Datei 109
 - lesen 109
 - öffnen 109
 - schreiben 111
 - zum Schreiben öffnen 111
- Dateibaum rekursiv durchwandern 489
- Dateibearbeitung mit os 471
- Dateideskriptor 474, 475
- Dateigröße 493
- Dateinamen im Verzeichnis 481
- Daten konservieren 423
- Daten sichern mit Pickle 423
- Datenabstraktion 236
- Datenkapselung 228, 236
- Datenpersistenz 423
- Datentypen 25
- Deadly Diamond of Death 300
- Decorator → Dekorateur
- deepcopy 138
- def 142
- Definition einer Variablen 26
- Dekorateur 381
 - Anwendungsfälle 392
 - classmethod 289

- Einführung 381
- Funktionsdekorateur 381
- Klassendekorateur 381
- Memoisation 399
- mit Parametern 395
- zur Überprüfung von Funktionsargumenten 392
- Dekoration
 - mit Klasse 398
- `__del__` 253
- deleter 244
- Destruktor 234
- Dezimalpunkt 31
- Dezimalsystem 332, 544
- Diamond-Problem 300
- dict → Dictionary
- Dictionary 83, 91
 - aus Listen erzeugen 95
 - clear 90
 - copy 90
 - flache Kopie 90, 139
 - get 87
 - in 86
 - Items 83
 - items 89
 - Items ändern 84
 - Items hinzufügen 84
 - keys 89
 - leeres 84
 - nested Dictionaries 89
 - pop 91
 - popitem 91
 - Schlüssel 83
 - setdefault 92
 - tiefe Kopie 90, 139
 - update 92
 - values 89
 - verschachtelte Dictionaries 89
 - view 89
- Differenz 32
- dirname 491
- Division, ganzzahlige 28
- `__doc__` 143
 - bei Dekorateuren 396
- Docstring 143
 - property 244
- doctest 412
- dup 476
- Dynamische Attribute 251
- Dynamische Erzeugung von Klassen 309
- dynamische Typdeklaration 34, 35
- Eigenschaften 229
- Eingabe 107
- Eingabeaufforderung
 - robust 216
- Eingabeprompt 11, 107
- Einkellern 51
- Einrückungen 63
- Einschubmethode → Hook-Methode
- Elternklasse → Basisklasse
- endliche Automaten 429, 436
- Endlos-Generator 359
- `__eq__` 250, 260
- erben → Vererbung
- errare humanum est 407
- erweiterte Zuweisungen 28, 260
- Escape-Zeichen 41
 - 16-Bit-Unicode-Zeichen 41
 - 32-Bit-Unicode-Zeichen 41
 - Anführungszeichen 41
 - ASCII-Zeichen hexadezimal 41
 - ASCII-Zeichen oktal 41
 - Hochkomma 41
 - Horizontaler Tabulator 41
 - Rückschritt 41
 - Seitenumbruch 41
 - Unicode-Zeichen 41
 - Vertikaler Tabulator 41
 - Zeilenumbruch 41
- Euklidischer Algorithmus 267
- eval 108
- except 215
- exists 491
- expandvars 492
- explizite Typumwandlung 36
- extend 52
- extsep 478
- Fabrikfunktion 386
- f-string 118
 - Ausrichtungsoptionen 121
 - linksbündige Ausgabe 121
 - rechtsbündige Ausgabe 121
 - selbst-dokumentierende Ausdrücke 122
 - sign-Optionen 122
 - Zentrierte Ausgabe 121
- Fakultätsfunktion 167
 - iterative Lösung 169
 - rekursive Implementierung 168
- False 31
- Fehler
 - Semantik 408
- Fehlerarten 407

- fib → Fibonacci-Modul
- fiblist → Fibonacci-Modul
- Fibonacci 170
- Fibonacci-Folge → Fibonacci-Zahlen
- Fibonacci-Modul 409
 - fib-Funktion 409
 - fiblist-Funktion 409
- Fibonacci-Zahlen 147, 378
 - Begrenzung 147
 - Fibonacci-ähnliche Folgen 278
 - Fibonacci-Folge 278
 - formale mathematische Definition 170
 - Funktion 147
 - Generator 378, 556
 - Modul 188
 - Modul zur Berechnung 409
 - rekursive Berechnung 176
 - rekursive Berechnung mit Dekoratoren 399
 - rekursive Funktion 169
- filter 339
- finally 220
- find 57, 205
- Finite State Machine 436
- firstn-Generator 359
- flache Kopie 90, 139
- flatten 534
- __float__ 260
- __floordiv__ 259
- flüchtige Daten 423
- Flussdiagramm 64
- for-Schleife 77
 - StopIteration 354
- for-Schleifen
 - optionaler else-Teil 77
 - Unterschied zur while-Schleife 78
 - Vergleich zu C 77
- fork 478, 505
- Forking 505
- forkpty 478
- formale Sprachen 429
- format 115, 123
 - Positionsparameter 123
 - Schlüsselwertparameter 123
- format-Methode 126
- Formatierte Ausgabe 115
- Formatierte Stringlitterale 115, 118
- Formatierung 115
 - f-String 115
 - format-Methode 115
 - formatierte Stringlitterale 115
 - Stringmodula-Operator 125
 - Stringmodulo 115
- Formatstring 126
- Fraction-Klasse 274
- fractions-Modul 274
- fromkeys → Dictionary
- Frosch
 - Aufgabe mit Summenbildung 81
 - Straßenüberquerung 81
- frozensets 101
- fully qualified 184
- functools 344
- funktionale Programmierung 337
- Funktionen 141
 - als Parameter 384
 - Attribute 231
 - Fabrikfunktionen 386
 - Kopf 142
 - Körper 142
 - Parameterliste 142
 - Parameterübergabe 148
 - Rückgabewert None 142
 - Rückgabewerte 146
 - Rumpf 142
 - statische Variablen 231
 - überladen 281
 - Überschreiben 281
 - variadische 152
 - verschachtelt 162
 - verschachtelte Funktionen 382
 - Zählen der Aufrufe 231
- Funktionsabschluss 400
- Funktionsdekorateur → Dekorateur

- Ganze Zahlen 29
- ganzzahlige Division 28, 32
- Gaußsche Summenformel 72
- __ge__ 260
- Geheimnisprinzip 236
- Generator-Ausdrücke 364
- Generator-Comprehension →
 - Generatoren-Abstraktion
- Generatoren
 - Allgemein 353
 - close-Methode 370
 - CLU 353
 - Endlosschleife in 358
 - firstn 359
 - frange 379
 - Icon 353
 - islice zur Ausgabe von unendlichen
 - Generatoren 359
 - pair_sum 380
 - pendulum 379

- Permutationen 361
- return 365
- round_robin 379
- send 366
- throw-Methode 371
- yield 355
- zur Erzeugung von Variationen 362
- Generatoren-Abstraktion 351
- get → Dictionary
- get_archive_formats 500
- getatime 492
- getattr 231
- getcwd 479, 497
- getcwdb 479
- getegid 479
- getenv 469
- getenvb 469
- geteuid 479
- get_exec_path 478
- getgid 479
- getgroups 479
- getloadavg 479
- getlogin 479
- getmtime 493
- getpgid 479
- getpgrp 479
- getpid 480
- getppid 480
- getresgid 480
- getresuid 480
- getsize → os.path
- Getter 238
- getuid 480
- get_unpack_formats 500
- ggT 267
- glob-Modul 503
- größter gemeinsamer Teiler 267
- Gruppierungen 443
- __gt__ 260
- GUI 467
- gztar-Datei → Archivdatei erzeugen

- hasattr 231
- Hash 83
- __hex__ 260
- Hexadezimalzahlen 30
- Hochkomma → Escape-Zeichen
- Höhlenmenschen-Arithmetik 332, 544
- Hook-Methode 419
- Hooks → Hook-Methode
- Horizontaler Tabulator → Escape-Zeichen

- __iadd__ 260
- __iand__ 260
- id 26, 134
- Identitätsfunktion 26, 134
- __idiv__ 260
- if-Anweisung 66
- __ifloordiv__ 260
- ignore_patterns 500
- __ilshift__ 260
- immutable 44
- __imod__ 260
- implizite Typumwandlung 36
- import 53
- import-Anweisung 184
- __imul__ 260
- index 57, 205
- Inheritance → Vererbung
- __init__ 258
- __init__-Methode 233
- __init__.py 190
- input 86, 107
- insert 57
- Instanz 227, 229
- Instanzattribute 229, 231, 251
- Instanzvariablen 232
- int 29, 260
- Integer 29
- interaktive Shell
 - Befehlsstack 15
 - Starten unter Linux 11
 - Starten unter Windows 12
 - Unterstrich 15
- __invert__ 260
- __ior__ 260
- __ipow__ 260
- __irshift__ 260
- isabs 494
- isalnum 209
- isalpha 209
- isdigit 209
- isdir 494
- isfile 494
- isinstance 36
- islice 359
 - als Ersatz für firstn 359
- islink 495
- islower 209
- ismount 495
- is_palindrome 157
- is_prime 392
- isspace 209
- istitle 209

- `__isub__` 260
- `isupper` 209
- `items` → Dictionary
- `itemgetter` 179
- `__iter__` 354
- Iteration
 - for-Schleife 353
 - über Dictionary 79
 - über String 79
- Iteratoren 353
 - for-Schleife 353
- iterierbare Objekte 76
- `itertools` 359
 - `islice` 359
- `__ixor__` 260

- `join` 204, 495
- Junit 417

- Kalender-Klasse 292
- KalenderUndUhr-Klasse 292
- kanonische Stringdarstellung 247, 334
- kanonischer Pfad 496
- Keller 51
- Kellerspeicher 51, 289
- `key`
 - Dictionary 83
- `keys` → Dictionary
- `KeyError` 86
- `kill` 480
- `killpg` 480
- Kindklasse → Unterklasse
- Klasse
 - abstrakte 327
 - Instanz 229
 - Kopf 228
 - Körper 228
 - minimale Klasse in Python 228
- Klassenattribute 230, 251
- Klassendekorateur 398 → Dekorateur
- Klassenmethoden 255, 287, 288
- Klassenobjekt 230
- Kombination 362
- Kommandozeilenparameter 151
- Kompilierung von regulären Ausdrücken 450
- Komplexe Zahlen 32
- Komponententest 409
- Konstruktor 233
- Kontrollstrukturen 66
- Kopieren 133
 - von Unterlisten 137
 - von verschachtelten Listen 133
- Kundenklasse → Personenklasse
- Kürzen 267

- `lambda` 339
- Lambda-Kalkül 337
- Länge von `dictionary`s 92
- Lazy Evaluation 90
- `__le__` 260
- `len` 92
- `len`-Funktion 46, 47
- Leonardo von Pisa → Fibonacci-Zahlen
- Lesbia 110
- `lexists` 495
- Liber Abaci 170
- Lieferant → Personenklasse
- `linesep` 480
- `link` 480
- Linux 468
 - Python unter 11
- `list`
 - `append` 51
 - `augmented assignment` 55
 - `count` 57
 - `extend` 52
 - `find` 57
 - `flatten` 534
 - `index` 57
 - `insert` 57
 - Klasse 227
 - `Packing` 59
 - `pop` 51
 - `remove` 56
 - `sort` 177
 - `Unpacking` 59
- List Comprehension 347
 - Kreuzprodukt 349
 - pythagoreisches Tripel 348
 - Sieb des Eratosthenes 349
 - Syntax 348
 - Vergleich mit konventionellem Code 348
 - Wandlung Celsius in Fahrenheit 348
 - Weitere Beispiele 348
 - zugrundeliegende Idee 349
- `listdir` 481
- `list_iterator` 354
- Listen-Abstraktion 347
 - Kreuzprodukt 349
 - pythagoreisches Tripel 348
 - Sieb des Eratosthenes 349
 - Syntax 348
 - Vergleich mit konventionellem Code 348
 - Wandlung Celsius in Fahrenheit 348

- Weitere Beispiele 348
- zugrundeliegende Idee 349
- ljust 209
- Löffelsprache 155
- __long__ 260
- lower 207
- lseek 475
- __lshift__ 259
- lstat 481
- __lt__ 260

- Magische Methoden 258
- __add__ 258
- binäre Operatoren 259
- Erweiterte Zuweisungen 260
- __floordiv__ 259
- __init__ 233, 258
- __mod__ 259
- __mul__ 259
- __new__ 258
- __pow__ 259
- __repr__ 246, 258
- __str__ 246
- __sub__ 259
- __truediv__ 259
- unäre Operatoren 260
- Vergleichsoperatoren 260
- Magische Operatoren
- Beispielklasse Length 261
- __main__ 229, 409
- major 481
- make_archive 500
- makedev 481
- makedirs 482
- Manager-Funktion 317
- map 339, 341
- Mapping 83
- Match eines beliebigen Zeichens 434
- Match-Objekte 443
- Matching-Problem 432
- Over Matching 432
- Under Matching 432
- math-Modul 53, 184
- maxsplit
- split 198
- Mehrfachschnittstellenvererbung 291
- Mehrfachvererbung 291
- Beispiel KalenderUndUhr 292
- Mehrfachzuweisung 43
- Memoisation 173, 399
- mit functools.lru_cache 401
- mit Klasse 400

- Memoization → Memoisation
- Memoize 400
- memoize-Funktion 400
- Mengen 99
- add 101
- clear 101
- copy 102
- difference 102
- difference_update 102
- isdisjoint 104
- issubset 104
- issuperset 105
- Obermenge 105
- pop 105
- remove 103
- Teilmenge 104
- Mengen-Abstraktion 350
- Mengenlehre 99
- Metaklassen 313
- Method Resolution Order 302, 305
- Methoden 228, 232
- abstrakte 327
- __init__ 233
- Overloading → Überladen
- Overwriting → Überschreiben
- Überladen 285
- überlagern 281
- Überschreiben 285
- Unterschiede zur Funktion 232
- Michie, Donald 399
- minor 481
- mkdir 482
- mkfifo 483
- __mod__ 259
- modulares Design 183
- modulare Programmierung 183
- Modularisierung 183
- Module 183
- Arten 185
- dir() 187
- Dokumentation 188
- dynamisch geladene C-Module 185
- eigene Module schreiben 187
- importieren 53
- Inhalt 187
- lokal 183
- math 184
- Namensraum 184
- re 431
- Suchpfad 186
- __module__
- bei Dekoratoren 396

- Modulo-Division 32
- Modulo-Operator 28
- Modultest 409
 - unter Benutzung von `__name__` 409
- Monty Python 4
- move 501
- MRO 302, 305
- mtime 492
- `__mul__` 259, 544
- mutable 44
- mypy 460

- `__name__` 229, 409
 - bei Dekorateuren 396
- Namensraum
 - umbenennen 185
- `__ne__` 260
- Nebeneffekte 150
- `__neg__` 260
- Nested Dictionaries 89
- `__new__` 258
- next 354
- nice 483
- normcase 496
- normpath 496
- not 68
- NotImplemented 271

- Oberklasse 279
- object 279
- Objekt 227
- Objektorientierte Programmiersprache 225
- Objektorientierte Programmierung 225
- `__oct__` 260
- öffentliche Attribute 240
- Oktalzahlen 30
- OOP 225
- open 472
- open() 109
- openpty 474
- operator-Modul 179
 - itemgetter 179
- Operatoren überladen
 - binäre Operatoren 259
 - erweiterte Zuweisungen 260
 - unäre Operatoren 260
 - Vergleichsoperatoren 260
- Operatorüberladung 258
- or 68, 259
- os
 - abort 476
 - access 477
 - chdir 477
 - chmod 477
 - chown 477
 - chroot 478
 - close 474
 - dup 476
 - extsep 478
 - fork 478
 - forkpty 478
 - getcwd 479
 - getcwdb 479
 - getegid 479
 - getenv 469
 - getenvb 469
 - geteuid 479
 - get_exec_path 478
 - getgid 479
 - getgroups 479
 - getloadavg 479
 - getlogin 479
 - getpgid 479
 - getpgrp 479
 - getpid 480
 - getppid 480
 - getresgid 480
 - getresuid 480
 - getuid 480
 - kill 480
 - killpg 480
 - linesep 480
 - link 480
 - listdir 481
 - lseek 475
 - lstat 481
 - major 481
 - makedev 481
 - makedirs 482, 485
 - minor 481
 - mkdir 482
 - mkfifo 483
 - nice 483
 - open 472
 - openpty 474
 - popen 483
 - putenv 470
 - read 474
 - readlink 485
 - remove 485
 - removedirs 485
 - rename 486
 - renames 486
 - rmdir 486

- sep 491
- setegid 486
- seteuid 486
- setgid 486
- setgroups 479
- setpgid 487
- setpgrp 487
- setregid 487
- setresgid 487
- setresuid 487
- setreuid 487
- setsid 487
- setuid 486
- stat 487
- stat_float_times 487
- strerror 488
- supports_bytes_environ 470
- symlink 488
- sysconf 488
- system 488
- times 488
- umask 489
- uname 489
- unlink 489
- unsetenv 470
- urandom 489
- utime 489
- wait 489
- wait3 489
- waitpid 489
- walk 489
- write 471
- os-Modul 468, 476
- os.path 490
- os.python
 - abspath 490
 - basename 490
 - commonprefix 491
 - dirname 491
 - exists 491
 - expandvars 492
 - getatime 492
 - getmtime 493
 - getsize 493
 - isabs 494
 - isdir 494
 - isfile 494
 - islink 495
 - ismount 495
 - join 495
 - lexists 495
 - normcase 496
 - normpath 496
 - realpath 496
 - relpath 497
 - samefile 497
 - split 497
 - splitdrive 497
 - splitext 498
- Over Matching 432
- Overloading 281 → Überladen
- Overwriting 281 → Überschreiben

- Packing 59
- Paket 190
- Paketkonzept 190
- Palindrome 157
- Parameter 143
 - beliebige Anzahl 152
 - Defaultwert 143
 - optional 143
 - Schlüsselwort 146
 - Standardwert 143
- Parameterliste 142
- Parameterübergabe 148
- partition 204
- Pascalsches Dreieck → Fibonacci-Zahlen
- pass 146
- peek 51
- peep 51
- Permutationen 361
 - mit Wiederholungen 361
 - ohne Wiederholungen 361
- Permutations-Generator 361
- Persistente Daten 423
- Personenklasse 280 → Vererbung
- Pfadname
 - absolut 494
 - relativ 494
- pi 53
- pickle-Modul 423
- Platzhalter 126
- Polymorphismus 283
- Polynom-Fabrikfunktion 386
- Polynomklasse 334
- pop 51, 289 → Dictionary
- popen 483
- popitem → Dictionary
- __pos__ 260
- Postleitzahlen erkennen 441
- Potenzieren 32
- __pow__ 259
- print 116
 - Anweisung 116

- Ausgabe in Fehlerkanal 117
- end 116
- end-Parameter 116
- file 117
- Funktion 116
- sep 116
- sep-Parameter 116
- sys.stderr 117
- Umlenkung in Datei 117
- printf 125
- private Attribute 240
- Produkt 32
- Programmablaufplan 64
- Programmiersprache, Unterschied zu Skriptsprache 21
- Properties 239
 - Definition mit Dekorateuren 245
- property
 - deleter 244
 - mit Dekorateuren 244
- protected Attribute 240
- public Attribute 240
- push 51, 289
- putenv 470

- Quantoren 441
- Quotient 32

- `__radd__` 263
- random-Methode
 - sample 364
- raw-Strings 41
- read 474
- readlink 485
- re-Modul 431
- realpath 496
- reduce 344
- Referenzen 25
- Regeln zur Interpretation von römischen Zahlen 518
- register_archive_format 502
- register_unpack_format 502
- reguläre Ausdrücke 429
 - Alternativen 449
 - Anfang eines Strings matchen 436
 - beliebiges Zeichen matchen 434
 - compile 450
 - Ende eines Strings matchen 436
 - findall 440
 - finditer 446
 - Kompilierung 450
 - optionale Teile 440
 - Quantoren 441
 - search 431
 - sub 455
 - Teilstringsuche 431
 - Überlappungen 431
 - vordefinierte Zeichenklassen 438
 - Wiederholungen von Teilausdrücken 441
 - Zeichenauswahl 435
- reguläre Auswahl
 - Caret-Zeichen, Zeichenklasse 435
- reguläre Mengen 429
- reguläre Sprachen 429
- Rekursion 167
 - Beispiel aus der natürlichen Sprache 167
- rekursiv → Rekursion
- rekursive Funktion 167, 168
- relpath 497
- remove 56, 485
- removedirs 485
- rename 486
- renames 486
- replace 207
- `__repr__` 246, 258
- Restdivision 32
- return 142
- rfind 205
- rindex 205
- rjust 209
- rmdir 486
- rmtree 502
- Robot-Klasse 541
- Roboter → Roboterklasse
- Robotergeretze 252
- Roboterklasse 228, 331
- römische Zahlen 80
- Rossum, Guido van 3, 339
- round_robin 379
- rpartition 204
- `__rshift__` 259
- rsplit 201
 - Folge von Trennzeichen 202
- rstrip 208
- Rückgabewerte 146
- Rückschritt → Escape-Zeichen
- Rückwärtsreferenzen 443

- samefile 497
- sample 364
- Sarah 133
- Schachbrett mit Weizenkörnern 82
- Schaltjahrberechnung 70, 295
- Schaltjahre 70

- Schleifen 66, 71
 - break 78
 - continue 78
 - Endekriterium 71
 - for-Schleife 77
 - kopfgesteuert 75
 - Schleifendurchlauf 71
 - Schleifenkopf 71
 - Schleifenkörper 71
 - Schleifenzähler 71
 - while-Schleife 72
- Schleifenzähler 71
- Schlüssel
 - Dictionary 83
 - zulässige Typen 88
- Schlüsselwortparameter 146
- Seiteneffekte 150
- Seitenumbruch → Escape-Zeichen
- semantische Fehler 408
- sep 491
- Sequentielle Datentypen 39
- Sequenz 39
- set 99
- set comprehension 350
- setdefault → Dictionary
- setegid 486
- seteuid 486
- setgid 486
- setgroups 479
- setpgid 487
- setpgrp 487
- setregid 487
- setresgid 487
- setresuid 487
- setreuid 487
- sets
 - add 101
 - clear 101
 - copy 102
 - difference 102
 - difference_update 102
 - discard 103
 - erweiterte Zuweisung 106
 - intersection 103
 - isdisjoint 104
 - issubset 104
 - issuperset 105
 - Operationen auf sets 101
 - pop 105
 - remove 103
 - union 104
- setsid 487
- Setter 238
- setuid 486
- setUp-Methode 419
- Shell 467
 - Bash 468
 - Bourne 468
 - C-Shell 468
 - CLI 467
 - GUI 467
 - Herkunft und Bedeutung des Begriffes 467
- Shell-Skripte ausführen 488
- shelve-Modul 425
- Shihram 82
- shutil
 - copy 498
 - copy2 498
 - copyfile 498
 - copyfileobj 498
 - copymode 499
 - copystat 499
 - copytree 499
 - get_archive_formats 500
 - get_unpack_formats 500
 - ignore_patterns 500
 - make_archive 500
 - move 501
 - register_archive_format 502
 - register_unpack_format 502
 - rmtree 502
 - unpack_archive 502
 - unregister_archive 502
 - unregister_unpack_format 502
- shutil-Modul 498
- Sieb des Eratosthenes 349
 - Rekursive Berechnung 175
 - Rekursive Funktion mit Mengen-Abstraktion 351
- silly_merge 462
- Simula 67 225
- sin 53
- singleton-Klasse 321
- Skriptsprache, Unterschied zu Programmiersprache 21
- Slicing 44
- Slots 307
- sort 177
 - eigene Sortierfunktionen 178
 - reverse 178
 - Umkehrung der Sortierreihenfolge 178
- sorted 177
- spezialisierte Klasse → Unterklasse
- splice 333

- split 198, 497
 - Folge von Trennzeichen 202
 - maxsplit 198
- splitdrive 497
- splitext 498
- splitlines 203
- Sprachfamilie 429
- sprintf 125
- Stack 51
 - Stapelspeicher 51
- Standardausnahmebehandlung 218
- Standardbibliothek 184
- Standardklassen als Basisklassen 289
- Stapelspeicher 51, 289
- stat 487
- stat_float_times 487
- staticmethod 253
- Statische Attribute 251
- Statische Methoden 253
- statische Typ-Deklaration 34
- Stelligkeit 152
- Stephen Cole Kleene 429
- StopIteration 354
- str 31, 246, 266
- strerror 488
- Stringformatierung 115
- Stringinterpolation 125
- Stringliterale 118
 - formatierte 115
- Stringmethoden
 - Alles in Großbuchstaben 207
 - Alles in Kleinbuchstaben 207
 - capitalize 208
 - center 209
 - count 205
 - find 205
 - index 205
 - isalnum 209
 - isalpha 209
 - isdigit 209
 - islower 209
 - isspace 209
 - istitle 209
 - isupper 209
 - ljust 209
 - lower 207
 - replace 207
 - rfind 205
 - rindex 205
 - rjust 209
 - rstrip 208
 - String-Tests 209
 - strip 208
 - title 208
 - upper 207
 - zfill 209
- Stringmodulo 125
- Strings 25, 31
 - Escape-Zeichen 41
 - formatieren 115
 - raw 41
 - Suchen und Ersetzen 207
 - Testmethoden 209
- rstrip 208
- Strukturierungselement 141
- __sub__ 259, 544
- Subklasse → Unterklasse
- Suchen und Ersetzen 207
- Suchen von Teilstrings 205
- sum 72
- Summe 32
- Summe von n Zahlen 72
 - Berechnung mit while-Schleife 72
- SUnit 417
- super 302
- Superklasse → Basisklasse
- supports_bytes_environ 470
- Sven 133
- symlink 488, 495
- Syntaxfehler 407
- sysconf 488
- system 488
- Systemprogrammierung 467
- tar-Datei → Archivdatei erzeugen
- TDD → test-driven development
- tearDown-Methode 419
- Teilbereichsoperator 44
- Tests 407
- test-driven development 416
- test first development 416
- TestCase 419
 - Methoden 419
 - setUp-Methode 419
 - tearDown-Methode 419
- testCase 417
- Testgesteuerte Entwicklung 416
- Testgetriebene Entwicklung 415, 416
- Testmethoden 419
- Textverarbeitung 197
- Theoretische Informatik 429
- throw-Methode
 - Generator 371
- tiefe Kopie 90, 139

- times 488
- Transiente Daten 423
- trigonometrische Funktionen 53
- True 31
- __truediv__ 259, 544
- try 215
- Tupel 42
 - entpacken 43
 - leere 58
 - mit einem Element 58
 - Packing 59
 - Unpacking 59
- tuple 42
- Typ-Alias 464
- Typ-Anmerkungen 459
 - Any 465
 - Callables 465
 - Dict 465
 - List 465
 - Listen 462
 - Variablen 461
- Typ-Variablenanmerkungen 461
- type 36, 227
- type annotations 459
- type conversion 35
- type hints 459
- type-Klasse 309
- TypeError 411
 - unhashable type 88
- typing-Modul 463
- Typumwandlung 35
 - explizit 36
 - implizit 36
- Typverletzung 35

- Überladen 281, 285
- überlagern 281
- Überlappungen 431
- Überschreiben 281, 285
- Uhr-Klasse 292
- umask 489
- Umgebungsvariablen 469
- uname 489
- unäre Operatoren 260
- Unärsystem 332, 544
- Under Matching 432
- unhashable type 88
- Unicode-Zeichen → Escape-Zeichen
- unittest 417 → Modultest
- Unix 468
- unlink 489
- unpack_archive 502
- Unpacking 59
- unregister_archive_format 502
- unregister_unpack_format 502
- unsetenv 470
- Unterklasse 279
- Unterstrich
 - Bedeutung in der interaktiven Shell 15
- immutable 44
- update → Dictionary
- upper 207
- urandom 489
- utime 489

- ValueError 216
- values → Dictionary
- Variable Anzahl von Parametern 152
- Variablen 25
 - global 162
 - nonlocal 162
 - Referenzen auf Objekte 25
- Variablennamen 28
 - Binnenversalien 29
 - CamelCase 29
 - gültige 28
 - Konventionen 29
- variadische Funktion 152
- Variation 362
- mutable 44
- Vererbung 279
 - Beispiel Angestelltenklasse, die von Person erbt 279
 - Beispiel Kundenklasse, die von Person erbt 279
 - Beispiel Lieferantenklasse, die von Person erbt 279
 - Beispiel Personenklasse 279
- Vererbungsbaum 291
- Vergleichsoperatoren 68, 260
- verschachtelte Dictionaries 89
- verschachtelte Funktionen 382
- Vertikaler Tabulator → Escape-Zeichen
- Verzeichnis löschen 486
- Verzweigungen 66
- view 89
- Vollständige Induktion 167

- wait 489
- wait3 489
- waitpid 489
- walk 489
- Weizenkornaufgabe 82
- while-Schleife 72

- while-Schleifen
 - optionaler else-Teil 74
- Wildcards 503
- with-Anweisung 112
- wraps-Dekorateur 397
- write 471

- __xor__ 259

- yield 355

- Zahlenratespiel 74
- Zeichenauswahl 435

- Zeichenkette 25, 31
- Zeilenumbruch → Escape-Zeichen
- Zeitrechnung 70
- Zen von Python 4, 54
- zfill 209
- zip-Datei → Archivdatei erzeugen
- zip-Funktion 92
- zip-Klasse 92
- Zugriffsmethoden 238
- Zustandsautomat 436
- Zustandsmaschine 436
- Zuweisung 26