

HANSER



Leseprobe

zu

Python für Ingenieure und Naturwissenschaftler

von Hans-Bernhard Woyand

Print-ISBN: 978-3-446-46483-4

E-Book-ISBN: 978-3-446-46501-5

E-Pub-ISBN: 978-3-446-46484-1

Weitere Informationen und Bestellungen unter
<http://www.hanser-fachbuch.de/978-3-446-46483-4>

sowie im Buchhandel

© Carl Hanser Verlag, München

Vorwort

Seit vielen Jahren halte ich an der Bergischen Universität Wuppertal die Lehrveranstaltung Informatik im Grundstudium Maschinenbau. Mehr als 10 Jahre haben wir in diesen Kursen die Programmiersprache C/C++ eingesetzt. Seit mehreren Jahren verwenden wir stattdessen nun *Python* als erste Programmiersprache und das mit großem Erfolg. Es zeigte sich, dass gerade bei Ingenieuren, die oftmals Schwierigkeiten mit dem algorithmischen Denken haben, Python einen leichteren Zugang ermöglicht.

Im Studiengang Maschinenbau wie auch in vielen anderen Studiengängen spielt das wissenschaftliche Rechnen und Visualisieren eine große Rolle. In der Literatur zu Python ist das nicht so: entweder werden die wissenschaftlichen Anwendungen gar nicht bzw. nur am Rande behandelt, oder die Bücher sind so umfangreich, dass sie als vorlesungsbegleitender Text ungeeignet sind. So entstand die Idee, das Skript zur Vorlesung zu diesem Buch auszuarbeiten.

Zielgruppe

Das Buch richtet sich an *Programmieranfänger*, die Python als erste Programmiersprache lernen möchten. Es wird also zuerst in Python eingeführt und dann werden die Grundlagen erläutert. Es folgen Vertiefungen und ein eigenes Kapitel über die objektorientierte Programmierung. Dann beginnt das wissenschaftliche Rechnen und Visualisieren. Hierzu wird in die Nutzung der wichtigsten Programmbibliotheken (Packages) eingeführt. Hauptzielgruppe sind also Studierende wissenschaftlicher Studiengänge, die Python erlernen wollen und bei denen die mathematischen und grafischen Anwendungen eine wichtige Rolle spielen. Vorausgesetzt wird nur mathematisch-naturwissenschaftliches Wissen, wie es an höheren Schulen vermittelt wird.

Aufgabenorientierte Lehre

Mehr als 90 Aufgabenstellungen mit fast immer kommentierten Lösungen werden im Buch behandelt. Nach meiner Erfahrung lernen Studierende am meisten durch das selbstständige Lösen von Aufgaben.

Software

Die Software, die benötigt wird, um mit diesem Buch zu arbeiten, ist kostenfrei erhältlich. Es handelt sich dabei um die neuere Variante 3.2 bzw. 3.5 der Programmiersprache Python, sowie deren Vorgängerversionen 2.6 und 2.7. Die wissenschaftlichen Anwendungspakete sind nämlich *noch nicht alle* mit der neuen Python-Version kompatibel. Es ist zu erwarten, dass es auch noch einige Zeit dauern wird, bis die wissenschaftlichen Pakete unter der neuesten Version laufen. Für Programmieranfänger ist der Unterschied zwischen diesen Versionen sowieso nicht sehr bedeutsam.

Webseite zum Buch

Zu dem vorliegenden Buch existiert eine Webseite, auf der in loser Folge Ergänzungen, Fehlerberichtigungen usw. bereitgestellt werden. Weiterhin können alle Beispiele sowie die Lösungen der Aufgaben dort abgerufen werden. Für Nutzer des Betriebssystems MS-Windows sind auch Hinweise zur Installation der Software dort verfügbar. Die Web-Adresse ist

http://woyand.eu/python_buch

Außerdem sind die Daten zum Buch unter folgender Webadresse aufrufbar:

<http://www.plus.hanser-fachbuch.de>.

Die Zugangsdaten finden Sie auf Seite 1 des Buches.

Hinweise, Fehlermeldungen und Anregungen werden über die E-Mail-Adresse

info@woyand.eu

gern entgegengenommen.

Haftungsausschluss

Das Buch wurde mit größtmöglicher Sorgfalt erstellt. Trotzdem können Fehler nicht ganz ausgeschlossen werden. Aus diesem Grund sind die in diesem Buch dargestellten Verfahren mit keinerlei Garantie verbunden. Ich weise darauf hin, dass weder Autor noch Verlag eine Haftung für direkt oder indirekt entstandene Schäden übernehmen, die sich aus der Benutzung dieses Buches ergeben könnten.

Danksagung

Ich danke meiner Frau Annette Woyand für die sorgfältige Durchsicht des Manuskripts. Frau Mirja Werner vom Carl Hanser Verlag danke ich für Ihr Engagement beim Zustandekommen dieses Buches.

Ich wünsche allen Lesern viel Erfolg und Spaß beim Einstieg in Python.

Wuppertal, im März 2017

Hans-Bernhard Woyand

■ Vorwort zur zweiten Auflage

Für die zweite Auflage wurden einige inhaltliche Verbesserungen vorgenommen und ein neues Kapitel hinzugefügt. In diesem Kapitel wird gezeigt, wie numerische Berechnungen mit der Python-Bibliothek Scipy durchgeführt werden können. Scipy ist sehr umfangreich. Deshalb werden – dem Ansatz dieses Buches entsprechend – wichtige Teilbereiche dieser Software-Bibliothek auf wenigen Seiten vorgestellt. Behandelt wird die numerische Berechnung von Integralen, die Interpolation, die Berechnung von Nullstellen, die numerische Optimierung, die Signalanalyse mit der schnellen Fourier Transformation (FFT) sowie die numerische Integration gewöhnlicher Differenzialgleichungen.

Ich hoffe, dass diese Erweiterung des Buchs für viele Leser hilfreich und anregend ist und wünsche viel Erfolg und Spaß mit Python.

Wuppertal, im Juli 2018

Hans-Bernhard Woyand

■ Vorwort zur dritten Auflage

Für die dritte Auflage wurde das Kapitel 8 (3D-Grafik mit VPython) völlig überarbeitet, da sich die neue Version VPython 7 deutlich von den Vorgängerversionen unterscheidet. Mit dieser neuen Version von VPython können die dreidimensionalen Szenen im Webbrowser dargestellt werden. Auch die Benutzerinteraktionen wurden erheblich vereinfacht und neu strukturiert.

Weiterhin wurde das Kapitel 10 (Numerische Analysen mit Scipy) um zwei Themen erweitert. Es handelt sich um die Erzeugung von Dreiecksnetzen mit der Delaunay-Triangulierung sowie um die Berechnung der konvexen Hülle einer Punktmenge. Vier weitere Aufgaben mit kommentierten Lösungen wurden dem Buch hinzugefügt.

Wuppertal, im Januar 2019

Hans-Bernhard Woyand

■ Vorwort zur vierten Auflage

Für die vierte Auflage des Buches wurden einige kleinere, inhaltliche Verbesserungen vorgenommen. Zudem wurde ein neues Kapitel 11 mit dem Titel „Bildverarbeitung mit scikit-image“ hinzugefügt. Scikit-image ist ein Erweiterungspaket der Programmiersprache Python, das zur automatisierten Manipulation und Analyse von Bildern (Fotos, Grafiken) entwickelt wurde. Es wird gezeigt, wie Bilder eingelesen, gewandelt und geschrieben werden können. Weiterhin werden Algorithmen zum Auffinden von Kanten, Ecken und Kreisen sowie zum Abgleich von Vorlagen vorgestellt und erläutert. Zwei weitere Aufgaben und deren Lösungen wurden ergänzt.

Witten, im November 2020

Hans-Bernhard Woyand

Inhalt

Vorwort	V
1 Einführung	1
1.1 Die Programmiersprache Python	1
1.2 Hinweise zur Installation	2
1.3 Erste Schritte - der Python-Interpreter	3
1.3.1 Addition und Subtraktion	4
1.3.2 Multiplikation und Division	4
1.3.3 Vergleichsausdrücke	6
1.3.4 Logische Ausdrücke	7
1.3.5 Mathematische Funktionen	7
1.3.6 Grundlegendes über Variablen und Zuweisungen	9
1.3.7 Zeichenketten (Strings)	10
1.3.8 Turtle-Grafik	10
1.4 Python-Programme mit IDLE erstellen	12
1.5 Aufgaben	18
1.6 Lösungen	22
2 Grundlagen	31
2.1 Einfache Objekttypen	31
2.1.1 Ganze Zahlen - Integer	31
2.1.2 Gleitpunktzahlen - Float	33
2.1.3 Komplexe Zahlen - Complex	34
2.1.4 Zeichenketten - Strings	36
2.1.5 Aufgaben	41
2.1.6 Lösungen	43
2.2 Operatoren und mathematische Standardfunktionen	46
2.2.1 Operatoren zur arithmetischen Berechnung	46
2.2.2 Mathematische Standardfunktionen	47

2.2.3	Aufgaben	49
2.2.4	Lösungen	50
2.3	Variablen und Zuweisungen	51
2.4	Funktionen	56
2.4.1	Funktionen mit Rückgabewert	57
2.4.2	Funktionen ohne Rückgabewert	60
2.4.3	Aufgaben	62
2.4.4	Lösungen	64
2.5	Ein- und Ausgabe	65
2.6	Programmverzweigungen	68
2.6.1	Einfache if-Anweisung	68
2.6.2	Erweiterte if-Anweisung	70
2.6.3	Aufgaben	72
2.6.4	Lösungen	73
2.7	Bedingungen	73
2.8	Programmschleifen	75
2.8.1	for-Schleifen	76
2.8.2	while-Schleifen	80
2.9	Aufgaben	84
2.10	Lösungen	85
3	Vertiefung	89
3.1	Listen	89
3.1.1	Aufgaben	94
3.1.2	Lösungen	96
3.2	Tupels	100
3.3	Sets – Mengen	101
3.4	Dictionaries	103
3.4.1	Aufgaben	106
3.4.2	Lösungen	107
3.5	Slicing	110
3.6	List Comprehensions	113
3.7	Iteratoren und die zip-Funktion	114
3.8	Funktionen, Module und Rekursion	116
3.8.1	Schlüsselwort-Parameter	116
3.8.2	Module	117
3.8.3	Rekursion	119
3.8.4	Globale und lokale Variablen	121

3.9	Turtle-Grafik – verbessert	123
3.10	Dateien lesen und schreiben	125
3.11	Aufgaben	130
3.12	Lösungen	136
4	Objektorientiertes Programmieren	149
4.1	Klassen und Objekte	149
4.1.1	Die Grundidee	150
4.1.2	Klassen	151
4.1.3	Methoden	153
4.2	Konstruktoren und Destruktoren	158
4.3	Überladen von Operatoren	161
4.4	Vererbung	165
4.5	Aufgaben	169
4.6	Lösungen	172
5	Numerische Berechnungen mit Numpy	183
5.1	Hinweise zur Installation	183
5.2	Arrays	184
5.3	Darstellung von Matrizen	185
5.4	Spezielle Funktionen	186
5.5	Operationen	187
5.6	Lineare Algebra	188
5.7	Zufallswerte	190
5.8	Aufgaben	190
5.9	Lösungen	192
6	Grafische Darstellungen mit Matplotlib	195
6.1	Hinweise zur Installation	195
6.2	XY-Diagramme	195
6.3	Balkendiagramme	200
6.4	Tortendiagramme	202
6.5	Polardiagramme	203
6.6	Histogramme	204
6.7	Subplots	205

6.8	Axes	207
6.9	Anmerkungen und Legenden	209
6.10	Aufgaben	211
6.11	Lösungen	211
7	Computeralgebra mit Sympy	215
7.1	Hinweise zur Installation	216
7.2	Differentiation	216
7.3	Integration	217
7.3.1	Unbestimmte Integrale	218
7.3.2	Bestimmte Integrale	218
7.3.3	Uneigentliche Integrale	219
7.4	Potenzreihen	220
7.5	Matrizenrechnung – lineare Algebra	220
7.6	Die Datentypen Rational und Float	222
7.7	Nützliche Ergänzungen	223
7.8	Aufgaben	226
7.9	Lösungen	227
8	3D-Grafik mit VPython 7	231
8.1	Hinweise zur Installation	231
8.2	Szenen	232
8.3	Grundkörper	237
8.4	Dreieck- und Viereckflächen (Triangle/Quad)	243
8.4.1	triangle	243
8.4.2	quad	244
8.4.3	STL-Dateien lesen und mit VPython darstellen	245
8.5	Widgets	248
8.6	Steuerung mit Tastatur und Maus	252
8.7	Aufgaben	260
8.8	Lösungen	262
9	Python-Versionen, Programmbibliotheken und Distributionen	271
9.1	Python 2	272
9.2	Die Python-Distribution Anaconda	274

9.3	Die Python-Distribution WinPython	276
9.4	Aufgaben	276
9.5	Lösungen	278
10	Numerische Analysen mit Scipy	281
10.1	Hinweise zur Installation	282
10.2	Numerische Berechnung von Integralen	282
10.3	Interpolation	284
10.4	Berechnung von Nullstellen – Rootfinding	287
10.5	Optimierung	289
10.6	Signalanalyse mit der Schnellen Fourier Transformation (FFT)	293
10.7	Numerische Integration gewöhnlicher Differenzialgleichungen	297
10.8	Delaunay-Triangulierung	303
10.9	Berechnung der konvexen Hülle	304
10.10	Aufgaben	306
10.11	Lösungen	308
11	Bildverarbeitung mit scikit-image	317
11.1	Hinweise zur Installation	317
11.2	Bilder einlesen, darstellen und ausgeben	317
11.3	Farbbilder in Graustufenbilder wandeln und Bilder skalieren ..	319
11.4	Graustufenbild durch Programmanweisungen erzeugen	320
11.5	Ecken ermitteln – Corner Detection	322
11.6	Kanten detektieren – Canny-Filter	323
11.7	Kreise erkennen – Hough-Transformation	324
11.8	Abgleich von Vorlagen – Template-Matching	327
11.9	Aufgaben	329
11.10	Lösungen	330
	Literaturverzeichnis	333
	Index	335

1

Einführung

In diesem Kapitel wird die Programmiersprache Python vorgestellt. Nach Bemerkungen zur Installation dieser Sprache wird gezeigt, wie Python interaktiv ausgeführt werden kann. Schließlich wird dargestellt, wie ein vollständiges Python-Programm geschrieben wird. Neben dem Umgang mit Zahlen und Ausdrücken wird insbesondere auf das Konzept der Variablen eingegangen.

■ 1.1 Die Programmiersprache Python

Die Sprache Python wurde in den neunziger Jahren des letzten Jahrhunderts von *Guido van Rossum* entwickelt. Mittlerweile ist Python eine der meistgenutzten Programmiersprachen. Die wesentlichen Vorzüge dieser Sprache sind:

- Python ist *leicht zu erlernen*, unterstützt mehrere Programmierparadigmen und ist klar strukturiert.
- Python eignet sich insbesondere zur *schnellen Entwicklung* von Softwareprototypen (RAD - Rapid Application Development). Dies ist gerade für Anwender in technisch-naturwissenschaftlichen Bereichen ein wichtiger Aspekt.
- Python ist *portabel*. Die Programme, die mit dieser Sprache geschrieben werden, laufen im Allgemeinen ohne Änderungen auf LINUX-, MAC OS- und Windows-Betriebssystemen.
- Python beinhaltet eine *Fülle von Anwendungspaketen* für unterschiedliche Bereiche. Ob grafische Darstellungen, numerische Berechnungen, Datenbanken oder Webanwendungen zu entwickeln sind: Der Anwender hat zumeist nur das Problem, aus der Vielzahl von angebotenen Lösungen, die für ihn geeignete zu finden. Es gibt kaum einen Anwendungsbereich, für den Python nicht schon vorgefertigte „Tools“ zur Verfügung stellt.
- Python ist *kostenlos*. Die Sprache kann auch für kommerzielle Zwecke kostenfrei genutzt werden.

- Python kann *leicht erweitert* oder selbst in Programme anderer Sprachen „eingebettet“ werden. Ein etwas fortgeschrittener Python-Programmierer ist in der Lage, eigene Anwendungspakete zu entwickeln.
- Mit Python können Programme geschrieben werden. Es ist aber auch möglich, einzelne Anweisungen interaktiv in der sogenannten Python-Shell auszuführen. Diese *interaktive Ausführung* hilft beim Ausprobieren von Sprachstrukturen und auch beim Testen von Programmen.

Im Rahmen dieses Buches steht die *leichte Erlernbarkeit* dieser Programmiersprache im Vordergrund.

■ 1.2 Hinweise zur Installation

In diesem Buch wird die Installation der Software *nicht* gezeigt. Für Nutzer des MS-Windows-Betriebssystems gibt es allerdings ein Zusatzdokument im PDF-Format, das die Installation Schritt für Schritt darstellt. Dieses Dokument kann auf den Webseiten:

http://www.woyand.eu/python_buch/ bzw. [plus.hanser-fachbuch.de](http://www.plus.hanser-fachbuch.de)

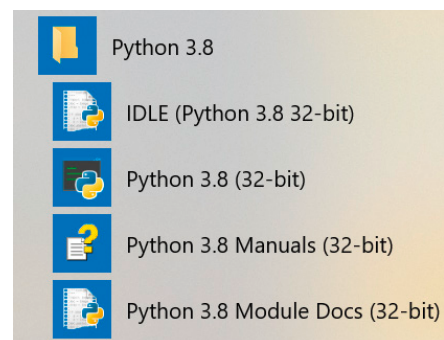
heruntergeladen werden. Zusammen mit der Anleitung kann der Leser dort auch ein ZIP-Verzeichnis mit einigen Installationsdateien sowie die Lösungen der Aufgaben und den Programmcode der Beispiele erhalten.

Eine Darstellung der Installation für alle gängigen Betriebssysteme (LINUX, Mac OS) würde den Rahmen des Buchs sprengen. Auf LINUX-Systemen ist Python oftmals schon vorinstalliert. Alle Installationsdateien sind auf der folgenden Python-Homepage unter dem Menüpunkt „Download“ zu finden:

<http://www.python.org>

Die Sprachversion, die in diesem Buch verwendet wird ist Python 3. Die Beispiele und Aufgaben wurden mit der relativ neuen Version 3.7.6 getestet.

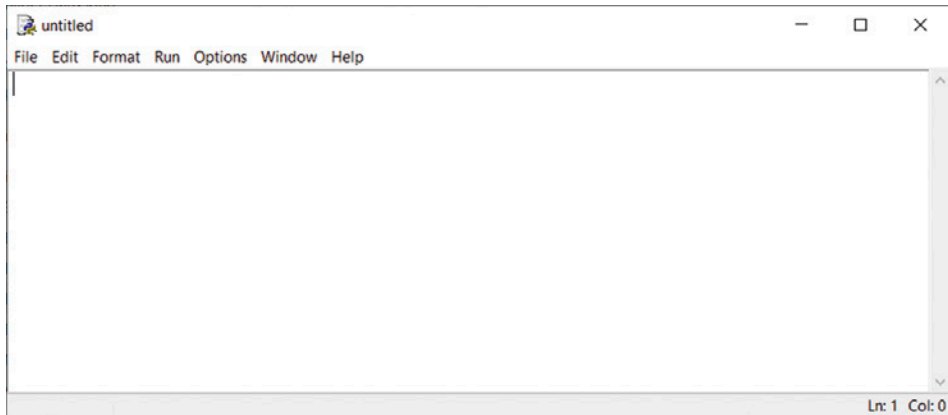
Nach erfolgreicher Installation kann Python gestartet werden. Hierzu wird über das Windows-Startmenü der Eintrag „Alle Programme“ ausgewählt. Um mit Python zu arbeiten, wählen Sie wie im Bild rechts beispielhaft gezeigt einfach „Python 3.8“ und dann den Eintrag „IDLE ...“ aus.



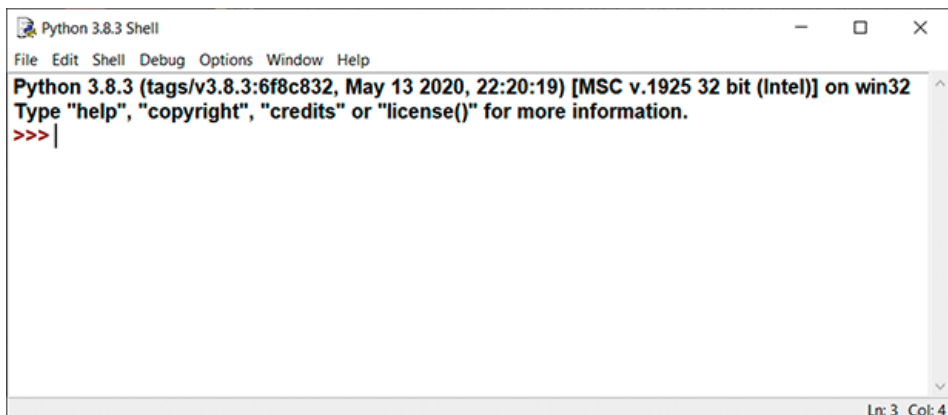
IDLE ist der Name der *Standard-Entwicklungsumgebung* für Python. Um einen Brief zu schreiben, kann beispielsweise Microsoft Word zur Eingabe verwendet werden. Soll dagegen ein Python-Programm geschrieben werden, so gibt man dieses Programm mithilfe von *IDLE* ein und kann es anschließend starten. Für den schnellen Zugriff auf IDLE kann eine Verknüpfung auf dem Desktop abgelegt werden.

■ 1.3 Erste Schritte – der Python-Interpreter

Wenn IDLE aufgerufen wird – wie im letzten Abschnitt gezeigt – so wird entweder das Editorfenster wie unten gezeigt oder das Shell-Fenster geöffnet. Welches Fenster geöffnet wird, hängt von den Voreinstellungen der Software ab.



Für den Fall, dass das Editorfenster geöffnet wurde, wählen wir aus dem Menü die Kommandofolge „Run → Python Shell“ aus. Daraufhin öffnet sich die „Python-Shell“ (siehe Bild). In den folgenden Unterkapiteln wird zunächst diese Shell verwendet.



**Hinweis**

Je nach Software-Version kann sich nach dem Aufruf von IDLE zunächst das Editorfenster oder die Python-Shell öffnen. Welches der Fenster nach dem Aufruf von IDLE geöffnet wird, kann über eine Einstellung im Optionen-Menü festgelegt werden. Wählen Sie hierzu in IDLE „Options“ und dann „Configure IDLE“. Unter der Registerkarte „General“ kann unter dem Eintrag „At startup“ die Option „Open Edit Window“ oder „Open Shell Window“ gewählt werden.

Die drei aufeinanderfolgenden Zeichen „>>>“ werden *Eingabeaufforderung* oder *Eingabeprompt* genannt. Sie können nun damit beginnen, Ausdrücke in der Sprache Python einzugeben und diese unmittelbar ausführen zu lassen. Dieser interaktive Modus der Programm-Eingabe und -Ausführung ist typisch für sogenannte Interpreter-Sprachen wie Python. Gerade Programmieranfänger profitieren viel davon, weil sie einzelne Befehle unmittelbar ausprobieren können.

Wir beginnen nun mit einer kleinen Rundtour durch Python. Wenn wir etwas falsch machen, weil wir beispielsweise gegen die Regeln der Sprache verstoßen, so erhalten wir eine Fehlermeldung, die vom System in *roter Farbe* ausgegeben wird.

1.3.1 Addition und Subtraktion

Im Folgenden wird nicht mehr das ganze Fenster, sondern nur noch der Eingabeprompt und die darauffolgende Antwort des Computers dargestellt. Probieren Sie am besten die nachfolgenden Eingaben selbst aus!

```
>>> 1+2
3
>>> 1-2
-1
```

Wie wir sehen, wird nach jeder unserer Eingaben die Antwort von Python ausgegeben. Der Operator + bedeutet dabei Addition, der Operator - Subtraktion.

1.3.2 Multiplikation und Division

Die folgenden Eingaben zeigen, dass mit den Zeichen * eine Multiplikation und mit / eine Division durchgeführt werden kann.

```
>>> 1+2*3
7
>>> (1+2)*3
9
```

```
>>> 2+7/5
3.4
>>> 1/2
0.5
```

Diese Beispiele zeigen, dass wir mithilfe von Klammern die Auswertungsreihenfolge innerhalb von mathematischen Ausdrücken beeinflussen können. Dies geschieht so, wie wir das intuitiv aus dem Prinzip „Punkt- vor Strichrechnung“ vermuten. Zuerst wird die Multiplikation bzw. die Division ausgeführt, anschließend die Addition bzw. die Subtraktion. Das Setzen von Klammern erzwingt gegebenenfalls eine andere Auswertungsreihenfolge.

Der Ausdruck $2+7/5$ ergibt die Zahl 3.4. Statt des Kommas, das wir beim Schreiben auf Papier verwenden, werden Dezimalzahlen in fast allen Programmiersprachen mithilfe eines *Dezimalpunkts* codiert. Statt 5,0 schreiben wir also 5.0. Wir sprechen deshalb in der Programmierung von *Gleitpunktzahlen*. Wenn wir das Divisionszeichen in Python 3.x verwenden, so wird immer eine sogenannte *Gleitpunkt-Division* durchgeführt, d.h. auch wenn beide Operanden ganzzahlig sind, ist das Ergebnis eine Gleitpunktzahl.

```
>>> 7/5
1.4
```

Das ist wichtig zu wissen! In der älteren Python-Version 2.x wurde eine Ganzzahl-Division durchgeführt, wenn beide Operanden ganze Zahlen waren. Dies führte oft zu unbeabsichtigten Fehlern bei Ausdrücken wie $1/2$. Das Ergebnis war dann 0. Soll in Python 3.x eine Ganzzahl-Division durchgeführt werden, so muss dies mit dem speziellen Operator `//` codiert werden. Hierzu auch ein Beispiel:

```
>>> 7//5
1
>>> 22//4
5
>>> 22.0//4.0
5.0
```

Wenn man sich für den Rest bei der Durchführung einer ganzzahligen Division interessiert, so kann dieser mit dem Operator `%` ermittelt werden. Dieser Operator wird *Modulo-Operator* genannt.

```
>>> 7%5
2
>>> 22%4
2
```

**Merke**

In Python 2.x – wie auch in anderen Programmiersprachen (z. B. C/C++) – muss beachtet werden, dass bei der Division von ganzen Zahlen eine Ganzzahl-Division durchgeführt wird. Ist jedoch nur einer der Operanden eine Dezimalzahl, so wird eine Gleitpunkt-Division ausgeführt. In der neueren Sprachversion Python 3.x, die wir in diesem Buch benutzen, wurde dies abgeschafft. Dort wird jede Division, die mit dem Operator / ausgeführt wird, als Gleitpunkt-Division durchgeführt. Der Programmierer kann jedoch auch eine Ganzzahl-Division durch die Anwendung eines besonderen Operators (// statt /) ausführen lassen.

Das folgende Beispiel zeigt, dass sogenannte rein periodische Zahlen natürlich nur mit einer endlichen Anzahl von Nachkommastellen dargestellt werden können.

```
>>> 32/3
10.666666666666666
```

Der Digitalrechner kann *nicht jede Zahl* exakt darstellen. Aufgrund der internen Darstellung in Form von binären Zuständen (jede Zahl wird intern durch eine Folge von Nullen und Einsen aufgebaut) können nicht alle Zahlen völlig präzise dargestellt werden. Dieser Sachverhalt spielt eine wichtige Rolle beim numerischen Rechnen. In unserem Fall ist die Abweichung klein und belanglos. Werden jedoch sehr viele arithmetische Operationen durchgeführt, so kann sich ein erheblicher Gesamtfehler akkumulieren.

1.3.3 Vergleichsausdrücke

Fahren wir mit unserer Erkundungstour durch Python fort. Von Handrechnungen kennen wir die Vergleichsoperatoren < und >. Wir probieren sie aus:

```
>>> 2<7
True
>>> 2>7
False
```

Vergleichsausdrücke werden von Python als wahr (engl. true) und falsch (engl. false) ausgewertet.

```
>>> 2==7
False
>>> 2!=7
True
>>> 5==5
True
```


Beim Testen auf Gleichheit wird kein Gleichheitszeichen geschrieben, sondern *zwei aufeinanderfolgende Gleichheitszeichen*. Beim Test auf Ungleichheit wird der Operator „!“ verwendet. Dieser Operator wird auch in den populären Programmiersprachen C und C++ verwendet. Python entlehnt viele Sprachelemente aus diesen Sprachen.

```
>>> 3<3
False
>>> 3<=3
True
```

Das letzte Beispiel zeigt uns den Unterschied zwischen „<“ und „<=“. Probieren Sie selbst ein Beispiel mit „>“ und „>=“ aus!

1.3.4 Logische Ausdrücke

Ausdrücke, die wahr oder falsch sind, können zu komplexeren Ausdrücken mithilfe der logischen Operatoren *and*, *or* und *not* zusammengefügt werden. Hier einige Beispiele für solche Ausdrücke, die auch boolesche (engl. boolean) Ausdrücke genannt werden.

```
>>> not(5==5)
False
>>> (2<7) and (5>4)
True
>>> (2>7) or (4>5)
False
>>> not(3!=3)
True
```

Ein Ausdruck der mit dem Und-Operator gebildet wird, ist dann und nur dann wahr, wenn beide Teilausdrücke bzw. Operanden wahr sind. Umgekehrt ist ein mit dem Oder-Operator gebildeter Ausdruck schon dann wahr, wenn nur ein einziger Operand wahr ist.

1.3.5 Mathematische Funktionen

Wir wollen mathematische Funktionen anwenden und probieren Folgendes aus:

```
>>> sin(90)

Traceback (most recent call last):
  File "<pyshell#32>", line 1, in <module>
    sin(90)
NameError: name 'sin' is not defined
```

Unsere Absicht war es, den Sinuswert von 90 (Grad) auszurechnen. Nun werden wir zum ersten Mal mit einer Fehlermeldung von Python konfrontiert. Diese ist auch hinreichend klar in englischer Sprache formuliert: „name 'sin' is not defined“. Dies bedeutet übersetzt etwa: „der Name ‚sin‘ ist nicht definiert“. Sollte Python keine mathematischen Standardfunktionen bereitstellen? Dann wäre ja jeder Taschenrechner „intelligenter“. Tatsächlich ist es so, dass wir die mathematischen Funktionen zunächst laden müssen, bevor wir diese anwenden können. Dies geschieht durch eine import-Anweisung. Mithilfe dieser Anweisung wird ein sogenanntes „Modul“ geladen. Was dies genau bedeutet, werden wir im Kapitel 3 genauer verstehen lernen.

```
>>> from math import *
>>> sin(90)
0.89399666360055785
>>> sin(180)
-0.80115263573383044
```

Nachdem wir die mathematischen Funktionen aus dem Modul *math* importiert haben, können wir die Sinusfunktion problemlos aufrufen. Dies gilt auch für andere mathematische Standardfunktionen. Probieren Sie einfach mal den Kosinus (cos) aus!

Allerdings wird uns auch klar, dass der Zahlenwert, der dieser Funktion übergeben wird, *nicht* als Winkel interpretiert wird. Sonst müssten im vorigen Beispiel die Werte 1 und 0 herauskommen. Wir vermuten, dass das Argument *im Bogenmaß* einzugeben ist und wandeln das Beispiel etwas ab.

```
>>> pi
3.141592653589793
>>> sin(pi/2)
1.0
>>> sin(pi)
1.2246467991473532e-16
```

Der math-Modul muss nur einmal pro Sitzung importiert werden. In diesem Modul ist eine Variable mit Namen pi definiert. Wir berechnen den Sinuswert von pi/2 (entsprechend 90 Grad) und von pi (entsprechend 180 Grad). Die letzte Zeile im letzten Codeabschnitt ist auf den ersten Blick verwirrend. Es sollte eigentlich genau 0 herauskommen. Stattdessen erhalten wir 1.2246467991473532e-16. Dies bedeutet in der Schreibweise für *Dezimalzahlen*: 1,2246467991473532 10^{-16} .

Dies ist eine sehr kleine Zahl, fast null. Damit haben wir auch eine zweite Darstellungsform für Gleitpunktzahlen kennen gelernt: die *Exponentendarstellung*.



Merke

Die mathematischen Standardfunktionen Sinus, Cosinus, Tangens etc. werden mit dem Bogenmaß als Argument aufgerufen. Sie müssen einen Winkel also zuerst ins Bogenmaß umrechnen, um dann den Wert der Funktion zu ermitteln.

1.3.6 Grundlegendes über Variablen und Zuweisungen

Wir wollen nun das wichtige Konzept der *Variablen* kennenlernen. Betrachten Sie dazu den folgenden Programmcode:

```
>>> a

Traceback (most recent call last):
  File "<pysshell#44>", line 1, in <module>
    a
NameError: name 'a' is not defined
>>> a = 7
>>> b = 3
>>> c = a*b
>>> print("c = ",c)
c = 21
>>> a
7
```

Zunächst geben wir den Buchstaben `a` ein. Python antwortet daraufhin mit der Fehlermeldung, dass der Name `a` nicht definiert ist.

Dies wird mit der folgenden Zeile durch die Anweisung `a = 7` nachgeholt. Bei dieser Anweisung handelt es sich um eine sogenannte *Zuweisung*. Es wird ein Name `a` erklärt und diesem Namen – genannt Variable – wird ein konstanter Wert `7` zugewiesen. Wir wollen uns vorläufig vorstellen, dass die Variable `a` von nun an stellvertretend für die Zahl `7` steht. Genauer gesagt, bildet die Variable einen „Behälter“, der den Wert `7` aufgenommen hat. Etwas Vergleichbares geschieht in den folgenden beiden Zeilen. Dort wird zunächst eine Variable `b` erklärt und erhält den Wert `3` zugewiesen. Schließlich wird eine Variable mit Namen `c` erzeugt. Diese Variable beinhaltet das Ergebnis der Multiplikation von `a` mit `b`. Damit dieser Wert am Bildschirm ausgegeben wird, verwenden wir eine eingebaute Funktion von Python: die `print()`-Funktion. Diese Funktion gibt den Text „`c =`“ in der Python-Shell aus, gefolgt von dem Inhalt der Variablen `c`. Zusammen wird das so geschrieben:

```
print("c = ",c)
```

Schließlich sehen wir an den letzten beiden Zeilen, dass der Name `a` nun bekannt ist, nachdem ihm ein Wert zugewiesen wurde. Gibt man einfach nur den Buchstaben `a` ein, antwortet Python mit dem Inhalt der Variable, d. h. mit dem Wert, der dieser Variablen zugewiesen wurde. Ebenso hätten wir den Inhalt von `c` ausgeben können.

Variablen (sie werden auch Bezeichner) genannt, können beliebig viele Zeichen umfassen. Die Regeln zur Bildung solcher Namen sind in Kurzform:

- Namen für Variablen können aus Buchstaben, Ziffern (0..9) und einem einzigen Sonderzeichen bestehen. Dieses Sonderzeichen ist der Unterstrich „`_`“ (engl. underscore).

- Das erste Zeichen in einem Variablennamen darf *keine Ziffer* sein. Ein Unterstrich ist allerdings als erstes Zeichen erlaubt.
- Deutsche Umlaute (Ä, ä usw.) dürfen in Python 3.x verwendet werden. In Python 2.x sind diese Zeichen *nicht* erlaubt.
- Es wird zwischen Groß- und Kleinschreibung unterschieden.
- Variablennamen dürfen *nicht* mit den reservierten Worten der Programmiersprache Python übereinstimmen.

Es wird empfohlen, möglichst selbsterklärende Namen für die Variablen zu erfinden, die einen Zusammenhang mit der durch das Programm zu lösenden Problematik schon im Namen ausdrücken. Also möglichst nicht „a“, „b“ und „c“ wie im vorangegangenen Beispiel, sondern „Zylinder_Durchmesser“, „StartZeit“, „Gesamt_Summe“ usw. Damit werden die Programme verständlicher und besser lesbar.

1.3.7 Zeichenketten (Strings)

Auch im technisch-wissenschaftlichen Bereich müssen oft Programme geschrieben werden, die Texte verarbeiten. Eine sogenannte Zeichenkette (engl. string) besteht aus beliebigen Zeichen und wird in Anführungszeichen gesetzt. Das folgende Beispiel zeigt, dass auch für solche Zeichenketten der „+“-Operator existiert. Dieser ist also *kontextsensitiv* und führt etwas anderes durch, wenn seine Operanden Zeichenketten statt Zahlen sind. In diesem Fall werden die Zeichenketten aneinander gehängt (engl. concatenation). In einem String dürfen Umlaute auch verwendet werden.

```
>>> Erster_Name="Bergische "  
>>> Zweiter_Name="Universität"  
>>> Name = Erster_Name + Zweiter_Name  
>>> print(Name)  
Bergische Universität
```

1.3.8 Turtle-Grafik

Python beinhaltet ein Modul zur Erstellung von einfachen Liniengrafiken. Die Programmierung funktioniert nach dem „Schildkrötenprinzip“, d. h. der Programmierer steuert ein Symbol – Schildkröte genannt – mit einfachen Befehlen innerhalb eines Grafikensters. Durch die Bewegung der Schildkröte (engl. turtle) wird dann die Grafik erzeugt. Die Turtle-Grafik ist ein gutes Hilfsmittel zum Erlernen der Programmierung. Wir werden allerdings im Rahmen dieses Buches auch anspruchsvollere Grafiken mit diesem Tool erzeugen.

Im Prinzip genügen zunächst elf Befehle (besser gesagt: Funktionsaufrufe), um mit der Schildkröte erste interessante Grafiken erzeugen zu können.

- `forward(steps)`: Mit diesem Befehl kriecht die Schildkröte in ihrer Blickrichtung vorwärts. Sie geht dabei um so viele Schritte bzw. *Pixel* nach vorn, wie der Parameter *steps* vorgibt. `fd(steps)` ist die Abkürzung dieses Befehls.
- `left(angle)`: Mit diesem Befehl ändert die Schildkröte ihre Richtung. Sie dreht sich um den Winkel *angle* nach links. Der Winkel wird dabei in *Grad* angegeben. `lt(angle)` ist eine Abkürzung des Befehls „left“.
- `right(angle)`: Mit diesem Befehl ändert die Schildkröte ihre Richtung. Sie dreht sich um den Winkel *angle* nach rechts. Der Winkel wird dabei auch in *Grad* angegeben. `rt(angle)` ist eine Abkürzung dieses Befehls.
- `color(col)`: Mit diesem Befehl kann die Farbe der Linien, welche die Schildkröte zeichnet, beeinflusst werden. Die Variable *col* kann die Werte „red“, „green“, „blue“, „yellow“ etc. annehmen. Statt der Anführungszeichen können auch Apostrophe verwendet werden (z. B. `color('red')`).
- `pensize(pixel)`: Mit diesem Befehl wird die Dicke der zu zeichnenden Linien in Pixel gesteuert.
- `penup()` und `pendown()` hebt und senkt den „Zeichenstift“ der Schildkröte.
- `reset()`: Dieser Befehl löscht den Inhalt des Grafikfensters und setzt die Schildkröte auf ihren Anfangszustand zurück (Ausrichtung nach rechts).
- `goto(x,y)`: Mit diesem Befehl springt die Schildkröte auf die Position (x,y), wobei *x* und *y* in Pixel gemessen werden. Das Bezugskoordinatensystem befindet sich in der oberen rechten Ecke des Zeichenfensters. Die *x*-Achse ist horizontal ausgerichtet und die *y*-Achse zeigt senkrecht nach unten.
- `bye()`: Mit diesem Befehl wird das Turtle-Grafikfenster geschlossen.
- `exitonclick()`: Mit diesem Befehl wird das Grafikfenster geschlossen, wenn der Benutzer mit der Maus in das Fenster klickt.



Merke

Am Ende einer Turtle-Befehlsfolge sollte der Befehl `bye()` ausgeführt werden, sonst „hängt“ das Turtle-Grafikfenster unter dem Betriebssystem.

Am Anfang – dies ist so vereinbart – steht die Schildkröte mit Blickrichtung nach rechts. Wir wollen dies nun erproben. Hierzu muss zuerst das Modul „turtle“ importiert werden, damit die obigen Befehle zur Verfügung stehen. Dabei bedeutet das Sternchen *, dass *alle* Befehle aus dem Modul *turtle* importiert werden.

Index

Zahlen

3D-Grafik 231

A

Ableich von Vorlagen 327
add_subplot()-Methode (Matplotlib) 205,
206
Anaconda 274
Animationen (VPython) 258
annotate()-Methode (Matplotlib) 209
Anwendungspaket 271
append()-Methode 91
arange()-Funktion (Numpy) 186
Array 184
arrow() (VPython) 239
asin()-Funktion 48
atan2()-Funktion 48
atan()-Funktion 48
Attribute 151
Axes 207
axes()-Methode (Matplotlib) 207

B

bar()-Methode (Matplotlib) 200
barh()-Methode (Matplotlib) 201
Bedingung 69, 73
Bedingungsschleife 80
bin()-Funktion 32
bind()-Methode (VPython) 254
box() (VPython) 237
Button (VPython) 249

C

Canny-Filter 323
center()-Methode 38
Checkbox (VPython) 249
class 152
close()-Funktion 126
complex()-Funktion 35
Computeralgebra 215
cone() (VPython) 240
ConvexHull()-Funktion (Scipy) 305
Corner Detection 322
cos()-Funktion 48
cosh()-Funktion 48
count()-Methode 38, 91
cross()-Methode (SymPy) 222
curve() (VPython) 240
cylinder() (VPython) 238

D

Datenkapselung 154
Datentyp 31, 54, 222
– Float (SymPy) 222
– Rational (SymPy) 222
def 58
Default-Werte 116
Delaunay()-Funktion (Scipy) 303
Delaunay-Triangulierung 303
Destruktor 158
det()-Methode (SymPy) 221
Diagramm
– Balken- 200
– Histogramm 204

- Polar- 203
- Torten- 202
- XY- 195
- Dictionaries 103
- Differentiation (SymPy) 216
- Differenzmenge 101
- diff()-Funktion (SymPy) 216
- Distributionen 271
- dot()-Funktion (Numpy) 187, 189
- dot()-Methode (SymPy) 222
- dtype
 - Numpy-Attribut 185

E

- Ecken ermitteln 322
- Eingabeaufforderung 4
- elif-Klausel 71
- ellipsoid() (VPython) 240
- else-Klausel 70
- endswith()-Methode 39
- enumerate()-Funktion 304
- Ereignis 252
- EVA-Prinzip 14
- exit()-Funktion (VPython) 254
- expand()-Methode (SymPy) 223
- exp()-Funktion 48
- eye()-Methode (SymPy) 221

F

- fft()-Funktion (Scipy) 293
- FFT (Scipy) 293
- Figure 205
- find()-Methode 39
- float()-Funktion 16, 34
- Formatelement 16, 67
- Funktion 56, 116
 - interpolierend 284
 - mit Rückgabewert 57
 - ohne Rückgabewert 60
- Funktionsaufrufe 11

G

- Ganzzahl 5, 31
- Ganzzahl-Division 5
- Geheimnisprinzip 154
- Geometrie-Elemente (SymPy) 224
- Gleitpunkt-Division 5
- Gleitpunktzahl 5, 33
- Graustufenbild 319, 320
- grid()-Methode (Matplotlib) 197

H

- has_key()-Methode 105
- helix() (VPython) 241
- help()-Funktion 48
- hex()-Funktion 32
- Hilfefunktion 48
- hist()-Methode (Matplotlib) 204
- Hough-Transformation 324

I

- id()-Funktion 153
- IDLE 3
- if-Anweisung
 - einfache 68
 - erweitert 70
- ifft()-Funktion (Scipy) 293
- import-Anweisung 8
- Indentation 59
- index()-Methode für Listen 91
- information hiding 154
- inheritance 165
- inner()-Funktion (Numpy) 188
- input()-Funktion 15, 41, 66
- insert()-Methode 91
- Instanz 151
- Integral
 - bestimmt (Scipy) 282
 - bestimmt (SymPy) 218
 - unbestimmt (SymPy) 218
 - uneigentlich (Scipy) 284
 - uneigentlich (SymPy) 219
- integrate()-Funktion (SymPy) 217

Integration (SymPy) 217
interp1d()-Funktion (Scipy) 285
Interpolation (Scipy) 284
Interpreter 3
int()-Funktion 32
inv()-Methode (SymPy) 221
isalnum()-Methode 40
isalpha()-Methode 40
isdigit()-Methode 40
islower()-Methode 40
isupper()-Methode 40
items()-Methode 104
Iteratoren 114
iter()-Funktion 115

K

Kanten detektieren 323
keys()-Methode 104
Klasse 151
Klassenattribute 153
Kommentar 15
Komplexe Zahlen 34
Konstruktor 158
konvexe Hülle (Scipy) 304
Kreise erkennen 324

L

len()-Funktion 37
len()-Funktion für Listen 90
linalg.solve()-Funktion (Numpy) 189
line feed 127
linspace()-Funktion (Numpy) 186
List Comprehensions 113
Listen 89
ljust()-Methode 38
log10()-Funktion 48
log()-Funktion 48
Logischer Ausdruck 7
lower()-Methode 40

M

Masken 188
Mathematische Funktion 7, 48
Matplotlib 195
Matrizen 185
Matrizenrechnung (SymPy) 220
Maus-Interaktion (VPython) 255 ff.
max()-Funktion 93
Mengen 101
Menu (VPython) 250
Methode 38, 153
min()-Funktion 93
Miniconda3 274
minimize()-Funktion (Scipy) 290
Modul 8, 47, 117
– cmath 49
– math 8, 47
– numpy 183
– pylab 196
– scipy 281
– scipy.fftpack 293
– scipy.optimize 287
– sympy 215
– turtle 10
– vpython 231
Modulo-Operator 5

N

Namensraum 121
Nebenbedingung 289
newton()-Funktion (Scipy) 287
Newton-Verfahren 287
next()-Methode 115
Nullstellenberechnung 287
numerische Integration 297
Numpy 183

O

Oberklasse 166
object 152
Objekt 31, 151
oct()-Funktion 32

odeint()-Funktion (Scipy) 298
 ones()-Funktion (Numpy) 186
 ones()-Funktion (SymPy) 221
 open()-Funktion 126
 Operator 4, 46
 – Berechnungs- 46
 – für Listen 93
 – logischer 7, 74
 – Vergleichs- 74
 – Zuweisungs- 55 f.
 operator overloading 163
 Optimierung (Scipy) 289

P

Parameter
 – optionale 116
 – Pflicht- 117
 – Schlüsselwort- 116
 – Übergabe- 58, 116
 pie()-Methode (Matplotlib) 202
 plot()-Methode (Matplotlib) 196
 points() (VPython) 242
 polar()-Methode (Matplotlib) 203
 Potenzreihe (SymPy) 220
 pow()-Funktion 48
 pprint()-Funktion (SymPy) 220
 print()-Funktion 9, 66
 Programmbibliothek 271
 Programmschleife 75
 Programmverzweigung 68
 Punkt-Operator 153
 pyramid() (VPython) 241

Q

quad()-Funktion (Scipy) 283
 Quad (VPython) 244

R

radians()-Funktion 23
 RAD – Rapid Application Development
 1
 random.randn()-Funktion (Numpy) 190

range()-Funktion 77
 rate()-Funktion (VPython) 259
 raw_input()-Funktion 43
 readlines()-Funktion 126
 Rekursion 119
 remove()-Methode 91
 replace()-Methode 39
 reverse()-Methode 91
 ring() (VPython) 242
 rjust()-Methode 38

S

Schieberegler (VPython) 250
 Schleife
 – Endlos- 78
 – for- 76
 – while- 80
 Schlüsselwert 103
 Schlüssel-Wert-Paar 103
 Schlüsselwort 52
 Schnelle Fourier Transformation (FFT)
 293
 Schnittmenge 101
 Schnittstellen 154
 scikit-image 317
 Scipy 281
 select()-Methode (VPython) 236
 Sequenzielle Objekttypen 37, 110
 series()-Methode (SymPy) 220
 Sets 101
 set_title()-Methode (Matplotlib)
 206
 shape (Numpy-Attribut) 185
 Shell 3
 show()-Methode (Matplotlib) 197
 sin()-Funktion 48
 sinh()-Funktion 48
 Slicing 110
 Slider (VPython) 250
 sort()-Methode 91
 Splinefunktion 286
 split()-Methode 40
 Sprachversion 271
 – Unterschiede 272

Spyder 275
sqrt()-Funktion 48
Stack 94
Standardklasse 166
Stapel 94
startswith()-Methode 39
Steuerelemente 248
STL-Datei 245
– grafisch darstellen 245
– lesen 245
str()-Funktion 38
String 10
Subklasse. Siehe Unterklasse
Subplots 205
subs()-Methode (SymPy) 224
Superklasse. Siehe Oberklasse
sympify()-Funktion (SymPy) 224
SymPy 215
Szene (VPython) 232

T

Tabulatorzeichen 127
tan()-Funktion 48
tanh()-Funktion 48
Tastatureingaben
– mit Callback-Funktion 254
– mit Endlosschleife 253
Template-Matching 327
Textdatei
– lesen 125
– schreiben 129
text()-Methode (Matplotlib) 209
title()-Methode (Matplotlib) 202
together()-Funktion (SymPy) 224
Triangle (VPython) 243
Triangulierung 243
triplot()-Methode (Scipy) 303
Tupel 100
Turtle-Grafik 10, 123
type()-Funktion 31

U

Überladen 161
Überladen von Operatoren
– Methoden 165
Unterklasse 166
upper()-Methode 40

V

values()-Methode 104
Variable 9, 51
– globale 121
– lokale 121
Variablenname 9, 26, 53
Verbund 150
Vereinigungsmenge 101
Vererbung 165
Vergleichsausdruck 6
Visual-Python 231
VPython 231

W

waitfor()-Methode (VPython) 253
Warteschlange (queue) 170
Widgets 248
Wiederholungsanweisung 75
WinPython 276
write()-Funktion 129
Wtext (VPython) 249

X

xlabel()-Methode (Matplotlib) 197
xtitle()-Methode (Matplotlib) 197

Y

ylabel()-Methode (Matplotlib) 197

Z

Zeichenkette 10, 36
Zeitreihen 184

zeros()-Funktion (Numpy) 186
zeros()-Methode (Sympy) 221
Zielfunktion 289

zip()-Funktion 115
Zufallswerte 190
Zuweisung 9, 51, 53