

Michael Weigend

PYTHON 3

FÜR STUDIUM UND AUSBILDUNG

Einfach lernen
und professionell anwenden



Hinweis des Verlages zum Urheberrecht und Digitalen Rechtemanagement (DRM)

Liebe Leserinnen und Leser,

dieses E-Book, einschließlich aller seiner Teile, ist urheberrechtlich geschützt. Mit dem Kauf räumen wir Ihnen das Recht ein, die Inhalte im Rahmen des geltenden Urheberrechts zu nutzen. Jede Verwertung außerhalb dieser Grenzen ist ohne unsere Zustimmung unzulässig und strafbar. Das gilt besonders für Vervielfältigungen, Übersetzungen sowie Einspeicherung und Verarbeitung in elektronischen Systemen.

Je nachdem wo Sie Ihr E-Book gekauft haben, kann dieser Shop das E-Book vor Missbrauch durch ein digitales Rechtemanagement schützen. Häufig erfolgt dies in Form eines nicht sichtbaren digitalen Wasserzeichens, das dann individuell pro Nutzer signiert ist. Angaben zu diesem DRM finden Sie auf den Seiten der jeweiligen Anbieter.

Beim Kauf des E-Books in unserem Verlagsshop ist Ihr E-Book DRM-frei.

Viele Grüße und viel Spaß beim Lesen,

Ihr mitp-Verlagsteam



Michael Weigend

Python 3 für Studium und Ausbildung

Einfach lernen und professionell anwenden



Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <<http://dnb.d-nb.de>> abrufbar.

ISBN 978-3-7475-0435-2

1. Auflage 2022

www.mitp.de

E-Mail: mitp-verlag@sigloch.de

Telefon: +49 7953 / 7189 - 079

Telefax: +49 7953 / 7189 - 082

© 2022 mitp Verlags GmbH & Co. KG, Frechen

Dieses Werk, einschließlich aller seiner Teile, ist urheberrechtlich geschützt. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung des Verlags unzulässig und strafbar. Dies gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen und die Einspeicherung und Verarbeitung in elektronischen Systemen.

Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.

Lektorat: Janina Bahlmann

Sprachkorrektorat: Christine Hofmeister

Covergestaltung: Christian Kalkert

Bildnachweis: © Gstudio / stock.adobe.com, © ylivdesign / stock.adobe.com

Kastenicons: Tanja Wehr, sketchnotelovers

Abbildungen & Grafiken: Michael Weigend

Satz: Petra Kleinwegen

Für Lars Jonah

Inhaltsverzeichnis

Einleitung	17
Python in Studium und Ausbildung	17
Der Aufbau des Buchs	17
Achten Sie auf den Schrifttyp!	18
Programmtexte und Lösungen zum Download	19
1 Willkommen zu Python!	21
1.1 Die Programmiersprache Python	21
1.2 Was ist ein Algorithmus?	22
1.3 Syntax und Semantik	22
1.4 Interpreter und Compiler	23
1.5 Python installieren	23
1.6 Python im interaktiven Modus	25
1.7 Die Entwicklungsumgebung IDLE	26
1.8 Hotkeys für die IDLE-Shell	27
1.9 Anweisungen	27
1.9.1 Ausdruck	27
1.9.2 Funktionsaufruf	28
1.9.3 Zuweisung	28
1.9.4 Erweiterte Zuweisungen	31
1.10 Zahlen verarbeiten – Python als Taschenrechner	32
1.10.1 Operatoren	32
1.10.2 Variablen verwenden	34
1.11 Eine weitere Entwicklungsumgebung: Thonny	34
1.12 Notebooks mit Jupyter und CoLab	36
1.13 Rückblick	36
1.14 Übungen	37
1.15 Lösung der Frage: Semantik im Alltag	38
2 Datentypen – die Python-Typ-Hierarchie	39
2.1 Literale und die Funktion type()	39
2.2 Die Python-Typ-Hierarchie	40

2.3	Standard-Typen	40
2.3.1	Ganze Zahl (int)	40
2.3.2	Gleitkommazahl (float)	42
2.3.3	Komplexe Zahlen (complex)	42
2.3.4	Zeichenkette (str)	43
2.3.5	Tupel (tuple)	44
2.3.6	Liste (list)	44
2.3.7	Menge (set)	44
2.3.8	Dictionary (dict)	45
2.3.9	Wahrheitswerte – der Datentyp bool	45
2.3.10	NoneType	46
2.4	Gemeinsame Operationen für Kollektionen	46
2.4.1	Kollektion	47
2.4.2	Sequenz	47
2.5	Objekte eines Typs erzeugen – Casting	48
2.6	Dynamische Typisierung	50
2.7	Rückblick	50
2.8	Übung: Anweisungen	51
3	Interaktive Programme	53
3.1	Das erste Python-Skript	53
3.2	Das EVA-Prinzip	55
3.3	Kommentare	57
3.4	Projekt: Volumenberechnung	58
3.4.1	Kürzerer Programmtext durch verschachtelte Funktionsaufrufe	59
3.4.2	Aufruf der Funktion print() mit mehreren Argumenten	60
3.5	Python-Programme starten	60
3.5.1	Ausführung auf der Kommandozeile	61
3.5.2	Start durch Anklicken des Programm-Icons unter Windows	62
3.5.3	Python-Programme unter Linux – die Shebang-Zeile	64
3.5.4	Starten im Finder von macOS	64
3.6	Fehler finden	64
3.6.1	Syntaxfehler	64
3.6.2	Laufzeitfehler	65
3.6.3	Semantische Fehler	66
3.6.4	Tipps zum Fehlerfinden	66
3.7	Rückblick	67
3.8	Übungen	67
3.9	Lösungen zu den Fragen	69

4	Kontrollstrukturen	71
4.1	Programmverzweigungen	71
4.1.1	Einseitige Verzweigung (if)	71
4.1.2	Projekt: Passwort	72
4.1.3	Zweiseitige Verzweigung (if...else)	73
4.1.4	Projekt: Kinokarte	74
4.1.5	Fallunterscheidung (if...elif...else)	75
4.1.6	Projekt: Auskunftautomat	76
4.2	Das Layout von Python-Programmen: Zeilen und Blöcke	77
4.2.1	Block	77
4.2.2	Zeilenstruktur	77
4.3	Bedingungen konstruieren	78
4.3.1	Boolesche Werte	79
4.3.2	Boolesche Operatoren	79
4.3.3	Vergleichsketten	80
4.3.4	Projekt: Früchte erkennen	80
4.4	Bedingte Wiederholung – while	82
4.4.1	Projekt: Aufsummieren	83
4.4.2	Projekt: Planeten	84
4.4.3	Projekt: Wurzelberechnung (Mathematik)	85
4.4.4	Endloswiederholung	87
4.5	Iterationen – for	87
4.5.1	Wiederholungen mit range()	89
4.6	Rückblick	90
4.7	Übungen	90
4.8	Lösungen zu den Fragen	92
5	Funktionen	93
5.1	Warum definiert man Funktionen?	93
5.2	Definition und Aufruf einer Funktion	93
5.2.1	Projekt: Fallhöhe berechnen	94
5.3	Optionale Parameter und voreingestellte Werte	96
5.3.1	Erweiterung des Projekts: Fallhöhe auf unterschiedlichen Himmelskörpern	96
5.4	Eine Funktion in der Shell testen	97
5.5	Die return-Anweisung	98
5.5.1	Prozeduren	98
5.5.2	Wirkungen der return-Anweisung	98
5.6	Positionsargumente und Schlüsselwortargumente	99

5.7	Guter Programmierstil	101
5.7.1	Funktionsname	101
5.7.2	Funktionsannotationen	101
5.7.3	Docstring	101
5.7.4	Signatur	102
5.8	Die print()-Funktion unter der Lupe	103
5.9	Globale und lokale Namen	104
5.10	Rekursive Funktionen	105
5.10.1	Projekt: Die Berechnung der Fakultät	105
5.11	Lambda-Funktionen *	107
5.12	Funktionen als Argumente: map() und filter() *	108
5.12.1	Mapping	108
5.12.2	Filtern	110
5.13	Rückblick	111
5.14	Übungen	111
5.15	Lösungen zu den Fragen	112
6	Mit Modulen arbeiten	115
6.1	Importanweisungen	115
6.1.1	Ein Modul importieren	115
6.1.2	Funktionen aus einem Modul importieren	116
6.2	Mathematische Funktionen: Das Modul math	117
6.3	Zufallsfunktionen: Das Modul random	118
6.3.1	Projekt: Würfeln	118
6.3.2	Projekt: Wer ist der Nächste?	119
6.4	Datum und Zeit	119
6.4.1	Projekt: Uhrzeit	120
6.4.2	Projekt: Rechentrainer	120
6.5	Ein eigenes Modul erstellen	122
6.5.1	Projekt: Ein Modul zur Volumenberechnung	122
6.5.2	Welchen Vorteil haben Module?	126
6.6	Module aus dem Python Package Index (PyPI)	126
6.7	Rückblick	126
6.8	Übungen	127
7	Mit Kollektionen modellieren	129
7.1	Sequenzen	129
7.1.1	Listen	129
7.1.2	Tupel	130
7.1.3	Komplexe Sequenzen	130
7.1.4	Iteration über eine Liste aus Tupeln	131
7.1.5	Gemeinsame Operationen für Sequenzen	132

7.1.6	Spezielle Operationen für Listen	133
7.1.7	Sortieren	134
7.1.8	Eine Liste erzeugen	135
7.2	Projekt: Telefonliste	137
7.3	Dictionaries	139
7.3.1	Operationen für Dictionaries	140
7.3.2	Ein Dictionary ändern	140
7.4	Projekt: Vokabeltrainer	141
7.5	Projekt: Routenplaner	143
7.5.1	Verkehrswege und Graphen	143
7.5.2	Programmierung	145
7.6	Rückblick	147
7.7	Übungen	147
7.8	Lösungen zu den Fragen	148
8	Daten speichern	151
8.1	Wie werden Daten gespeichert?	151
8.1.1	Dateien öffnen	151
8.1.2	Stream-Methoden	152
8.1.3	Texte speichern und laden	153
8.1.4	Binärdateien und Bytestrings	153
8.1.5	Pfade im Verzeichnisbaum	154
8.2	Laufzeitfehler abfangen	155
8.2.1	try...except	156
8.2.2	try...except...finally	156
8.3	with-Anweisungen	157
8.4	Projekt: Logbuch	158
8.5	Datenstrukturen speichern und laden: Das Modul pickle	160
8.5.1	Speichern	160
8.5.2	Laden	161
8.6	Projekt: Digitaler Planer	162
8.7	Daten im JSON-Format speichern	165
8.7.1	Aufbau eines JSON-Texts	165
8.7.2	Die Grenzen von JSON	167
8.8	Projekt: Temperaturdaten	167
8.8.1	Writer	167
8.8.2	Reader	168
8.9	Daten aus dem Internet	169
8.10	Projekt: Digitale Bibliothek	170
8.11	Rückblick	172
8.12	Übung: News-Check	172
8.13	Lösungen zu den Fragen	173

9	Textverarbeitung	175
9.1	Unicode-Nummern für Zeichen	175
9.2	Escape-Sequenzen	176
9.3	Stringmethoden	177
9.4	Projekt: Goethes Wortschatz	179
9.5	Projekt: Wie warm wird es heute?	180
9.6	Ausblick: Reguläre Ausdrücke *	182
9.6.1	Was ist ein regulärer Ausdruck?	182
9.6.2	Aufbau eines regulären Ausdrucks	183
9.6.3	Textpassagen finden mit findall()	184
9.6.4	Platzhalter für Zeichen aus einer Zeichenmenge	186
9.6.5	Reguläre Ausdrücke verknüpfen	187
9.6.6	Quantoren	187
9.6.7	Sonderzeichen maskieren	188
9.6.8	Gieriges oder nicht gieriges Finden	189
9.6.9	Webscraping mit regulären Ausdrücken	190
9.7	Texte mit variablen Teilen	191
9.7.1	Formatierung	191
9.7.2	Platzhalter mit Namen	192
9.7.3	Formatangaben für Platzhalter	192
9.8	Projekt: Textanalyse	193
9.9	Projekt: Storytelling	195
9.10	Rückblick	196
9.11	Übungen	196
9.12	Lösungen zu den Fragen	198
10	Zugriff auf die Systemumgebung	201
10.1	Schnittstelle zum Betriebssystem: Das Modul os	201
10.2	Suchen und Eigenschaften ermitteln	202
10.2.1	Unterverzeichnisse ausgeben	202
10.2.2	Verzeichnisbaum durchlaufen	203
10.3	Dateien und Verzeichnisse anlegen und umbenennen	206
10.3.1	Projekt: Bilddateien umbenennen	206
10.4	Das Modul sys – die Schnittstelle zum Laufzeitsystem	208
10.4.1	Informationen über die aktuelle Systemumgebung abfragen	208
10.4.2	Kommandozeilenargumente abfragen	209
10.4.3	Blick hinter die Kulissen: Speicherverwaltung *	210
10.4.4	Zugriff auf Module	212
10.4.5	Die Standardausgabe in eine Datei umleiten	213

10.5	Rückblick	214
10.6	Übungen	215
10.7	Lösung zu der Frage: Welcher Kommentar passt?	216
11	Grafische Benutzungsoberflächen	217
11.1	Widgets	217
11.2	Das Anwendungsfenster Tk	218
11.3	Ein Widget einfügen	219
11.4	Das Aussehen der Widgets gestalten	220
	11.4.1 Die Größe eines Widgets	222
11.5	Gemeinsame Methoden der Widgets	222
11.6	Schaltflächen und Eventhandler	223
	11.6.1 Projekt: Motivator	223
11.7	Das Layout verfeinern	224
11.8	Raster-Layout	227
11.9	Projekt: 25 Farben – ein automatisches Farbfelder-Bild	228
11.10	Widgets zur Texteingabe	230
	11.10.1 Einzeilige Eingabe: Das Entry-Widget	230
	11.10.2 Mehrzeilige Eingabe: Das Text-Widget	231
	11.10.3 Projekt: Reimen mit Goethe	233
11.11	Radiobuttons	235
	11.11.1 Projekt: Währungsrechner	236
11.12	Dialogboxen	237
	11.12.1 Projekt: Texteditor	238
11.13	Parallele Abläufe: Threads	239
	11.13.1 Ein Experiment: Countdown	240
	11.13.2 Eine Funktion in einem eigenen Thread ausführen	241
11.14	Rückblick	242
11.15	Übungen	243
11.16	Lösungen zu den Fragen	245
12	Grafik programmieren	247
12.1	Bilder auf Schaltflächen und Labels	247
	12.1.1 Projekt: Würfelspiel	247
	12.1.2 Bilder verändern	249
	12.1.3 Projekt: Graustufen	250
12.2	Canvas	252
	12.2.1 Flächen gestalten	252
	12.2.2 Linien gestalten	254
	12.2.3 ID-Nummern: Elemente löschen oder bewegen	254
12.3	Projekt: Creative Coding	255

12.4	Die Python Imaging Library (PIL)	257
12.4.1	Ein Image-Objekt aus einer Datei gewinnen	258
12.4.2	Ein Image-Objekt ohne Datei erzeugen	259
12.4.3	Attribute und Methoden von Image-Objekten	259
12.4.4	Bilder über Listen verarbeiten	261
12.4.5	Bilder einfügen	263
12.4.6	Projekt: Greenscreen	263
12.4.7	PIL.Image-Objekte in tkinter-Anwendungen	265
12.4.8	Projekt: Webcam-Viewer	266
12.5	Rückblick	267
12.6	Übungen	268
13	Fehler finden und vermeiden	271
13.1	Zusicherungen	271
13.2	Tracing	273
13.2.1	Beispiel: Quicksort	273
13.3	Debugging mit IDLE	275
13.3.1	Der Debugger der Python-Shell	275
13.3.2	Das Programm schrittweise durchlaufen	276
13.3.3	Haltepunkte setzen	277
13.4	Debugging mit Thonny	278
13.5	Rückblick	280
13.6	Lösungen zu den Fragen	281
14	Objektorientierte Programmierung	283
14.1	Klassen und Objekte	283
14.1.1	Was ist Objektorientierung?	283
14.1.2	Klassen entwerfen und grafisch darstellen – UML	284
14.1.3	Definition einer Klasse	285
14.1.4	Objekte einer Klasse erzeugen: Instanziierung	286
14.1.5	Auf Attribute zugreifen	286
14.1.6	Methoden aufrufen	287
14.1.7	Objekte mit variablen Anfangswerten	287
14.1.8	Metaphern in der Programmierung	288
14.2	Projekt: Geld	288
14.2.1	Mit Geld-Objekten rechnen	289
14.2.2	Klassenattribute	290
14.2.3	Operatoren überladen – Polymorphie	290
14.3	Magische Methoden	293
14.4	Projekt: Abrechnung	294

14.5	Vererbung	296
14.6	Projekt: Farbtester	298
14.7	Projekt: Zahlenregen	301
14.8	Rückblick	305
14.9	Übungen	305
14.10	Lösungen zu den Fragen	308
15	Datenbanktechnik	309
15.1	Was ist ein Datenbanksystem?	309
15.2	Eine Datenbank entwerfen – das Entity-Relationship-Diagramm (ER)	309
15.3	Relationale Datenbanken	311
15.4	Relationen mit Python darstellen *	313
	15.4.1 Menge von Tupeln	313
	15.4.2 Benannte Tupel (named tuples)	313
15.5	Das Modul sqlite3 – Schnittstelle zu einer SQL-Datenbank	314
	15.5.1 Mit SQL Tabellen anlegen und Tupel eintragen	315
	15.5.2 Mit sqlite3 eine SQLite-Datenbank aufbauen	316
	15.5.3 Formulierung von Anfragen (Queries) mit SQL	317
	15.5.4 Datenbankanfragen in Python-Programmen	318
	15.5.5 SQL-Anweisungen mit variablen Teilen	320
15.6	Projekt: Zitatesammlung	320
	15.6.1 ER-Diagramm	321
	15.6.2 Tabellen (Beispiel)	321
	15.6.3 Administration der Zitatesammlung	322
	15.6.4 Nach Zitaten suchen	324
15.7	Rückblick	327
15.8	Übungen	328
15.9	Lösungen zu den Fragen	329
16	Wissenschaftliche Projekte	331
16.1	NumPy – Rechnen mit Arrays	331
	16.1.1 Arrays	331
	16.1.2 Indizieren	336
	16.1.3 Slicing	337
	16.1.4 Arrays verändern	338
	16.1.5 Arithmetische Operationen	340
	16.1.6 Funktionen, die elementweise ausgeführt werden	341
	16.1.7 Matrizenmultiplikation mit dot()	341
	16.1.8 Array-Funktionen und Achsen	342

16.2	Datenvisualisierung mit matplotlib	343
16.2.1	Liniendiagramme	344
16.2.2	Mehrere Linien in einem Diagramm	346
16.2.3	Histogramme	347
16.2.4	Projekt: Würfeln	348
16.2.5	Heatmaps	349
16.3	Projekt: Bewegungsprofil	350
16.4	Google Colaboratory – Colab	354
16.4.1	Ein Colab-Notebook erzeugen	354
16.4.2	Text-Zellen	356
16.4.3	Bilder einfügen	358
16.4.4	Notebooks speichern und öffnen	359
16.5	Projekt: Füchse und Hasen – Simulation eines Räuber-Beute- Systems	360
16.5.1	Notebooks teilen	363
16.6	Rückblick	364
16.7	Übungen	365
16.8	Lösungen zu den Fragen	367
17	Dynamische Webseiten: CGI und WSGI	369
17.1	Dynamische Webseiten mit CGI	370
17.1.1	Projekt: Wie spät ist es?	371
17.1.2	Die Ausgabe eines CGI-Skripts	372
17.1.3	Wie ist ein CGI-Skript aufgebaut?	372
17.1.4	CGI-Skripte unter Windows	373
17.1.5	Aufruf mit dem Webbrowser	373
17.1.6	Ein HTTP-Server	374
17.1.7	Zugriff von einem anderen Computer im lokalen Netz	375
17.2	Interaktive Webseiten	375
17.2.1	Eingabekomponenten in einem HTML-Formular	377
17.2.2	Verarbeitung von Eingabedaten mit FieldStorage	379
17.3	Wie verarbeitet man Umlaute? *	380
17.4	Dynamische Webseiten mit WSGI	382
17.4.1	Das Applikationsobjekt	382
17.4.2	Skripte mit eigenem HTTP-Server – das Modul wsgiref ...	383
17.5	Projekt: Wie spät ist es? (II)	383
17.6	Projekt: Umfrage	386
17.6.1	Die HTML-Schablonen	387
17.6.2	Der algorithmische Teil	388

17.7	Einen Hosting-Dienst nutzen	390
17.7.1	Python Anywhere	390
17.7.2	Das vorgefertigte OWSGI-Programm ausprobieren	390
17.7.3	Projekt: Wie spät ist es? (III)	393
17.7.4	WSGI-Projekte modularisieren	394
17.8	Rückblick	395
17.9	Übungen	395
17.10	Lösung zur Frage: Interaktive Webseite	397
18	Professionelle Software-Entwicklung	399
18.1	Die Laufzeit von Programmen	399
18.1.1	Schnelles Sortieren – Quicksort versus Straight Selection .	399
18.1.2	Performance-Analyse mit dem Profiler	402
18.2	Agile Software-Entwicklung	403
18.2.1	Software Engineering	403
18.2.2	Einige Grundideen der agilen Software-Entwicklung	404
18.3	Projekt: Digitales Notizbuch	405
18.3.1	Stories	405
18.3.2	Erste Iteration	406
18.3.3	Zweite Iterationen	407
18.3.4	Refactoring	408
18.3.5	Neue Stories und Änderbarkeit	412
18.4	Test Driven Development mit doctest	413
18.5	Übung: Ticketbuchung	415
19	Glossar	417
20	Stichwortverzeichnis	425



Einleitung

Python in Studium und Ausbildung

In vielen Berufen – auch außerhalb der Informationstechnik – werden heute Programmierkenntnisse als Basiskompetenz vorausgesetzt. Selbst wenn Ihr Schwerpunkt nicht die professionelle Softwareentwicklung ist, werden Sie in Rahmen von wissenschaftlichen Projekten oder in der Berufspraxis Computerprogramme schreiben oder an Entwicklungen beteiligt sein. Darüber hinaus schult das Programmieren das logische Denken. Wer programmieren kann, ist besser in der Lage, Probleme zu analysieren und Lösungen zu finden.

Dieses Buch wendet sich vor allem an Menschen, die im Rahmen eines Studiums oder einer beruflichen Ausbildung einen Einstieg in die Programmierung mit Python suchen. Es lässt sich sowohl als Materialgrundlage für einen Programmierkurs als auch für das eigenständige Lernen einsetzen.

Alle Erklärungen sind leicht verständlich formuliert und setzen keine Vorkenntnisse voraus. Am besten lesen Sie das Buch neben der Computer-Tastatur und probieren die Programmbeispiele gleich aus. Zahlreiche praktische Programmier-Übungen helfen Ihnen, Ihr neues Wissen zu verinnerlichen. Sie werden schnell erste Erfolge erzielen und Freude an der Programmierung finden.

Der Aufbau des Buchs

Das Buch beginnt mit den Grundlagen: Installation von Python, Nutzung der Entwicklungsumgebung und Formulierung einfacher Anweisungen. Sie lernen Schritt für Schritt, wie man Daten lädt, verarbeitet und speichert, und erhalten eine Einführung in die Verwendung von Funktionen und Modulen, objektorientierte Programmierung und die Gestaltung von grafischen Benutzungsoberflächen.

In den hinteren Kapiteln wenden Sie die gelernten Python-Konzepte in wichtigen und spannenden Gebieten der Informatik an: Datenbanktechnik, Bildverarbeitung, wissenschaftliches Rechnen mit NumPy, Visualisierung von Daten mit Matplotlib und Internetprogrammierung.

Abschnitte, die mit einem Sternchen * versehen sind, können Sie überspringen, wenn Sie das Thema nicht interessiert. Sie behandeln sehr spezielle Inhalte, die für das Verständnis des nachfolgenden Texts nicht benötigt werden.

Das letzte Kapitel schließlich gibt einen Einblick in fortgeschrittene Techniken (z.B. das Aufspüren von Schwachstellen im Programm mit einer Performance-Analyse) und zeigt

Ihnen einige Ideen des agilen Programmierens, die helfen können, ein größeres Softwareprojekt erfolgreich zu planen und durchzuführen.

Gelegentlich stoßen Sie auf Zwischenfragen. Sie sind als kleine Lernaktivierung gedacht und werden am Ende des Kapitels beantwortet. Jedes Kapitel schließt mit praktischen Programmier-Übungen, in denen Sie Ihr neu gewonnenes Wissen vertiefen können. Mit Sternchen * wird der Schwierigkeitsgrad der Aufgaben gekennzeichnet. Je mehr Sternchen, desto schwieriger. Die Lösungen zu diesen Übungen, die meist viel Programmtext enthalten, stehen in einem Online-Kapitel zum Download zur Verfügung. Mehr dazu im übernächsten Abschnitt.

Am Ende des Buchs finden Sie ein Glossar mit den wichtigsten Fachbegriffen sowie ein Stichwortverzeichnis, das Ihnen hilft, bestimmte Themen im Buch schneller zu finden.

Achten Sie auf den Schrifttyp!

In diesem Buch hat der Schrifttyp eine Bedeutung. Das soll das Lesen erleichtern. Alle Programmtexte oder Teile von Programmtexten (wie z.B. Variablennamen) sind in einer nicht proportionalen Schrift (Monotype-Schrift) gesetzt.

Beispiel: Die Variable `name` hat den Wert `'Jessy'`.

In einigen Passagen der Programmtexte kommen *kursiv* gesetzte Monotype-Texte vor, die als Platzhalter gemeint sind. In einem Programm würde man den Platzhalter durch einen anderen, in den Zusammenhang passenden Text ersetzen.

Beispiel: Bei

```
stream = open(dateiname)
```

sind *stream* und *dateiname* Platzhalter.

In Textpassagen, die einen Dialog mit dem Computer wiedergeben, ist der Text, den ein Mensch eingegeben hat, etwas heller gesetzt als der Text, den der Computer ausgibt.

Beispiel:

```
Dein Name: Helena  
Guten Morgen Helena!
```

Programmtexte und Lösungen zum Download

Das Buch enthält viele kleine Beispielprogramme. Sie sind als »Starterprojekte« gedacht und sollen Sie ermuntern, den Code weiterzuentwickeln und eigene Ideen umzusetzen.

Alle Programmtexte sowie die Lösungen zu den Übungen stehen Ihnen auf der Webseite des Verlags unter <https://www.mitp.de/0434> zum Download zur Verfügung. Dort finden Sie gegebenenfalls auch eine Errata-Liste. Wenn Sie einen Fehler finden, würde ich mich über eine Rückmeldung freuen. Am besten schreiben Sie eine E-Mail an Tektorat@mitp.de.

Ich wünsche Ihnen viel Erfolg und Spaß bei der Programmierung mit Python!

Michael Weigend

Willkommen zu Python!

Dieses Kapitel hilft Ihnen bei den ersten Schritten im Umgang mit einer der erfolgreichsten und faszinierendsten Programmiersprachen unserer Zeit. Python ist erfolgreich, weil es in praktisch allen Wissensbereichen eingesetzt wird: Naturwissenschaft, Technik, Mathematik, Musik und Kunst. Viele Menschen finden Python faszinierend, weil das Programmieren mit Python das Denken beflügelt. Mit Python können Sie digitale Modelle entwickeln und Problemlösungen elegant und verständlich formulieren.

Nach einer kurzen Einführung in einige wichtige Grundbegriffe der Informatik erfahren Sie, wie man Python installiert. Sie arbeiten praktisch an der Tastatur, probieren Anweisungen aus und lernen dabei, was Ausdrücke, Zuweisungen und Variablen sind.

1.1 Die Programmiersprache Python

Im Unterschied zu »natürlichen« Sprachen wie Deutsch oder Englisch, die sich über Jahrhunderte entwickelt haben, sind Programmiersprachen »künstliche« Sprachen. Sie wurden von Fachleuten designt und sind speziell auf die Formulierung von Algorithmen zugeschnitten.

Die ersten höheren Programmiersprachen (z.B. Fortran und Lisp) wurden in den 1950er Jahren entwickelt. Heute (27. Januar 2022) listet Wikipedia 374 Programmiersprachen auf.

Die erste Python-Version wurde 1991 von dem niederländischen Informatiker Guido van Rossum veröffentlicht. Der Name der Sprache soll an die englische Comedy-Gruppe *Monty Python* erinnern. Seit 2001 wird Python von der Python Software Foundation (PSF) gepflegt, kontrolliert und verbreitet (www.python.org).

Viele digitale Produkte, die Sie aus dem Alltag kennen, basieren auf Python, z.B. Google Maps, YouTube und Instagram. Im PYPL-Index (Popularity of Programming Language Index) wird die Beliebtheit einer Programmiersprache danach gemessen, wie oft bei Google nach einem Sprach-Tutorial gesucht wird. Demnach ist Python (im Jahre 2022) mit Abstand die populärste Programmiersprache.

Warum ist Python unter Programmierern so beliebt?

- Mit Python kann man sehr kurze Programmtexte schreiben. Das verbessert die Verständlichkeit eines Programms, erleichtert die Fehlersuche und verkürzt die Entwicklungszeit.
- Python ist leicht zu lernen, da vertraute Schreibweisen verwendet werden, die man z.B. schon aus der Mathematik kennt.
- Python unterstützt unterschiedliche Programmierstile (»Paradigmen«).

- Zu Python gibt es viele frei verfügbare Erweiterungen (sogenannte *Module*) für spezielle Anwendungsbereiche wie etwa Grafik, Astronomie, Mathematik, Spracherkennung, Quantencomputer und künstliche Intelligenz.

1.2 Was ist ein Algorithmus?

In der Informatik versteht man unter einem Algorithmus eine *präzise Anleitung zur Lösung einer Aufgabe*. Ein Algorithmus besteht aus einer Folge von einzelnen *Anweisungen*, die so genau und eindeutig formuliert sind, dass sie auch von einem völlig Unkundigen rein mechanisch ausgeführt werden können. Algorithmen, die man aus dem Alltag kennt, sind z.B.

- ein Kochrezept,
- eine Anleitung zum Zusammenbau eines Regals,
- eine Gebrauchsanweisung.

Ein Computerprogramm ist ein Algorithmus, der in einer Programmiersprache geschrieben worden ist und von einem Computer »verstanden« und ausgeführt werden kann.

1.3 Syntax und Semantik

Eine Programmiersprache ist – wie jede Sprache – durch Syntax und Semantik definiert. Die *Syntax* legt fest, welche Folgen von Zeichen ein gültiger Programmtext in der jeweiligen Sprache sind.

Zum Beispiel ist

```
print['Hallo']
```

kein gültiger Python-Programmtext, weil die Python-Syntax vorschreibt, dass nach dem Wort `print` eine runde Klammer folgen muss.

Dagegen ist die Zeichenfolge

```
print('Hallo')
```

ein syntaktisch korrektes Python-Programm. Die Syntax sagt aber nichts darüber aus, welche *Wirkung* dieses Mini-Programm hat. Die Bedeutung eines Programmtextes wird in der *Semantik* definiert. Bei diesem Beispiel besagt die Semantik, dass auf dem Bildschirm das Wort `Hallo` ausgegeben wird.

Bei einem Programmtext ist die Semantik *eindeutig*. Dagegen kann ein Text in einer natürlichen Sprache mehrdeutig sein.

Frage: Semantik im Alltag

Inwiefern ist der Satz »Schau nach vorne!« semantisch nicht eindeutig?

1.4 Interpreter und Compiler

Python ist eine sogenannte *höhere* Programmiersprache. Das bedeutet, dass Besonderheiten des Computers, auf dem das Programm laufen soll, nicht beachtet werden müssen. Ein Python-Programm läuft praktisch auf jedem Computer und unter jedem gängigen Betriebssystem. Eine höhere Programmiersprache ist für Menschen gemacht und ermöglicht es, gut verständliche Programmtexte zu schreiben.

Einen Programmtext, der in einer höheren Programmiersprache geschrieben ist, nennt man *Quelltext* (auf Englisch *source code*). Damit der Quelltext vom Computer abgearbeitet werden kann, muss er in eine »maschinennahe Sprache« übersetzt werden. Dazu gibt es zwei unterschiedliche Methoden:

- Ein *Compiler* übersetzt einen kompletten Programmtext und erzeugt eine direkt ausführbare (*executable*) Programmdatei, die vom Betriebssystem geladen und gestartet werden kann.
- Ein *Interpreter* liest jede Anweisung eines Programmtextes einzeln und führt sie über das Betriebssystem direkt aus. Wenn ein Programm gestartet werden soll, muss zuerst der Interpreter aufgerufen werden.

Python ist eine interpretative Programmiersprache. Das hat den Vorteil, dass ein Python-Programm auf jeder Plattform funktioniert. Voraussetzung ist allerdings, dass auf dem Computer ein Python-Interpreter installiert ist. Das Betriebssystem allein ist nicht in der Lage, das Python-Programm auszuführen.

1.5 Python installieren

Python ist völlig kostenlos und wird für Microsoft Windows, Linux/Unix und macOS angeboten.

Sämtliche Software, die Sie für die Arbeit mit Python benötigen, ist frei und kann von der Python-Homepage <http://www.python.org/download> heruntergeladen werden. Dieses Buch bezieht sich auf Version 3.10.1, die im Dezember 2021 herauskam. Falls Sie eine neuere Version installieren, werden aber dennoch alle Programme, die in diesem Buch beschrieben werden, funktionieren.

Windows

Auf der Download-Seite <http://www.python.org/download> werden Installationsdateien angeboten, die zu Ihrem System passen.

Klicken Sie auf die Schaltfläche oben links mit der aktuellen Version von Python 3.



Abb. 1.1: Download-Seite von Python

Laden Sie das Installationsprogramm herunter und starten Sie es. Achten Sie darauf, dass im Rahmen der Installation das Verzeichnis mit dem Python-Interpreter dem Systempfad (PATH) hinzugefügt wird (siehe Abbildung 1.2). Damit ist sichergestellt, dass das Betriebssystem den Python-Interpreter findet, wenn Sie im Konsolenfenster (Eingabeaufforderung) den Befehl `python` eingeben. Schließlich klicken Sie auf **INSTALL NOW**.

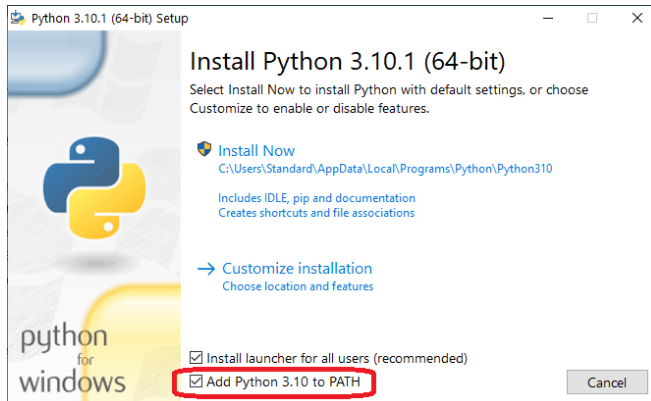


Abb. 1.2: Installation von Python unter Windows

Linux

Auf Linux-Systemen ist Python in der Regel bereits installiert. Prüfen Sie, welche Version vorliegt, indem Sie in einem Konsolenfenster auf der Kommandozeile den Befehl `python -V` eingeben.

```
$ python -V
Python 3.10.1
```

Wenn Sie keine Version von Python 3 vorfinden, müssen Sie sie nachinstallieren. Verwenden Sie am besten das Advanced Packaging Tool (APT):

```
$ sudo apt-get install python3.10
```

macOS

Wie auf Linux-Systemen ist auch auf Apple-Computern Python in der Regel bereits installiert. Um das nachzuprüfen, öffnen Sie auf Ihrem Mac ein Terminal-Fenster (PROGRAMME|DIENSTPROGRAMME|TERMINAL) und geben folgenden Befehl ein:

```
python -V
```

Wenn Sie keine Version von Python 3 vorfinden, besuchen Sie die Python-Website, laden eine zu Ihrem System passende Installer-Datei herunter und führen sie aus.

1.6 Python im interaktiven Modus

Wenn Sie Python heruntergeladen und installiert haben, befinden sich auf Ihrem Computer folgende Komponenten:

- der Python Interpreter,
- die Entwicklungsumgebung IDLE (Integrated Development and Learning Environment),
- eine ausführliche Dokumentation,
- Hilfsprogramme.

Sie können den Python-Interpreter in einer Konsole (Shell) direkt aufrufen, um dann einzelne Python-Befehle auszuprobieren. Auf einem Windows-Rechner öffnen Sie eine Konsole z.B. auf folgende Weise: Geben Sie im Suchfeld unten links den Befehl `cmd` ein und drücken Sie die Taste `Enter`. Es erscheint ein Anwendungsfenster mit dem Titel EINGABEAUFFORDERUNG ungefähr wie in Abbildung 1.3. Auf einem Mac heißt die Konsole TERMINAL. Drücken Sie gleichzeitig die Befehlstaste und die Leertaste, um Spotlight zu starten, und geben Sie `Terminal` ein.

Eine Konsole enthält die sogenannte *Kommandozeile*, die mit dem Prompt des Betriebssystems endet. Bei Windows ist der Prompt das Zeichen `>`, bei Linux und macOS `$`.

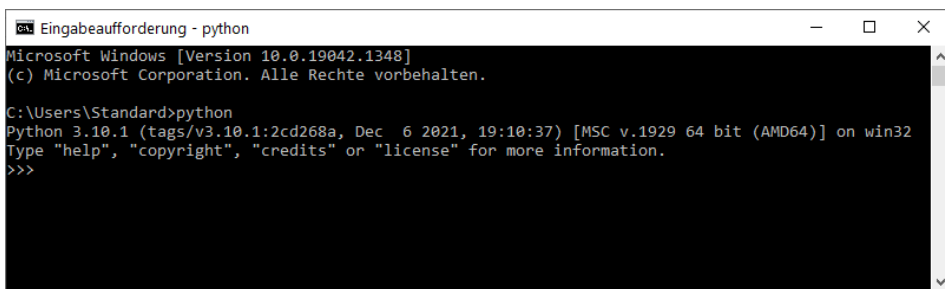


Abb. 1.3: Aufruf des Python-Interpreters in einem Konsole-Fenster (Eingabeaufforderung) unter Windows

Hinter dem Prompt des Betriebssystems geben Sie den Befehl

```
python
```

Kapitel 1

Willkommen zu Python!

ein und drücken die Taste `Enter`. (Achten Sie auf das kleine `p` zu Beginn.) Damit wird der Python-Interpreter im »interaktiven Modus« gestartet. Unter einem Begrüßungstext sehen Sie diesen Prompt:

```
>>>
```

Im interaktiven Modus führen Sie eine Art »Gespräch« mit dem Python-Interpreter. Hinter dem Prompt geben Sie eine einzelne Python-Anweisung ein. Sobald Sie `Enter` drücken, führt der Interpreter die Anweisung aus und liefert in der nächsten Zeile ein Ergebnis – sofern die Anweisung ein Ergebnis berechnet. Im Englischen nennt man dieses Prinzip *Read-Eval-Print-Loop* oder kurz *REPL*.

Auch arithmetische Ausdrücke sind gültige Python-Anweisungen. Probieren Sie es aus:

```
>>> 2 + 2
4
>>> (2 + 2) * 4
16
>>>
```

Sie beenden den Python-Interpreter mit der Tastenkombination `Strg` + `C`.

1.7 Die Entwicklungsumgebung IDLE

IDLE (Integrated Development and Learning Environment) ist die Standard-Entwicklungsumgebung für Python. Eine Entwicklungsumgebung ist eine Software, die Programmierer benutzen, wenn sie Programme entwickeln. IDLE besteht aus der *IDLE-Shell* und einem *Editor*:

- In der IDLE-Shell verwenden Sie Python im interaktiven Modus. Die Python-Shell nutzen Sie, um Anweisungen auszuprobieren und ihre Semantik zu erkunden.
- Mit dem Editor können Sie ein Python-Programm aus mehreren Anweisungen schreiben, speichern und ausführen. Mehr dazu lesen Sie im nächsten Kapitel.

Wenn Sie IDLE starten, öffnet sich zunächst die IDLE-Shell. Sie sehen ein Anwendungsfenster wie in Abbildung 1.4.

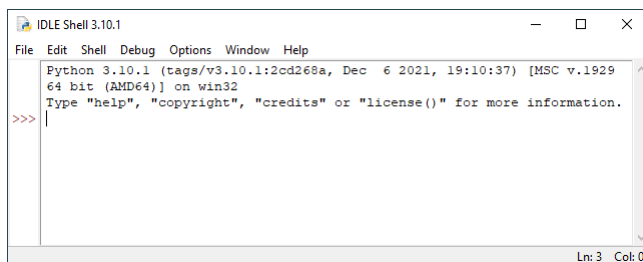


Abb. 1.4: Die IDLE-Shell

Nach einem Begrüßungstext erscheint der Prompt `>>>` des Python-Interpreters. Wenn Sie eine Python-Anweisung eingeben und `Enter` drücken, erscheint in der nächsten Zeile das Ergebnis.

1.8 Hotkeys für die IDLE-Shell

Es gibt zwei Tastenkombinationen (Hotkeys), die die Arbeit mit der IDLE-Shell erleichtern.

Mit `Alt+p` und `Alt+n` können Sie in der Folge der zuletzt eingegebenen Kommandos (*History*) vor- und zurückgehen. Geben Sie zunächst zwei beliebige Befehle ein:

```
>>> 1 + 1
2
>>> 2 * 2
4
>>>
```

Wenn Sie *einmal* die Tastenkombination `Alt+p` betätigen, erscheint hinter dem letzten Prompt das vorige Kommando (*previous*):

```
>>> 2 * 2
```

Bei nochmaliger Eingabe dieses Hotkeys erscheint die vorvorige Zeile:

```
>>> 1 + 1
```

1.9 Anweisungen

Anweisungen sind die Grundbausteine von Computer-Programmen. Man kann sie grob in einfache und zusammengesetzte Anweisungen einteilen. Eine zusammengesetzte Anweisung enthält als Bestandteile weitere Anweisungen und kann sehr kompliziert aufgebaut sein. An dieser Stelle lernen Sie zunächst nur einige grundlegende einfache Anweisungen kennen. Alle anderen werden später in verschiedenen Kapiteln eingeführt.

1.9.1 Ausdruck

Die einfachste Form einer Anweisung besteht aus einem Ausdruck. Bereits eine einzelne Zahl oder eine Zeichenkette ist ein Ausdruck und ergibt eine Anweisung, die freilich nichts bewirkt. Der eingegebene Wert wird vom Python-Interpreter so, wie er ist, wieder ausgegeben:

```
>>> 12
12
>>> 'Hallo'
'Hallo'
```

Mithilfe von Operatoren (z.B. +, -, *, / für die vier Grundrechenarten) und runden Klammern können Sie wie in der Mathematik komplexe arithmetische Ausdrücke aufbauen. Sie werden vom Python-Interpreter ausgewertet und das Ergebnis in der nächsten Zeile ausgegeben:

```
>>> 1000 * 1000
1000000
>>> (1 + 2) * (3 - 4)
-3
```

Kapitel 1

Willkommen zu Python!

Vergleiche gehören ebenfalls zu den Ausdrücken. Ist ein Vergleich wahr, liefert der Interpreter den Wert `True`, ansonsten `False`.

```
>>> 'Tag' == 'Nacht'  
False  
>>> 2 > 1  
True
```

1.9.2 Funktionsaufruf

Funktionen sind aufrufbare Objekte (*callable objects*), die eine bestimmte Aufgabe lösen können. Wenn eine Funktion aufgerufen wird, übernimmt sie gewisse Daten als Eingabe, verarbeitet diese und liefert neue Daten als Ausgabe zurück. Man kann sich die Funktion als einen Spezialisten vorstellen, der bestimmte Tätigkeiten beherrscht. Beim Aufruf übergibt man ihm Material, das bearbeitet er und gibt schließlich dem Auftraggeber ein Produkt zurück.

Die Daten, die man einer Funktion übergibt, nennt man *Argumente* oder *aktuelle Parameter*. Im interaktiven Modus kann man eine Funktion aufrufen und erhält dann in der nächsten Zeile das zurückgegebene Ergebnis. Hier einige Beispiele:

```
>>> round(1.7)  
2
```

Hier ist `round` der Name der Funktion und die Zahl `1.7` das Argument. Zurückgegeben wird die gerundete Zahl.

Die Funktion `min()` akzeptiert eine beliebige Anzahl von Argumenten und gibt den kleinsten Wert (das Minimum) als Ergebnis zurück:

```
>>> min(1, 2)  
1  
>>> min(10, 2, 45, 5)  
2
```

1.9.3 Zuweisung

Zuweisungen sind wahrscheinlich die häufigsten Anweisungen in Programmtexten.

Einen Wert zuweisen

Die einfachste Form der Zuweisung besteht aus einem Namen, gefolgt von einem Gleichheitszeichen und einem Wert, sie hat also die Form:

```
name = wert
```

Beispiel:

```
>>> x = 1
```

In diesem Beispiel ist `x` ein Name und `1` ein Wert. Man bezeichnet `x` auch als Variable, der man einen Wert zugewiesen hat.

Der Zuweisungsoperator ist das Gleichheitszeichen. Beachten Sie, dass die Zuweisung etwas anderes ist als ein Vergleich! Wenn man in einem Ausdruck zwei Objekte auf Gleichheit testen will, verwendet man ein doppeltes Gleichheitszeichen.

Beispiel:

```
>>> 2 == 1
False
```

Anschaulich kann man sich Variablen als Namen für Daten vorstellen. Der Variablenname ist eine Art »Etikett«, das an einer Zahl oder einem anderen Wert »klebt«.

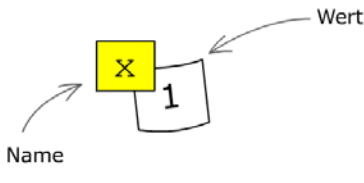


Abb. 1.5: Variable als Name für eine Zahl

Manchmal sagt man auch, dass eine Variable einen Wert *speichert*. Dann stellt man sich die Variable als Behälter vor. Der Variablenname ist die Aufschrift des Behälters und der Wert ist der Inhalt.

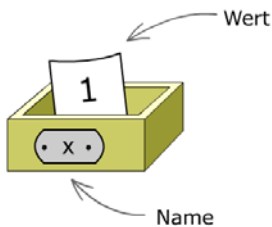


Abb. 1.6: Variable als Behälter

Über den Namen der Variablen kann man auf ihren Inhalt zugreifen. Gibt man im interaktiven Modus den Namen ein, so liefert der Interpreter den Inhalt zurück:

```
>>> x
1
```

Bei einer weiteren Zuweisung wird der alte Wert der Variablen durch einen neuen Wert überschrieben:

```
>>> x = 100
>>> x
100
```

Variablennamen können auch in Ausdrücken verwendet werden. Wenn der Interpreter den Ausdruck auswertet (also ein Ergebnis ermittelt), verwendet er den Wert, der dem Namen zugeordnet ist.