

Michael Weigend

PYTHON 3

FÜR STUDIUM UND AUSBILDUNG

Einfach lernen
und professionell anwenden



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	Literele 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: Würfel	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 WSGI-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
	Glossar	417
	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 lektorat@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**.

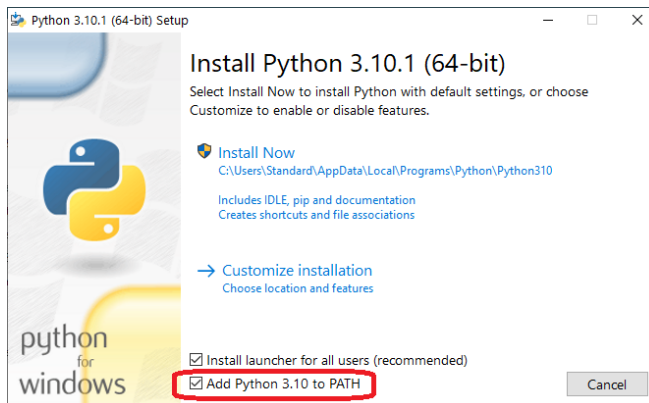


Abb. 1.2: Installation von Python unter Windows

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 `$`.

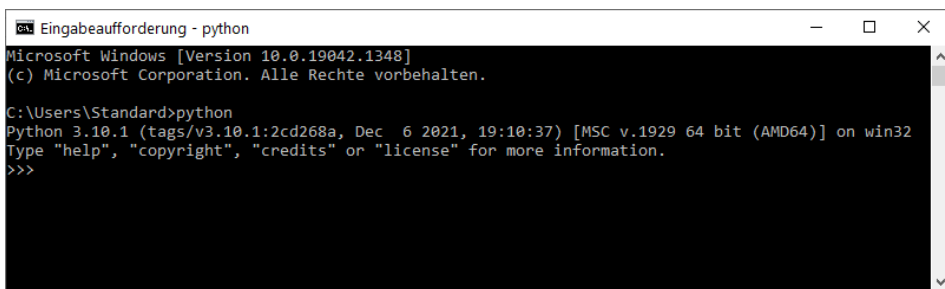


Abb. 1.3: Aufruf des Python-Interpreters in einem Konsole-Fenster (Eingabeaufforderung) unter Windows

Hinter dem Prompt des Betriebssystems geben Sie den Befehl

```
python
```

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.

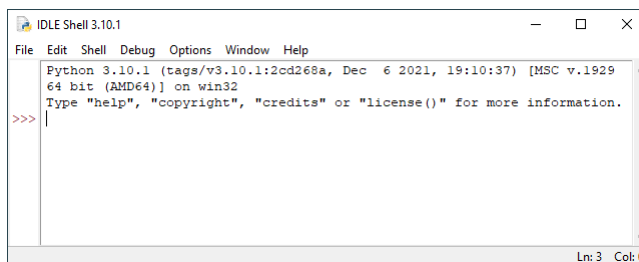


Abb. 1.4: Die IDLE-Shell

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
```

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«.

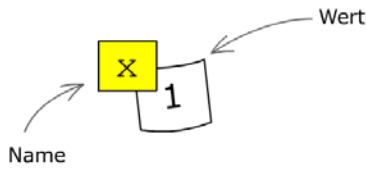


Abb. 1.5: Variable als Name für eine Zahl

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.

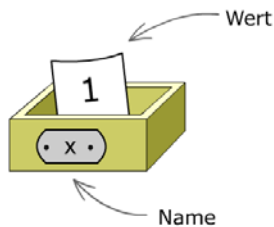


Abb. 1.6: Variable als Behälter

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.

Beispiel:

```
>>> x = 2
>>> 2 * x + 1
5
```

Werte übertragen

Werte können von einer Variablen auf eine andere übertragen werden. Das allgemeine Format einer solchen Art der Zuweisung ist

```
name1 = name2
```

Beispiel: Nach den folgenden Zuweisungen haben die Variablen *x* und *y* den gleichen Wert:

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

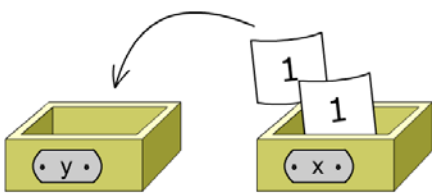


Abb. 1.7: Veranschaulichung einer Werteübertragung

Zuweisungen für mehrere Variablen

In einer einzigen Zuweisung kann man bei Python mehreren Variablen gleichzeitig einen (gemeinsamen) Wert zuordnen:

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

Man kann auch in einer einzigen Zuweisung gleich mehreren Variablen Werte zuordnen. Links vom Gleichheitszeichen stehen dann mehrere Namen (jeweils durch Kommas getrennt) und rechts gleich viele Werte (ebenfalls durch Kommas getrennt):

```
>>> x, y = 1, 2
>>> x
1
>>> y
2
```

Es ist möglich, in einer einzigen Zuweisung (fett gedruckt) die Werte zweier Variablen zu *vertauschen*:

```
>>> x, y = 1, 2
>>> x, y = y, x
>>> x
2
>>> y
1
```

Welche Variablennamen sind erlaubt?

Den Namen einer Variablen können Sie bestimmen. Sie müssen sich aber an folgende drei Syntaxregeln halten:

- Ein Name kann Buchstaben, Ziffern und Unterstriche `_` enthalten. Andere Zeichen sind nicht erlaubt.
- Ein Name muss mit einem Buchstaben oder einem Unterstrich beginnen.
- Ein Schlüsselwort (*keyword*) darf nicht als Name verwendet werden.
- Schlüsselwörter (*keywords*) sind reservierte Wörter, die eine programmtechnische Bedeutung haben. So steht z.B. das Wort `True` für den Wahrheitswert *wahr*.

Hier ist eine Liste der Schlüsselwörter:

<code>and</code>	<code>as</code>	<code>assert</code>	<code>break</code>	<code>class</code>
<code>continue</code>	<code>def</code>	<code>del</code>	<code>elif</code>	<code>elseexcept</code>
<code>False</code>	<code>finally</code>	<code>for</code>	<code>from</code>	<code>global</code>
<code>import</code>	<code>if</code>	<code>in</code>	<code>is</code>	<code>lambda</code>
<code>None</code>	<code>nonlocal</code>	<code>not</code>	<code>or</code>	<code>pass</code>
<code>raise</code>	<code>return</code>	<code>True</code>	<code>try</code>	<code>while</code>
<code>with</code>	<code>yield</code>			

Gültige Namen sind z.B. `a`, `zahl`, `zahl_1`, `geldbetrag`, `_körpergröße`.

Ungültig sind dagegen die folgenden Wörter:

- `1_zahl` (beginnt mit Ziffer),
- `Atem-Frequenz` (enthält nicht erlaubtes Sonderzeichen `-`).

Üblicherweise schreibt man Variablennamen mit kleinem Anfangsbuchstaben.

1.9.4 Erweiterte Zuweisungen

Eine erweiterte Zuweisung ist eine Kombination aus einer Zuweisung und einer Rechenoperation.

Beispiel:

```
>>> x = 10
>>> x += 1
>>> x
11
```

Die Anweisung `x += 1` hat die gleiche Wirkung wie:

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

Der Wert der Variablen `x` wurde um 1 erhöht. Auch für die anderen Grundrechenarten gibt es erweiterte Zuweisungen.

Beispiel Multiplikation:

```
>>> y = 100
>>> y *= 2
>>> y
200
```

1.10 Zahlen verarbeiten – Python als Taschenrechner

Sie können die IDLE-Shell als komfortablen Taschenrechner verwenden. Im einfachsten Fall geben Sie einen mathematischen Ausdruck ein und drücken die Taste `Enter`. In der nächsten Zeile erscheint das Ergebnis.

Ein mathematischer Term (Ausdruck) kann aus Zahlen, Operatoren und runden Klammern aufgebaut werden. Die Schreibweise ist im Prinzip wie in der Mathematik. Allerdings gibt es ein paar Besonderheiten.

1.10.1 Operatoren

Für Multiplikationen verwendet man in Python den Stern `*`.

Beispiel:

```
>>> 100 * 21
2100
```

Es gibt keine langen Bruchstriche. Für Zähler oder Nenner müssen Sie eventuell Klammern verwenden. Den Bruch

$$\frac{3+2}{2}$$

stellen Sie durch den Ausdruck `(3+2) / 2` dar:

```
>>> (3 + 2) / 2
2.5
```

Achten Sie darauf, dass das Ergebnis `2.5` und nicht `2,5` ist. Dezimalbrüche enthalten kein Komma, sondern stattdessen einen Punkt.

Es gibt eine exakte Division `/` und eine ganzzahlige Division `//`. Die ganzzahlige Division liefert immer eine ganze Zahl. Sie ist das abgerundete Rechenergebnis einer Division. Probieren Sie es aus:

```
>>> 3 / 2
1.5
>>> 3 // 2
1
```

Zum Potenzieren einer Zahl verwenden Sie den Operator `**`. Die Potenz 2^8 («zwei hoch acht») schreiben Sie `2**8`.

```
>>> 2**8
256
>>> 5**1
5
>>> 5**2
25
>>> 5**-1
0.2
>>> 5**200
6223015277861141707144064053780124240590252168721167133101116614789
6988340353834411839448231257136169569665895551224821247160434722900
390625
>>>
```

Speziell ist die Modulo-Operation. Sie liefert den Rest einer ganzzahligen Division. So ergibt 5 geteilt durch 2 das Ergebnis 2 mit dem Rest 1.

```
>>> 5 % 2
1
```

Die Zahl 6 ist durch 2 teilbar. Bei der Division bleibt kein Rest:

```
>>> 6 % 2
0
```

Sie verwenden die Modulo-Operation zum Beispiel, wenn Sie prüfen wollen, ob eine Zahl durch eine andere Zahl teilbar ist.

Bei Termen mit mehreren Operatoren müssen Sie deren Prioritäten beachten. Sie kennen das aus der Schulmathematik. Die Operation mit höherer Priorität wird zuerst ausgeführt. Der Potenzoperator hat die höchste Priorität, dann kommen Multiplikation und Division. Addition und Subtraktion haben die niedrigste Priorität.

```
>>> 2*3**2
18
```

Hier berechnet der Computer *zuerst* 3 hoch 2 (das macht 9) und multipliziert dann das Ergebnis mit 2.

```
>>> (2*3)**2
36
```

Der Term in der Klammer wird immer zuerst ausgewertet. Klammern haben die allerhöchste Priorität.

Operator	Erklärung
()	Klammern
**	Potenzieren
* / // %	Multiplikation, Division, ganzzahlige Division, Modulo
+ -	Addition, Subtraktion

Tab. 1.1: Arithmetische Operatoren in der Reihenfolge ihrer Priorität (von oben nach unten)

1.10.2 Variablen verwenden

Bei komplizierten Rechnungen können Sie Zwischenergebnisse in Variablen speichern. Auf diese Weise wird die Rechnung übersichtlicher.

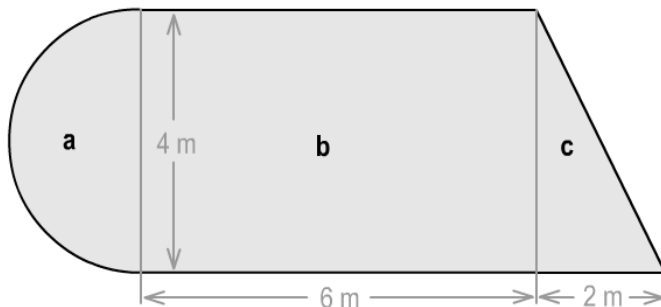


Abb. 1.8: Eine Fläche, die aus mehreren einfachen Flächen zusammengesetzt ist

Nehmen wir als Beispiel die Berechnung einer Fläche wie in Abbildung 1.8. Mit Variablen kann man die Berechnung in vier Schritte aufteilen. Zuerst werden drei Teilflächen berechnet und dann die Summe gebildet:

```
>>> a = 3.14 * 2**2 / 2
>>> b = 6 * 4
>>> c = 2 * 4 / 2
>>> fläche = a + b + c
>>> fläche
34.28
```

1.11 Eine weitere Entwicklungsumgebung: Thonny

IDLE wird bei der Installation von Python gleich mitgeliefert. Darüber hinaus gibt es aber auch noch weitere kostenlose Entwicklungsumgebungen (IDEs), die aus anderen Quellen stammen. Eine bekannte IDE für professionelle Python-Entwickler ist PyCharm (<https://www.jetbrains.com/pycharm/>). Einige Entwicklungsumgebungen kann man nicht nur für Python, sondern auch für andere Programmiersprachen verwenden. Dazu gehören Eclipse und Geany.

Thonny ist eine IDE für Python-Einsteiger, die an der Universität von Tartu in Estland entwickelt worden ist (<https://thonny.org>). Ein besonderes Merkmal von Thonny ist, dass es die Arbeitsweise des Python-Interpreters veranschaulicht. Sie können ein Pro-

gramm schrittweise durchlaufen und z.B. nachvollziehen, wie Ausdrücke ausgewertet werden. Thonny ist selbst in Python geschrieben und läuft auf den gängigen Betriebssystemen (Linux, Windows, macOS). Wenn Sie Python auf Ihrem Rechner installiert haben, können Sie Thonny mit dem Programm `pip` (*Package Installer for Python*) herunterladen und installieren, denn `pip` gehört zur Python-Standardinstallation.

Öffnen Sie ein Konsole-Fenster. Unter Windows geben Sie dazu unten links in das Suchfeld `cmd` ein und drücken `Enter`. Geben Sie das folgende Kommando ein und beenden Sie die Eingabe mit `Enter`.

```
pip install thonny
```

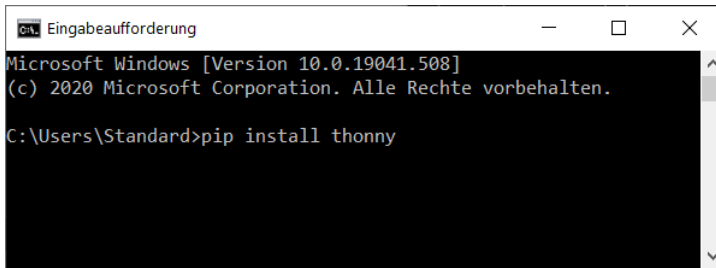


Abb. 1.9: In der Konsole mit `pip` ein Python-Programm installieren (Windows)

Abbildung 1.9 zeigt die Benutzungsoberfläche von Thonny. Wie IDLE bietet Thonny einen Editor (oben) und eine Shell (unten) – allerdings in ein und demselben Fenster.

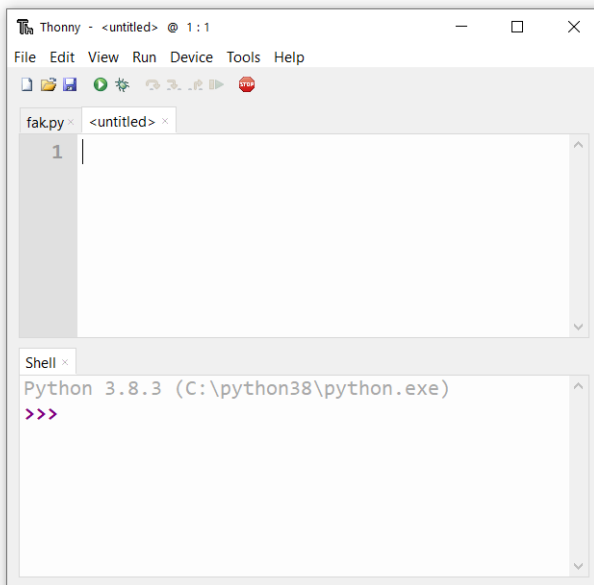


Abb. 1.10: Die Benutzungsoberfläche von Thonny. Editor (oben) und Shell (unten) sind in einem Fenster.

1.12 Notebooks mit Jupyter und CoLab

Eine Alternative zu konventionellen Entwicklungsumgebungen ist *Jupyter Notebook*. Dabei handelt es sich um ein System, mit dem Sie sogenannte *Notebooks* gestalten können.

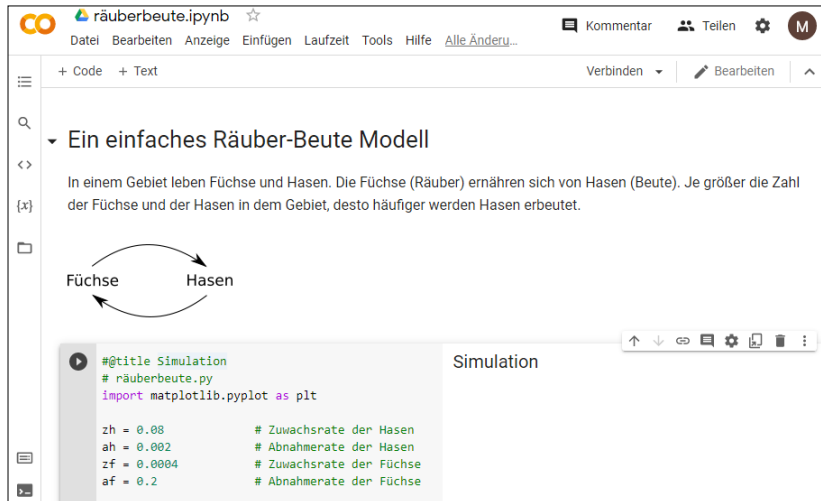


Abb. 1.11: Jupyter-Notebook mit Text, Bild und Python-Code

Ein Notebook ist ein interaktives Dokument, das formatierten Text, Bilder, Python-Code und Zellen für Eingaben und Ausgaben enthält. Man verwendet Notebooks typischerweise für wissenschaftliche Projekte, in denen ein Algorithmus zur Analyse von Daten und die Visualisierung der Berechnungsergebnisse im Vordergrund stehen. Das Notebook kann als pdf-Datei exportiert und dann als wissenschaftliche Arbeit veröffentlicht werden. Der Charme liegt unter anderem darin, dass die Diagramme von dem im Notebook angegebenen und erläuterten Python-Programm erzeugt werden.

Jupyter Notebook ist eine komplexe Software, die Sie kostenlos herunterladen und auf Ihrem heimischen Computer installieren können (<https://jupyter.org/>).

Einfacher ist es, Google Colaboratory (oder kurz Colab) zu nutzen, um Notebooks zu erstellen (<https://colab.research.google.com/>). Colab ist ein Jupyter-Notebook-System, das von Google gehostet wird. Mit Colab brauchen Sie auf Ihrem Computer nichts zu installieren. Sie nutzen einfach Ihren Webbrowser, und alle Berechnungen finden in der Cloud statt.

In Kapitel 16 finden Sie einige Beispiele für die Nutzung von Colab im Zusammenhang mit wissenschaftlichem Rechnen und Datenvisualisierung.

1.13 Rückblick

- Ein *Algorithmus* ist eine präzise Anleitung zur Lösung einer Aufgabe.
- Ein ist ein Algorithmus, der mit einer Programmiersprache wie Python formuliert worden ist.

- Die `def` definiert die Schreibweise und die `:` die Bedeutung von Anweisungen der Programmiersprache.
- Ein Python-Programm wird von einem `python` ausgeführt.
- Die Entwicklungsumgebung IDLE besteht aus der *IDLE-Shell* und einem *Programmeditor*.
- In der IDLE-Shell kann man einzelne Anweisungen ausprobieren, im Programm-
editor schreibt man ein komplettes Python-Programm, speichert es ab und testet es.
- In einer `Variable` wird einem Namen ein Wert (z.B. eine Zahl) zugeordnet.
- Für `Identifikatoren` gibt es Syntaxregeln: Ein Name besteht aus Buchstaben, Ziffern und Unterstrichen `_` und muss mit einem Buchstaben oder Unterstrich beginnen.
- Python enthält `Reservewörter`, die nicht als Namen verwendet werden dürfen.

1.14 Übungen

Übung 1: Ausdruckanweisungen

Welche Ergebnisse liefern die folgenden Ausdruckanweisungen? Schreiben Sie in die Tabelle Ihre Vermutung und vielleicht ein Stichwort zur Erklärung.



Einige Anweisungstexte sind fehlerhaft.

Anweisung	Ergebnis	Erklärung
<code>2 + 3 * 2</code>		
<code>(2 + 3) * 2</code>		
<code>10 / 2</code>		
<code>10 // 3</code>		
<code>-10 // 3</code>		
<code>10 % 3</code>		
<code>11 % 3</code>		
<code>12 % 3</code>		
<code>2 ** 3</code>		
<code>4 ** 0.5</code>		
<code>2 + - 2</code>		
<code>1,5 * 2</code>		
<code>1.5 * 2</code>		
<code>2 + * 3</code>		
<code>round(1.23)</code>		
<code>1 > 2</code>		
<code>1 == 1.0</code>		

Übung 2: Zuweisungen

Durch Zuweisungen werden Variablen erzeugt und die Werte von Variablen verändert. Ergänzen Sie in der folgenden Tabelle die Werte der Variablen.

Anweisung	a	b	c
a = b = 1	1	1	
a = a + 2			
b += 1			
c = a + b			
c = 2 * c			
a, b = 2.0, 2			
c = a / b			



Zu Beginn (in den ersten drei Zeilen) ist die Variable c nicht definiert.

Die Lösungen zu diesen Übungen können Sie durch Experimente in der IDLE-Shell selbst ermitteln. Außerdem finden Sie alle Lösungen in einer Datei, die Sie von der Website des mitp-Verlages herunterladen können (www.mitp.de/0434).

1.15 Lösung der Frage: Semantik im Alltag

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

Obwohl der Satz nach einer klaren Anweisung klingt, kann er in unterschiedlichen Situationen unterschiedlich verstanden werden, z.B.

- Richte deinen Blick nach vorne und passe auf, was da passiert.
- Sei optimistisch und denke an die Zukunft.

Stichwortverzeichnis

Symbole

1-zu-n-Beziehung 311

A

Abstraktion 144

Aggregation 294

Agile Software-Entwicklung 404
and 79

Änderbarkeit 405, 412

Annotation 101

Anweisung

 aufteilen 77

 leere 77

Anwendungsfenster 218

 Titel ändern 219

Applikationsobjekt 382

Arbeitsspeicher 151

Argument 93

argv 209

Arithmetische Operatoren 340

Array 331

 ändern 338

 Datentyp 332

 erzeugen (NumPy) 332

 Funktion 342

 Index 336

 Matrizen 333

 mehrdimensionales 332

 Methoden 339, 340

 Sicht 337

 Vektoren 333

 Wahrheitswerte 342

 Zahlenfolgen 333

 Zufallszahlen 334

ASCII 380

askopenfile() 238

asksaveasfile() 238

assert 272

Assoziation 294

Asynchron 217

Attribut 283

 Zugriff auf 286

Ausdruck 27, 32

Auskommentieren 66, 273

B

background 220

Basisklasse 296

Baumpatenschaft 329

bd 220

Beck, Kent 404

Bedingte Wiederholung 82

Bedingung 71, 78

 erfüllt 74

 nicht erfüllt 74

Benanntes Tupel 313

Benutzungsoberfläche 217

Betriebssystem 201

Bewegungsprofil 350

Beziehungssymbol 299

Beziehungstyp 310

bg 220

Bild

 als Schaltfläche 247

 einfügen 263

 pixelweise vearbeiten 260

 über Listen verarbeiten 261

 verändern 249

Bildformate 247

Binärdatei 153, 165

 laden 161

 öffnen 160

 speichern 161

Binärzahlen 41

- Bit 154
- Block 77
- Bogenmaß 117
- bool 45, 79
- Boolescher Operator 79
- Boolescher Wert 79
- borderwidth 220
- Bounding Box 252
- Breakpoint 277
- Bug 64
- Button 218, 223
- Byte 154
- Bytestring 154
 - in String umwandeln 154

C

- Calltip 100
- Canvas 218, 252
 - Methoden 252
- Casting 48, 49
- CGI 369
 - Umlaute 380
 - Unix 373, 375
 - Windows 373
- CGI-Skript
 - Aufbau 372
- chdir() 202
- Checkbox 218
- Client 371
- Client-Server-System 314
- close() 152
- Codierung 152
- Colab
 - Siehe Google Colaboratory*
- column 228
- columnspan 228
- Common Gateway Interface
 - Siehe CGI*
- Compiler 23
- Computer-Labor 329
- CREATE TABLE 315
- Cursor 316
 - Position 233

D

- Daemon 371
- Datei
 - anlegen 206
 - aus dem Internet 169
 - Modus 151
 - öffnen 151
 - Öffnen-Dialog 238
 - speichern 152
 - Speichern-Dialog 238
- Daten
 - dauerhaft speichern 151
- Datenbank 309
 - relationale 311
- Datenbank-Managementsystem 309
- Datenbanksystem 309
- Datenverarbeitung 57
- Datum 119
- DBMS
 - Siehe Datenbank-Managementsystem*
- Debugger 275
 - Befehle 276
- Debugging 64, 271
 - Ausdrücke 278
 - rekursive Funktionen 278
 - Thonny 278
 - Typen 271
 - Variablen 273
- Default 96
- Deklaration 50
- delete() 232
- Deselektieren 235
- Deserialisierung 161
- Dialogbox 237
- Dictionary 45, 139
 - ändern 140
 - Operationen 140
- Divide and conquer 273
- Division 32
- Docstring 101
- doctest 413
- Dokumentation
 - automatische 102
- Dynamische Typisierung 50

E

Editor 26, 53
 Einrückung 57, 72, 75, 77
 elif 75
 elif-Klausel 76
 else 73
 else-Klausel 73
 Emoji 175
 Encoding 176
 Endlosrekursion 105
 Endloswiederholung 87
 endswith() 177
 Entitätstyp 310
 Entity-Relationship-Diagramm 309
 Entry 218
 Entscheidungsbaum 91
 Entwicklungsprozess 403
 Entwicklungsumgebung
 IDLE 26
 Thonny 34
 Epoche 413
 ER
 Siehe Entity-Relationship-Diagramm
 Ergebnisrelation 317
 Ergebnistabelle 317
 Erweiterte Zuweisung 31
 Escape-Sequenz 176
 EVA-Prinzip 55, 67
 Event 223
 Eventhandler 223, 224
 Execution-Frame 278, 279
 exists() 202
 Exponentialschreibweise 42
 Extreme Programming 404

F

Fallunterscheidung 75
 False 45
 Farbfeld 228
 Fehler
 logischer 66
 semantischer 271
 Syntaxfehler 66
 Fehlersuche 64
 Tipps 66

fg 220
 FieldStorage 379
 filter() 110
 filter-Objekt 110
 finally 156
 find_all() 252
 findall() 184
 kürzester Teilstring 189
 float 42
 FLOAT 315
 flush() 152
 font 220
 for 88
 foreground 220
 Formatierung 191
 Formatstring 192
 Formular 375
 Fremdschlüssel 312
 Funktion 93
 Annotation 101
 anonyme 107
 Benennung 101
 Definition 94
 rekursive 93, 105, 145
 zweistellige 109
 Funktionales Paradigma 110
 Funktionsaufruf
 verschachtelt 60
 Funktionsdefinition
 verbessern 101
 Funktionskopf 94
 Funktionskörper 94

G

Ganzzahl 40
 Garbage Collection 211
 Gaußsche Normalverteilung 335
 get 377
 get() 232
 getatime() 202
 getcwd() 202
 getmtime() 202
 getPixel() 260
 getsize() 202
 Gieriges Finden 189
 Gleitkommazahl 42

Google Colaboratory 354

Siehe Colab

Grafik-Element

ID 254

löschen 255

malen 252

Grafische Benutzungsoberfläche 217

Graph 143

als Dictionary 143

greedy 185, 189

Greenscreen 263

grid() 227

GUI

Siehe Grafische Benutzungsoberfläche

H

Haltepunkt 277

Hashing 140

Hashtag 57

Heatmap 349

height 220

help() 102

Hexadezimal 175, 221

Hexadezimalzahlen 41

Histogramm 347

Klassen 348

History 27

Hosting-Dienst 390

Hotkey 27

HTML 369

Body 372

Eingabefeld 377

Radiobutton 377

Submit-Button 378

Tag 370

html-Text 172

HTTPResponse 169

HTTP-Server 374

I

Icon 247

IDLE 26

Debugger 275

ID-Nummer 254

if 71

if-Anweisung

verschachtelt 75

if-Klausel 72

image 220

Image-Objekt 258

Methoden 259, 260

tkinter 265

Imperativer Programmierstil 283

Imperatives Paradigma 110

Implementierung 291, 293

Import

Funktion eines Moduls 116

Konstante 117

Module 115

Index 232

Induktiv 85

Initialisierung 84

Initialisierungsmethode 285

input() 63

insert() 232

Installation 23

Instanz 284, 286

Instanziieren 286

int 40

INT 315

Interaktiver Modus 25

Interaktives Programm 125

Internet

Daten laden 169

Interpreter 23

IPv4-Adresse 375

isdir() 202

isfile() 202

Item 47

Iteration 87, 404, 406

Liste 131

Tupel 131

J

JavaScript 165

JavaScript Object Notation

Siehe JSON

JPEG 265

JSON 165
 Array 165
 Elementare Werte 166
 Objekt 165
 Jupyter Notebook 36, 354
 Bilder einfügen 358
 öffnen 359
 speichern 359
 teilen 363
 Text-Zelle 356
 justify 220
K
 Kardinalität 311
 keys() 140
 Kinokarte 74
 Klasse 284, 293
 abgeleitete 296
 definieren 285
 Klassenattribut 288, 290
 zugreifen auf 290
 Klassendiagramm 284
 Klassensymbol 299
 Kollektion 47
 Vorkommen 47
 Kommandozeile 25
 Kommandozeilenargumente 209
 Kommentar 57
 Konkatenation 48, 132
 Konstante 117
 importieren 117
 Kontextmanager 158
 Kontrollstruktur 71
 Kontrollvariable 235
 Koordinate 252
 Kurze Zeichenkette 43

L

Label 218
 Lambda 107
 Lastenheft 403
 Laufvariable 88
 Laufzeit 399
 Laufzeitfehler 155

Laufzeitsystem 208
 Layout-Manager 224
 Lesezugriff 152
 LIKE 325
 Lineares Programm 57
 Linie 254
 Operationen 254
 Liniendiagramm 344
 Farbe ändern 346
 Legende 346
 mehrere Linien 346
 list 44
 List Comprehension 136
 listdir () 202
 Liste 44, 129, 133
 aus Kollektion 135
 auspacken 131
 aus Zahlen 135
 erzeugen 135
 Literal 39
 Logging 273
 Logischer Fehler 66
 lower() 177

M

Magische Methode 291, 293
 map() 108
 map-Objekt 108
 Markdown 356, 358
 maskieren 188
 math 117
 Matrix 331, 333
 transponiert 339
 Matrizenmultiplikation 341
 Menge 44, 47
 Mengenabstraktion 234
 Metapher 288
 Metatag 380
 Methode 132, 134, 283
 aufrufen 287
 definieren 285
 magische 291, 293
 überschriebene 296
 mkdir() 206

Modell 129
 Modul 115
 erstellen 122
 importieren 115
 installieren 256
 testen 123
 Vorteile 126
 Modulo 33
 Modus 151
 interaktiver 25
 Multiplikation 32
 MySQL 311
 m-zu-n-Beziehung 311

N

Näherungsrechnung 85
 Namensraum 95
 Nebenläufigkeit 239
 NoneType 46
 Normalverteilung 335
 not 79
 Notebook 36
 Siehe Jupyter Notebook
 Notizbuch 405
 NumPy 252, 331

O

Objekt 134, 283
 anonymes 296
 neues erzeugen 286
 variable Anfangswerte 287
 Wert ausgeben 289
 Objektorientierte Programmierung 283
 Objektvariable 283
 Oktalzahlen 42
 Oktett 154
 OOP
 Siehe Objektorientierte Programmierung
 Operationen
 arithmetische 340
 Operator 78
 boolescher 79
 überladen 291
 or 79
 os (Modul) 201

Oval 252
 Overhead 211

P

Packer 224
 Optionen 226
 padx 228
 pady 228
 Pair-Programming 404
 Paket
 Siehe Package
 Parameter 94, 138
 optionaler 96
 Parameterliste 94
 pass 77
 Passwortkontrolle 72
 PEP3333 382
 Performance 126, 399, 402
 Performance-Analyse 402
 Pfad 151
 absoluter 155
 relativer 155
 PhotoImage 218
 Methoden 249
 pickle 160
 PIL.Image
 resize() 260
 size 260
 PIL.Image-Objekt 265
 Pivot 273
 Platzhalter 192
 Format 193
 Namen 192
 Polymorphie 291
 Positionsargument 99
 post 377
 Potenz 33
 Primärschlüssel 310
 print() 60, 103
 Profiler 402
 Programm
 ausführen 60
 interaktives 125
 linear 57
 Programmabsturz 155

Programm ausführen 60
 Doppelklick 62
 Kommandozeile 61
 Linux 64
 macOS 64
 Programmierstil
 imperativer 283
 Programmierung
 objektorientierte 283
 Programmmlauf
 abbrechen 87
 Promptstring 56
 Prozedur 98
 PyPI 126, 256
 Python Anywhere 390
 Dashboard 390
 Framework 391
 https 392
 Konfigurationsdatei 392
 Link 392
 manuelle Konfiguration 391
 Web-Tab 390
 Python-Homepage 23
 Python Imaging Library 257
 Python Package Index 126, 256

Q

Quantor 187
 Query 317, 318
 Querystring 377
 Quicksort 273, 399
 Zeitkomplexität 401

R

Radiobutton 218, 235
 vorselektieren 235
 random 118
 range() 89
 range-Objekt 89
 Raster-Layout 227
 Raw-String 185
 re 184
 Read-Eval-Print-Loop 26
 Rechteck 252
 Refactoring 405, 408
 Referenz 211

regex
 Siehe Regulärer Ausdruck
 Regulärer Ausdruck 182
 maskieren 188
 Platzhalter 186
 Sonderzeichen 183, 184
 verknüpfen 187
 Rekursion 93, 105
 endlose 105
 Relation 311
 Relationale Datenbank 311
 Release Planning 404, 405
 relief 220
 REPL
 Siehe Read-Eval-Print-Loop
 replace() 177
 Repository 126
 return-Anweisung 94
 Richter, Gerhard 228
 Routenplaner modellieren 143
 row 228
 rowspan 228

S

Schaltfläche
 Siehe Button
 Schleife 82
 Schlüssel 45, 139
 Schlüsselattribut 310
 Schlüssel-Wert-Paar 140
 Schlüsselwort 31
 Schlüsselwortargument 99
 Vorteile 99
 Schreibzugriff 152
 Schrifttyp 221
 Seiteneffekt 249
 Selektieren 235
 self 285
 Semantik 22
 Sequenz 47, 129
 änderbare 129
 komplexe 130
 Konkatenation 48
 nicht änderbare 130
 Operationen 47, 132
 Zugriff über den Index 47

- Sequenziell 239
- Serialisierung 161
- Server 371
- set 44
- Shebang 64
- Shell 25
- Shortcut 27
- Signatur 102
- Skript 53
- Slicing 337
- Software Engineering 403
- Sortieren
 - Strings 134
 - Zahlen 134
- Spaltenvektor 333
- Speichern
 - String 153
- Speicherverwaltung 210
- split() 177
- SQL 314, 315
 - Datentypen 315
 - Platzhalter 325
 - Variablen 320
- sqlite3 314
- Standardausgabe 213, 273
- Standard-Typ 40
- Story 404, 405
 - implementieren 406
- Storytelling 195
- str 43
- Straight Selection 400
- Stream 151
- String 43
 - kurze Zeichenkette 43
 - mit Variablen 191
 - Operationen 177
 - Platzhalter 192
 - speichern 153
- Stringformatierung 191
- strip() 177
- Suchfunktion 137
- SVG-Format 255
- Synchron 217
- Syntax 22
- Syntaxfehler 64, 67
- Syntax-Highlighting 54
- Syntaxregel 101
- sys (Modul) 208
- T**
 - Tabelle 192
 - Tastenkombination 27
 - Teile und herrsche 273
 - Temperaturdaten verwalten 167
 - Term 32
 - Test Driven Development 405, 413
 - Text 218, 220
 - Textautomat 195
 - Text-Dateien
 - freie 172
 - Textpassagen finden 184
 - Textposition 232
 - Textverarbeitung 175
 - Codierung 176
 - Text-Widget 231
 - Thonny 34, 278
 - Thread 241
 - neuen starten 241
 - Ticketbuchungssystem 415
 - time 119
 - Tk 218
 - Tracing 273
 - True 45
 - try except 156
 - Tupel 130
 - auspacken 131
 - benanntes 313
 - definieren 131
 - Typ 293
 - abfragen 39
 - Vorbedingung testen 271
 - Typ-Hierarchie 40
 - Typisierung
 - dynamische 50
 - Typumwandlung 49
- U**
 - Überladen 291
 - Uhrzeit 119
 - Umfrage 386
 - UML 284, 294, 296, 299, 302
 - UML-Klassendiagrammen 403

UML-Klassensymbol 291
 Unicode 175
 Unix 373
 Unix-Zeit 119
 Unterklasse 296
 upper() 177
 URL 169
 utf-8 152, 176, 381

V

values() 140
 VARCHAR 315
 Variable 29
 globale 104
 in Texten 191
 lokale 104
 prüfen 275
 Variablenname 29, 31
 Vektor 331, 333
 Vektorgrafik 255
 Verbinden
 explizit 78
 implizit 78
 Verbindungsobjekt 316
 Vererbung 296
 Vergleichskette 80
 Verzeichnis
 anlegen 206
 Verzeichnisbaum 154, 203
 Viewer 258
 Vorbedingung 271
 testen 272

W

Wahrheitswert 45, 79, 272
 Währungsrechner 236
 Webcam 266
 Webscraping 180, 190
 Webseite
 dynamische 369, 370
 Informationen auslesen 180
 interaktive 375
 Web Server Gateway Interface
 Siehe WSGI
 Wert 139
 boolescher 79

Wertübergabe 95
 while 82
 Widget 217
 Abstand 225
 Aussehen ändern 220
 Bilddatei 247
 einfügen 219
 Farbe ändern 221
 Größe ändern 222
 Master 219
 Methoden 222
 nebeneinander 225
 Optionen 220
 Texteingabe 230, 231
 width 220
 Wiederholung 89
 bedingte 82
 Wikimedia Commons 270
 Windows 373
 with-Anweisungen 157
 WLAN 375
 Wörter zählen 179
 write() 152
 WSGI 369, 382
 Applikationsobjekt 384
 modularisieren 394
 Würfel 118

Z

Zeichen
 Codierung 175
 Zeichenkette 43
 kurze 43
 Zeile 77
 verbinden 77
 Zeilenvektor 333
 Zeitkomplexität 401
 Zeitstempel 256
 Zitatesammlung 320
 Zufallsfunktionen 118
 Zuständigkeit 283
 Zuweisung 28
 erweitert 31
 Zuweisungsoperator 29