

Michael Weigend

9,99 €
inkl. E-Book

PYTHON 3

Schnelleinstieg

Programmieren lernen in 14 Tagen

Einfach und ohne Vorkenntnisse zum Profi

Zahlreiche
Praxisbeispiele
und Übungen



Inhalt

Einleitung

E.1	Programmieren lernen in 14 Tagen	11
E.2	Der Aufbau des Buchs	11
E.3	Achten Sie auf den Schrifttyp!	12
E.4	Programmtexte und Lösungen zum Download	12
E.5	Fragen und Feedback	13



Willkommen zu Python!

1.1	Die Programmiersprache Python	15
1.2	Was ist ein Algorithmus?	16
1.3	Syntax und Semantik	16
1.4	Interpreter und Compiler	17
1.5	Python installieren	18
1.6	Python im interaktiven Modus	20
1.7	Die Entwicklungsumgebung IDLE	21
1.8	Hotkeys für die Python-Shell	22
1.9	Anweisungen	23
1.10	Zahlen verarbeiten – die Python-Shell als Taschenrechner	29
1.11	Übungen	32
1.12	Lösung der Frage: Semantik im Alltag	34



Datentypen – die Python-Typ-Hierarchie

2.1	Literale und die Funktion type()	35
2.2	Die Python-Typ-Hierarchie	36
2.3	Standard-Typen	37
2.4	Gemeinsame Operationen für Kollektionen	42
2.5	Objekte eines Typs erzeugen – Casting	44
2.6	Dynamische Typisierung	46
2.7	Übung: Anweisungen	46



Interaktive Programme

3.1	Das erste Python-Skript	49
3.2	Das EVA-Prinzip	52
3.3	Kommentare	54
3.4	Projekt: Volumenberechnung	55
3.5	Python-Programme starten	58
3.6	Fehler finden	63
3.7	Übungen	65
3.8	Lösungen zu den Fragen	67



Kontrollstrukturen

4.1	Programmverzweigungen	69
4.2	Das Layout von Python-Programmen: Zeilen und Blöcke	76
4.3	Bedingungen konstruieren	77
4.4	Bedingte Wiederholung – while	82
4.5	Iterationen – for	87
4.6	Übungen	90
4.7	Lösungen zu den Fragen	92



Funktionen

5.1	Warum definiert man Funktionen?	93
5.2	Definition und Aufruf einer Funktion	94
5.3	Optionale Parameter und voreingestellte Werte	96
5.4	Eine Funktion in der Shell testen	98
5.5	Die return-Anweisung	99
5.6	Positionsargumente und Schlüsselwortargumente	100
5.7	Guter Programmierstil	102
5.8	Die print()-Funktion unter der Lupe	105
5.9	Globale und lokale Namen	106

5.10	Rekursive Funktionen	107
5.11	Übungen	110
5.12	Lösungen zu den Fragen	112



Mit Modulen arbeiten

6.1	Importanweisungen	113
6.2	Mathematische Funktionen: Das Modul math	115
6.3	Zufallsfunktionen: Das Modul random	116
6.4	Datum und Zeit	118
6.5	Ein eigenes Modul erstellen	120
6.6	Module aus dem Python Package Index (PyPI)	125
6.7	Übungen	125



Mit Kollektionen modellieren

7.1	Sequenzen	127
7.2	Projekt: Telefonliste	137
7.3	Dictionaries	139
7.4	Projekt: Vokabeltrainer	142
7.5	Übungen	144
7.6	Lösungen zu den Fragen	145



Daten speichern

8.1	Wie werden Daten gespeichert?	147
8.2	Projekt: Logbuch	153
8.3	Datenstrukturen speichern und laden: Das Modul pickle	155
8.4	Projekt: Digitaler Planer	158
8.5	Daten aus dem Internet	162
8.6	Übung: News-Check	162
8.7	Lösungen zu den Fragen	163

9

Textverarbeitung

9.1	Unicode-Nummern für Zeichen	165
9.2	Was sind Escape-Sequenzen?	166
9.3	Operationen für Strings	167
9.4	Projekt: Goethes Wortschatz	170
9.5	Projekt: Tageshöchsttemperatur	171
9.6	Texte mit variablen Teilen	175
9.7	Projekt: Storytelling	176
9.8	Übungen	177
9.9	Lösungen zu den Fragen	179

10

Grafische Benutzungsoberflächen

10.1	Widgets	182
10.2	Das Anwendungsfenster Tk	182
10.3	Ein Widget einfügen	184
10.4	Das Aussehen der Widgets gestalten	185
10.5	Gemeinsame Methoden der Widgets	188
10.6	Schaltflächen und Eventhandler	188
10.7	Das Layout verfeinern	190
10.8	Widgets zur Texteingabe	193
10.9	Radiobuttons	199
10.10	Dialogboxen	203
10.11	Parallele Abläufe: Threads	205
10.12	Übungen	209
10.13	Lösungen zu den Fragen	211

11

Grafik programmieren

11.1	Bilder auf Schaltflächen und Labels	213
11.2	Die Python Imaging Library (PIL)	219
11.3	Übungen	222

12

Fehler finden und vermeiden

12.1	Zusicherungen	225
12.2	Tracing	228
12.3	Debugging mit IDLE	230
12.4	Lösungen zu den Fragen	234

13

Objektorientierte Programmierung

13.1	Klassen und Objekte	235
13.2	Projekt: Geld	241
13.3	Operatoren überladen – Polymorphie	244
13.4	Projekt: Abrechnung	249
13.5	Vererbung	252
13.6	Übungen	254
13.7	Lösungen zu den Fragen	257

14

Professionelle Software-Entwicklung

14.1	Die Laufzeit von Programmen	259
14.2	Agile Software-Entwicklung	264
14.3	Projekt: Digitales Notizbuch	267
14.4	Test Driven Development mit doctest	277
14.5	Programmieren als Hobby und Beruf	279
14.6	Übung: Ticketbuchung	281

Stichwortverzeichnis

Einleitung

E.1 Programmieren lernen in 14 Tagen

Mit diesem Buch haben Sie sich für einen einfachen, praktischen und fundierten Einstieg in die Welt der Programmierung entschieden. Sie lernen ohne unnötigen Ballast alles, was Sie wissen müssen, um Python effektiv für Projekte in Ihrem Berufs- und Interessensgebiet einzusetzen.

Wenn Sie Zeit genug haben, können Sie jeden Tag ein neues Kapitel durcharbeiten und so innerhalb von zwei Wochen Programmieren lernen. 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 und Übungen gleich aus. Sie werden schnell erste Erfolge erzielen und Freude an der Programmierung finden.

E.2 Der Aufbau des Buchs

Das Buch beginnt mit den Grundlagen: Installation von Python, Nutzung der Entwicklungsumgebung und Formulierung einfacher Anweisungen. Die Kapitel bauen aufeinander auf. 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 Benutzeroberflächen. 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 Möglichkeiten, wie Sie nach dem Schnelleinstieg Ihre Programmierkenntnisse weiterentwickeln können.

Gelegentlich stoßen Sie auf Zwischenfragen. Sie sind als kleine Lernaktivierung gedacht und werden am Ende des Kapitels beantwortet. Ihr Tagespensum schließt mit praktischen Programmier-Übungen, in denen Sie Ihr neu gewonnenes Wissen vertiefen können. Die Sterne neben den Übungen geben einen Hinweis auf die Schwierigkeit. Dabei sind Übungen mit zwei bis drei Sternen für Leserinnen und Leser gedacht, die eine besondere Herausforderung suchen. Die Lösungen zu diesen Übungen, die relativ viel Programmtext enthalten, sowie ein Glossar mit den wichtigsten Fachbegriffen stehen Ihnen

auf der Website des Verlags zum Download zur Verfügung. Mehr dazu im übernächsten Abschnitt.

Am Ende des Buchs finden Sie ein Stichwortverzeichnis, das Ihnen hilft, bestimmte Themen im Buch schneller zu finden.

E.3 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 nichtproportionalen 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.

E.4 Programmtexte und Lösungen zum Download

Das Buch enthält viele kleine Beispielprogramme. Sie sind als »Starterprojekte« gedacht und sollen Sie ermuntern, den Code weiterzuentwickeln und selbst etwas Neues auszuprobieren.

Der Code aller Beispielprogramme, die Lösungen zu den Übungen sowie ein Glossar stehen Ihnen auf der Webseite des Verlags unter www.mitp.de/0328 zum Download zur Verfügung.

E.5 Fragen und Feedback

Unsere Verlagsprodukte werden mit großer Sorgfalt erstellt. Sollten Sie trotzdem einen Fehler bemerken oder eine andere Anmerkung zum Buch haben, freuen wir uns über eine direkte Rückmeldung an lektorat@mitp.de.

Falls es zu diesem Buch bereits eine Errata-Liste gibt, finden Sie diese unter www.mitp.de/0328 im Reiter DOWNLOADS.

Wir wünschen Ihnen viel Erfolg und Spaß bei der Programmierung mit Python!

Michael Weigend und das mitp-Lektorat



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 befähigt. 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, Cobol und Lisp) wurden in den 1950er Jahren entwickelt. Heute (im Jahre 2021) listet Wikipedia 358 Programmiersprachen auf.

Die erste Python-Version wurde 1990 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 2021) 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 3.9, die im Oktober 2020 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.

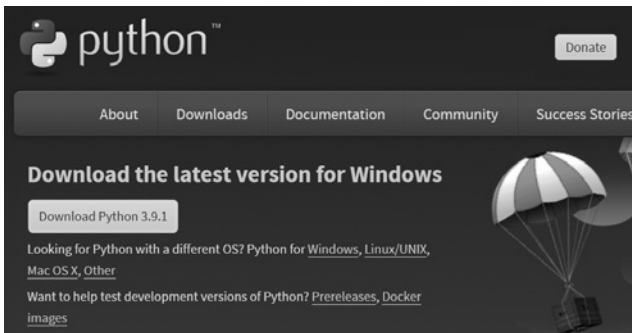


Abb. 1.1: Download-Seite von Python

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

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 3.9.0
```

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 python3.9
```

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.



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

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 `$`.

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 *Python-Shell* und einem *Editor*:

- In der Python-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 Python-Shell. Sie sehen ein Anwendungsfenster wie in Abbildung 1.4.

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.

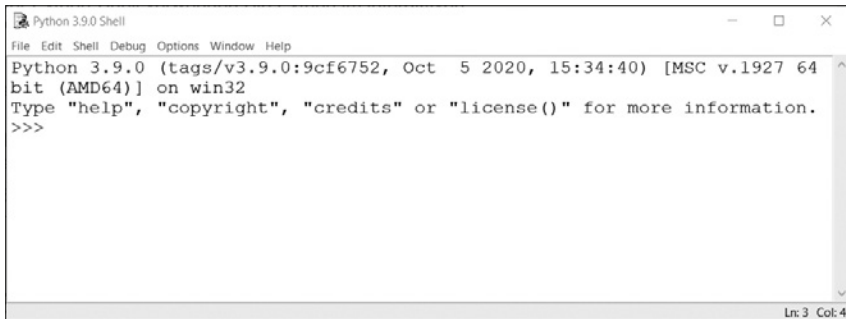


Abb. 1.4: Die Python-Shell

1.8 Hotkeys für die Python-Shell

Es gibt zwei Tastenkombinationen (Hotkeys), die die Arbeit mit der Python-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 Programmentexten.

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«.

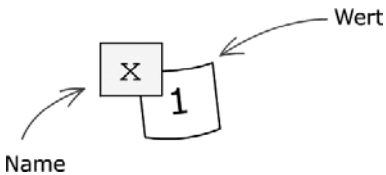


Abb. 1.5: Variable als Name für eine Zahl

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.

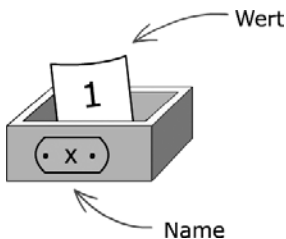


Abb. 1.6: Variable als Behälter

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
```

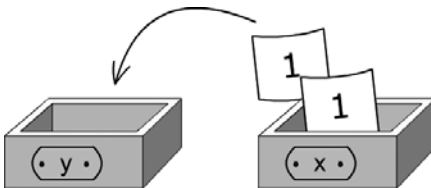


Abb. 1.7: Veranschaulichung einer Werteübertragung

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 Regeln 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>else</code>	<code>except</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 – die Python-Shell als Taschenrechner

Sie können die Python-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
622301527786114170714406405378012424059025216872116713310111
661478969883403538344118394482312571361695696658955512248212
47160434722900390625
>>>
```

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.

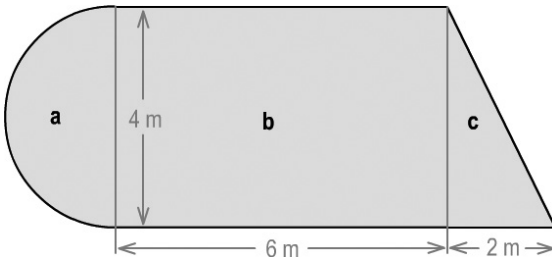


Abb. 1.8: Eine Fläche, die aus mehreren einfachen Flächen zusammengesetzt ist

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 Übungen

Übung 1: Ausdrucksanweisungen



Welche Ergebnisse liefern die folgenden Ausdrucksanweisungen? 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>		

Anweisung	Ergebnis	Erklärung
<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
<code>a = b = 1</code>	1	1	
<code>a = a + 2</code>			
<code>b += 1</code>			
<code>c = a + b</code>			
<code>c = 2 * c</code>			
<code>a, b = 2.0, 2</code>			
<code>c = a / b</code>			



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

Die Lösungen zu diesen Übungen können Sie durch Experimente in der Python-Shell selbst ermitteln. Außerdem finden Sie alle Lösungen in einer Datei, die Sie von der Website des mitp-Verlages herunterladen können: www.mitp.de/0328

1.12 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

.pyw 184

A

Aggregation 249
 Agile Software-Entwicklung 265
 and 79
 Änderbarkeit 266, 275
 Annotation 103
 Anweisung
 aufteilen 77
 leere 76
 Anwendungsfenster 182
 Titel ändern 183
 Arbeitsspeicher 147
 Argument 94
 assert 227
 Assoziation 249
 Asynchron 181
 Attribut 236
 Zugriff auf 239
 Ausdruck 23, 29
 regulärer 174
 Auskommentieren 65, 228

B

BarCamp 280
 Basisklasse 252
 Beck, Kent 265
 Bedingte Wiederholung 82
 Bedingung 69, 77
 erfüllt 73
 nicht erfüllt 73
 Benutzeroberfläche 181
 Bild
 als Schaltfläche 213
 verändern 216
 Bildformate 213
 Binärdatei 150
 laden 157
 öffnen 155
 speichern 156

Bit 1050
 Block 76
 Bogenmaß 116
 bool 41, 78
 Boolescher Operator 79
 Boolescher Wert 78
 Breakpoint 233
 Bug 63
 Button 182, 188
 Byte 150
 Bytestring 151
 in String umwandeln 151

C

Calltip 101
 Canvas 182
 Casting 44
 Checkbutton 182
 close() 149
 Codierung 148
 Community 279
 Compiler 17
 Cursor
 Position 197

D

Datei
 aus dem Internet 162
 Modus 148
 öffnen 147
 Öffnen-Dialog 203
 speichern 149
 Speichern-Dialog 203
 Daten
 dauerhaft speichern 147
 Datenverarbeitung 53
 Datum 118
 Debugger 230
 Befehle 231
 Debugging 63, 225
 Typen 226
 Variablen 228

Default	97
Deklaration	46
Deselektieren	199
Deserialisierung	157
Dialogbox	203
Dictionary	40, 140
ändern	141
Operationen	141
Divide and conquer	228
Division	29
Docstring	103
doctest	277
Dokumentation	
automatische	104
Dynamische Typisierung	46

E

Editor	21, 49
Einrückung	54, 70, 74, 76
elif	73
elif-Klausel	74
else	72
else-Klausel	71
Emoji	165
Encoding	166
Endlosrekursion	108
Endloswiederholung	86
endwith()	167, 168
Entry	182
Entry-Widget	193
Entscheidungsbaum	90
Entwicklungsprozess	264
Entwicklungsumgebung	
IDLE	21
Epoche	276
Erweiterte Zuweisung	28
Escape-Sequenz	166
EVA-Prinzip	52
Event	188
EventHandler	188, 190
Exponentialschreibweise	38
Extreme Programming	265

F

Fallunterscheidung	73
False	41

Fehler

logischer	64
semantischer	225
Syntaxfehler	64
Fehlersuche	63
Tipps	64
float	37
flush()	149
for	87
Formatstring	175
Funktion	93
Annotation	103
Benennung	103
Definition	94
rekursive	93, 107

Funktionsaufruf

verschachtelt	57
Funktionsdefinition	
verbessern	102
Funktionskopf	94
Funktionskörper	94

G

Ganzzahl	37
Gleitkommazahl	37
Grafische Benutzungsoberfläche	181
GUI	
Siehe Grafische Benutzungsoberfläche	

H

Haltepunkt	233
Hashing	141
Hashtag	54
help()	104
Hexadezimal	165, 186
History	22
Hotkey	22

I

Icon	213
IDLE	21
Debugger	231
Sternchen	52
if	69
if-Anweisung	
verschachtelt	74
if-Klausel	70

Imperativer Programmierstil	236
Implementierung	245, 247
Import	
Funktion eines Moduls	114
Konstante	116
Module	113
Index	197
Initialisierung	84
Initialisierungsmethode	238
input()	62
Installation	18
Instanz	236, 239
Instanziieren	239
int	37
Interaktiver Modus	20
Interaktives Programm	123
Internet	
Daten laden	162
Interpreter	17
Item	42
Iteration	87, 265, 267
Liste	130
Tupel	130
J	
JPEG	219
K	
keys()	141
Kinokarte	72
Klasse	236, 247
abgeleitete	252
definieren	237
Klassenattribut	242, 244
zugreifen auf	244
Klassendiagramm	237
Kollektion	42
Vorkommen	42
Kommandozeile	20
Kommentar	54
Konferenz	280
Konkatenation	44, 131
Konstante	115
importieren	116
Kontrollstruktur	69
Kontrollvariable	200
Kurs	280
Kurze Zeichenkette	38

L

Label	182
Lastenheft	264
Laufvariable	88
Laufzeit	259
Laufzeitfehler	157
Layout-Manager	190
Lesezugriff	148
Lineares Programm	54
list	40
List comprehension	230
Liste	40, 127, 132
aus Kollektion	135
auspacken	130
aus Zahlen	135
erzeugen	135
Listenabstraktion	136
Literal	35
Logging	228
Logischer Fehler	64
lower()	168

M

Magische Methode	245, 247
math	115
Menge	40, 42
Mengenabstraktion	199
Metapher	241
Methode	131, 133, 236
aufrufen	240
definieren	238
magische	245, 247
überschriebene	253
Modell	127
Modul	113
erstellen	120
importieren	113
testen	122
Vorteile	125
Modulo	30
Modus	148
interaktiver	20
Multiplikation	29

N

Namensraum	96
Nebenläufigkeit	206

not	79
Notizbuch	267
NumPy	219

O

Objekt	133, 236
anonymes	251
neues erzeugen	239
variable Anfangswerte	241
Wert ausgeben	243
Objektorientierte Programmierung	235
Objektvariable	236
Oktett	151
OOP	
Siehe Objektorientierte Programmie- rung	
Operator	78
boolescher	79
überladen	245
or	79

P

Packer	190
Optionen	192
Pair-Programmierung	266
Parameter	94, 139
optionaler	97
Parameterliste	94
pass	76
Passwortkontrolle	70
Performance	125, 259, 263
Performance-Analyse	263
Pfad	148
absoluter	151
relativer	152
PhotoImage	182
Methoden	216
pickle	155
PIL.Image-Objekt	219
Pivot	228
Platzhalter	175
Namen	176
Polymorphie	245
Positionsargument	101
Potenz	30
print()	58, 105
Profiler	263

Programm	
ausführen	58
interaktives	123
linear	54
Programmabsturz	157
Programm ausführen	58
Doppelklick	60
Kommandozeile	58
Linux	62
macOS	62
Programmierstil	
imperativer	236
Programmierung	
objektorientierte	235
Programmierungswettbewerb	281
Programmlauf	
abbrechen	86
Promptstring	53
Prozedur	99
Pycon	280
PyPI	125
Python Academy	280
Python-Homepage	18
Python Imaging Library	219
Python Package Index	125

Q

Quicksort	228, 259
Zeitkomplexität	262

R

Radiobutton	182, 199
vorselektieren	201
random	116
range()	89
range-Objekt	89
Raspberry Pi	280
Read-Eval-Print-Loop	21
Refactoring	266, 270
Regulärer Ausdruck	174
Rekursion	93, 107
endlose	108
Release Planning	265, 267
REPL	
Siehe Read-Eval-Print-Loop	
replace()	168
Repository	125
return-Anweisung	94

S

Schaltfläche. Siehe Button	
Schleife	82
Schlüssel	41, 140
Schlüssel-Wert-Paar	141
Schlüsselwort	28
Schlüsselwortargument	101
Vorteile	101
Schreibzugriff	148
Schrifttyp	186
Seiteneffekt	215
Selektieren	199
self	238
Semantik	16
Sequenz	42, 127
änderbare	127
komplexe	128
Konkatenation	44
nicht änderbare	128
Operationen	43, 130
Zugriff über den Index	43
Sequenziell	205
Serialisierung	156
set	40
Shebang	62
Shell	20
Shortcut	22
Signatur	105
Skript	49
Software Engineering	264
Sortieren	
Strings	134
Zahlen	134
Speichern	
String	149
split()	168
Standardausgabe	228
Standard-Typ	37
Story	265, 267
implementieren	267
str	38
Straight Selection	260
Stream	147
String	38
kurze Zeichenkette	38
mit Variablen	175
Operationen	167

Platzhalter	175
speichern	149
Stringformatierung	175
strip()	168
Studium	280
Suchfunktion	137
Synchron	181
Syntax	16
Syntaxfehler	63
Syntax-Highlighting	50
Syntaxregel	103

T

Tastenkombination	22
Teile und herrsche	228
Term	29
Test Driven Development	266, 277
Text	182
Textautomat	176
Textposition	197
Textumbruch	196
Textverarbeitung	165
Codierung	166
Text-Widget	195
Methoden	196
Thread	207
neuen starten	207
Ticketbuchungssystem	281
time	118
Tk	182
Tracing	228
True	41
Tupel	128
auspacken	130
definieren	130
Typ	247
abfragen	36
Vorbedingung testen	226
Typ-Hierarchie	37
Typisierung	
dynamische	46
Typumwandlung	44

U

Überladen	245
Uhrzeit	118
UML	237, 249, 253
UML-Klassendiagrammen	264

UML-Klassensymbol	245
Unicode	165
Unix-Zeit	118
Unterklasse	252
upper()	168
URL	162
utf-8	148, 166

V

values()	141
Variable	25
globale	107
in Texten	175
lokale	107
prüfen	230
Variablenname	25, 27
Verbinden	
explizit	77
implizit	77
Vererbung	252
Vergleichskette	79
Verzeichnisbaum	151
Vorbedingung	225
testen	227

W

Wahrheitswert	41, 78, 226
Währungsrechner	201
Webcam	220
Web-Scraping	171
Website	
Informationen auslesen	171
Wert	140
boolescher	78
Werteübergabe	96

Wettbewerb	281
while	82
Widget	182
Abstand	191
Aussehen ändern	185
Bilddatei	213
einfügen	184
Farbe ändern	186
Größe ändern	187
Master	184
Methoden	188
nebeneinander	191
Optionen	185
Texteingabe	193, 195
Wiederholung	89
bedingte	82
Wikimedia Commons	223
Wörter zählen	170
write()	149
Würfel	117

Z

Zeichen	
Codierung	165
Zeichenkette	38
kurze	38
Zeile	77
verbinden	77
Zeitkomplexität	262
Zufallsfunktionen	116
Zuständigkeit	236
Zuweisung	24
erweitert	28
Zuweisungsoperator	25