Michael Weigend

PYTHON 3 Schnelleinstieg

Programmieren lernen in 14 Tagen

Einfach und ohne Vorkenntnisse zum Profi

Zahlreiche Praxisbeispiele und Übungen

Für Annelie Margarete

Michael Weigend

Python 3

Schnelleinstieg

Programmieren lernen in 14 Tagen
Einfach und ohne Vorkenntnisse zum Profi

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

ISBN 978-3-7475-0329-4
1. Auflage 2021

www.mitp.de
E-Mail: mitp verlag@sigloch.de
Telefon: +49 7953 / 7189 - 079
Telefax: +49 7953 / 7189 - 082

Lektorat: Janina Bahlmann
Sprachkorrektorat: Petra Heubach-Erdmann
Covergestaltung: Janina Bahlmann, Christian Kalkert
Covergrafik & Icons: Tanja Wehr, sketchnotelovers
Abbildungen & Grafiken: Michael Weigend
Satz: Petra Kleinwegen

Inhalt

Einleitung

Willkommen zu Python!

Datentypen – die Python-Typ-Hierarchie

Interaktive Programme

Kontrollstrukturen

Funktionen

Mit Modulen arbeiten

Mit Kollektionen modellieren

Daten speichern

Textverarbeitung

Grafische Benutzungsoberflächen

Grafik programmieren

Fehler finden und vermeiden

Objektorientierte Programmierung

Professionelle Software-Entwicklung

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 Benutzungsoberflächen. 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 E-Books zum Download

Geben Sie den untenstehenden Code unter

https://mitp.code-load.de/

in das Textfeld ein und klicken Sie auf Einlösen, um das E-Book (als PDF und EPUB) herunterzuladen.

```
exbf311a71
```

E.6 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

1 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, 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.

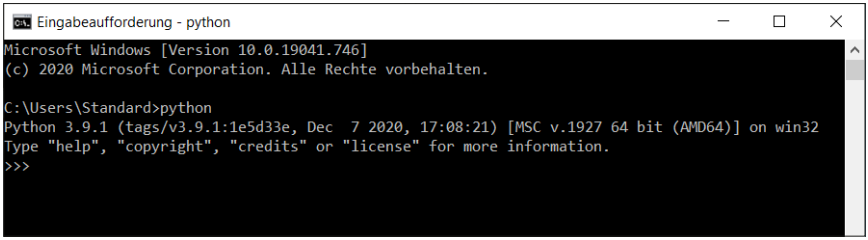

Abb. 1.3: Aufruf des Python-Interpreters in einem Konsole-Fenster (Eingabeaufforderung) unter Windows

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.

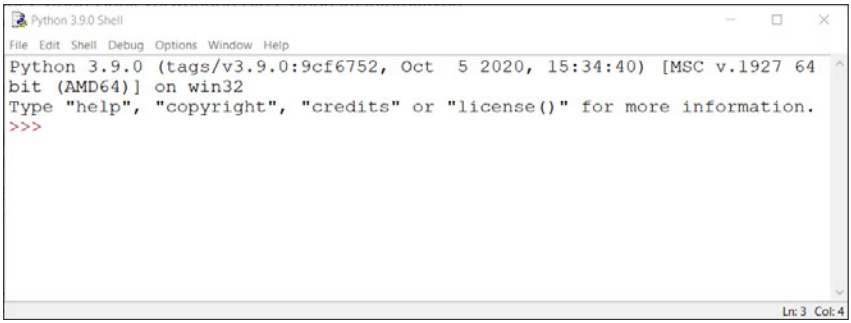

Abb. 1.4: Die Python-Shell

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 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«.

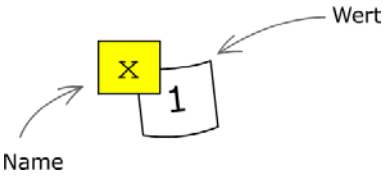

Abb. 1.5: Variable als Name für eine Zahl

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.

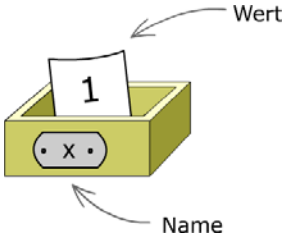

Abb. 1.6: Variable als Behälter

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
```

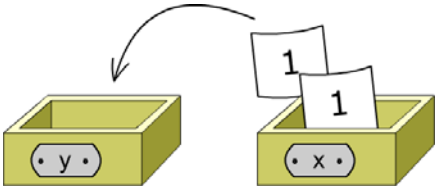

Abb. 1.7: Veranschaulichung einer Werteübertragung

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 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:

```
and       as        assert    break     class     continue
def       del       elif      else      except    False
finally   for       from      global    import    if
in        is        lambda    None      nonlocal  not
or        pass      raise     return    True      try
while     with      yield
```

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
```