Michael Weigend

PYTHON 3

FÜR STUDIUM UND AUSBILDUNG

Einfach lernen und professionell anwenden

Inhaltsverzeichnis

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

Kapitel 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 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.

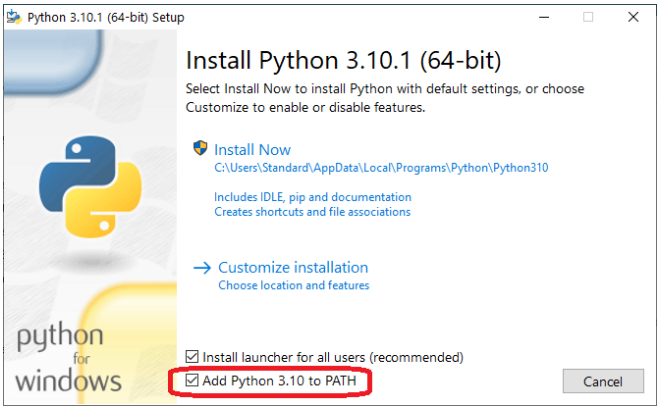

Abb. 1.2: Installation von Python unter Windows

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 $.

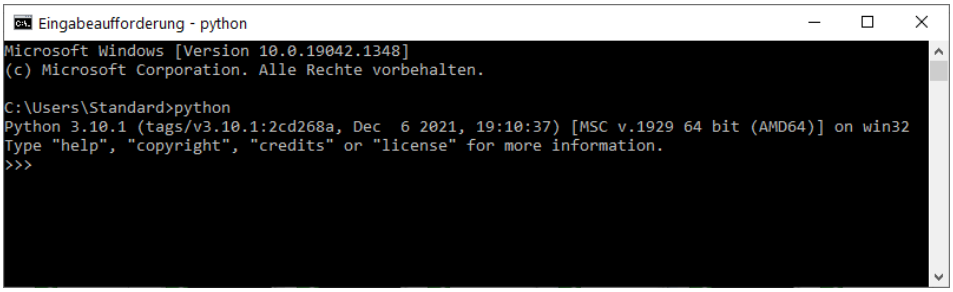

Abb. 1.3: Aufruf des Python-Interpreters in einem Konsole-Fenster (Eingabeaufforderung) unter Windows

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.

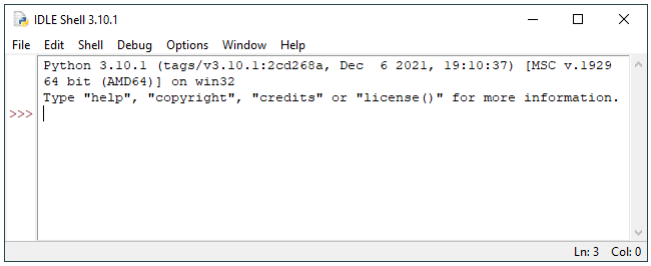

Abb. 1.4: Die IDLE-Shell

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«.

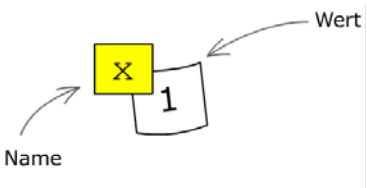

Abb. 1.5: Variable als Name für eine Zahl

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.

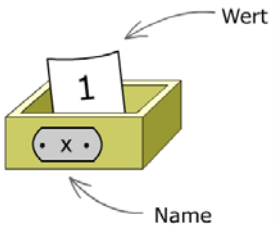

Abb. 1.6: Variable als Behälter

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
```

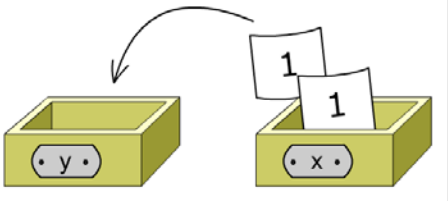

Abb. 1.7: Veranschaulichung einer Werteübertragung

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:

```
and         as          assert      break       class
continue    def         del         elif        elseexcept
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
```

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
622301527786114170714406405378012424059025216872116713310111661478969
698834035383441183944823125713616956966589555122482124716043472290039
390625
>>>
```

Spezieller 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.

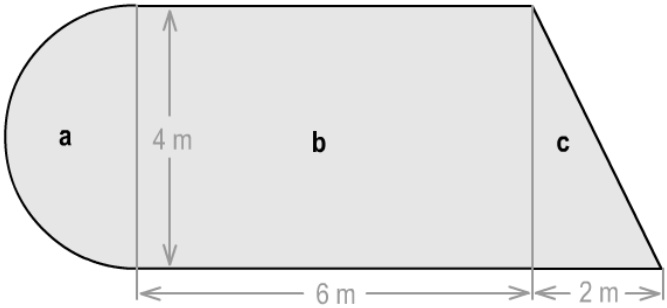

Abb. 1.8: Eine Fläche, die aus mehreren einfachen Flächen zusammengesetzt ist

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
```

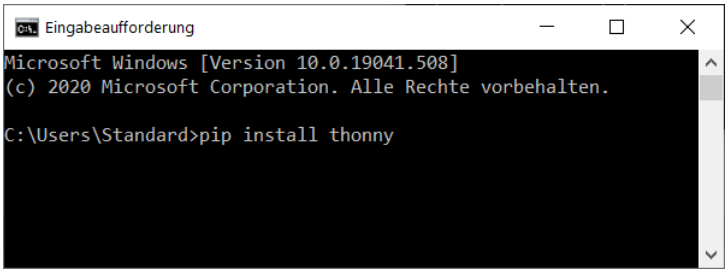

Abb. 1.9: In der Konsole mit pip ein Python-Programm installieren (Windows)

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.

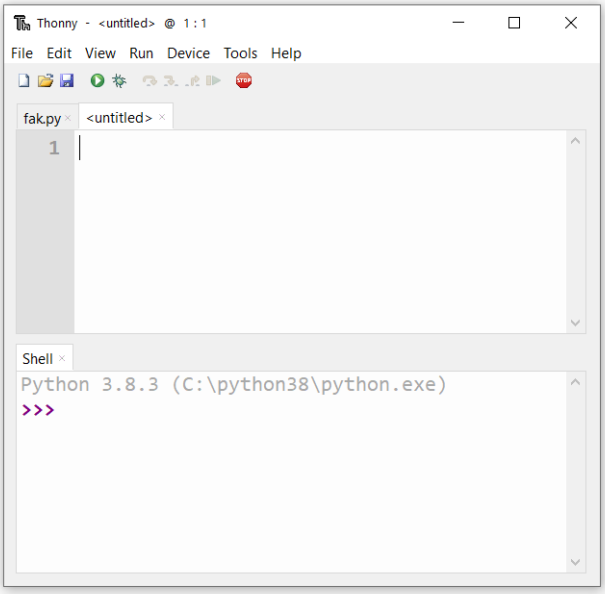

Abb. 1.10: Die Benutzungsoberfläche von Thonny. Editor (oben) und Shell (unten) sind in einem Fenster.

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.

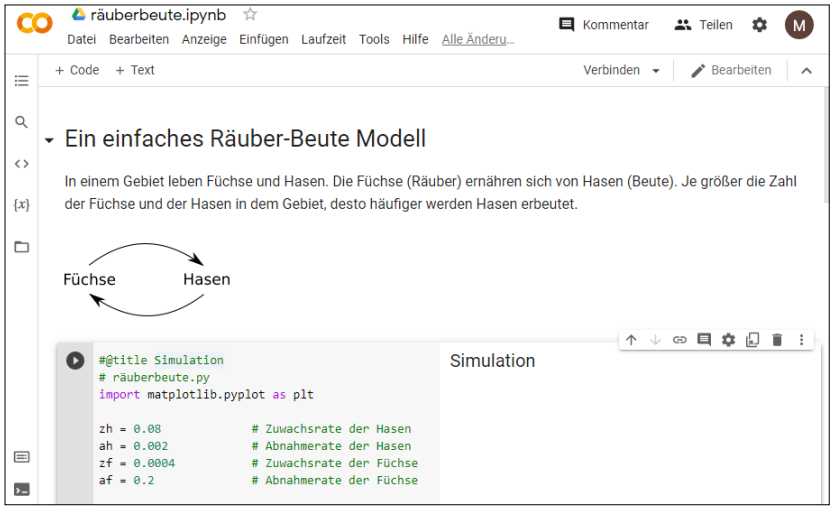

Abb. 1.11: Jupyter-Notebook mit Text, Bild und Python-Code

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 Jupiter-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 definiert die Schreibweise und die die Bedeutung von Anweisungen der Programmiersprache.
- Ein Python-Programm wird von einem ausgeführt.
- Die Entwicklungsumgebung IDLE besteht aus der *IDLE-Shell* und einem *Programmeditor*.
- In der IDLE-Shell kann man einzelne Anweisungen ausprobieren, im Programmeditor schreibt man ein komplettes Python-Programm, speichert es ab und testet es.
- In einer wird einem Namen ein Wert (z.B. eine Zahl) zugeordnet.
- Für gibt es Syntaxregeln: Ein Name besteht aus Buchstaben, Ziffern und Unterstrichen _ und muss mit einem Buchstaben oder Unterstrich beginnen.
- Python enthält , 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
`2 + 3 * 2`		
`(2 + 3) * 2`		
`10 / 2`		
`10 // 3`		
`-10 // 3`		
`10 % 3`		
`11 % 3`		
`12 % 3`		
`2 ** 3`		
`4 ** 0.5`		
`2 + - 2`		
`1,5 * 2`		
`1.5 * 2`		
`2 + * 3`		
`round(1.23)`		
`1 > 2`		
`1 == 1.0`		

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