

Thomas Theis

Einstieg in Python

Ideal für Programmieranfänger

- + Schritt für Schritt eigene Programme entwickeln
- + Mit vielen Beispielen und Übungen
- + GUI, OOP, Datenbank- und Internetanwendungen u. v. m.



Gedruckt in Deutschland
Ohne Folienkaschierung
Mineralölfreie Druckfarben



Alle Codebeispiele zum Download



Rheinwerk
Computing

Thomas Theis

Einstieg in Python

Ideal für Programmieranfänger

7., aktualisierte Auflage 2022



Impressum

Dieses E-Book ist ein Verlagsprodukt, an dem viele mitgewirkt haben, insbesondere:

Lektorat Anne Scheibe

Korrektorat Petra Schomburg, Hilter-Borgloh

Covergestaltung Mai Loan Nguyen Duy

Herstellung E-Book Denis Schaal

Satz E-Book SatzPro, Krefeld

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

ISBN 978-3-8362-8832-3

7., aktualisierte Auflage 2022

© Rheinwerk Verlag GmbH, Bonn 2022

Liebe Leserin, lieber Leser,

mit Python haben Sie eine gute Wahl getroffen: Python ist vielseitig, leicht zu lernen und bietet Ihnen später das Potenzial für anspruchsvollere Projekte. Die hier erworbenen Python-Kenntnisse können Sie zudem im großen Bereich KI gewinnbringend einsetzen, und sie sind auch für Maker unverzichtbar.

Dieses Buch wird Sie bei Ihrem Einstieg in Python von Anfang an begleiten. Schritt für Schritt lernen Sie das Programmieren an Beispiel-Projekten. Sie werden schnell Ihre ersten eigenen Programme schreiben, selbst wenn Sie bisher noch nicht programmiert haben. Übungsaufgaben helfen Ihnen dabei, Ihr neu gewonnenes Wissen zu testen und weiter zu vertiefen.

Dieses Buch wurde mit großer Sorgfalt geschrieben, geprüft und produziert. Sollte dennoch einmal etwas nicht so funktionieren, wie Sie es erwarten, freue ich mich, wenn Sie sich mit mir in Verbindung setzen. Ihre Kritik und konstruktiven Anregungen, aber natürlich auch Ihr Lob sind uns jederzeit herzlich willkommen!

Viel Spaß mit Python wünscht Ihnen nun

Ihre Anne Scheibe

Lektorat Rheinwerk Computing

anne.scheibe@rheinwerk-verlag.de

www.rheinwerk-verlag.de

Rheinwerk Verlag • Rheinwerkallee 4 • 53227 Bonn

Inhaltsverzeichnis

Aus dem Lektorat
Inhaltsverzeichnis

Materialien zum Buch

1 Einführung

- 1.1 Vorteile von Python
- 1.2 Verbreitung von Python
- 1.3 Aufbau des Buchs
- 1.4 Übungen
- 1.5 Installation unter Windows
- 1.6 Installation unter Ubuntu Linux
- 1.7 Installation unter macOS

2 Erste Schritte

- 2.1 Python als Taschenrechner
 - 2.1.1 Eingabe von Berechnungen
 - 2.1.2 Addition, Subtraktion und Multiplikation

2.1.3 Division, Ganzzahldivision und Modulo

2.1.4 Rangfolge und Klammern

2.1.5 Variablen und Zuweisung

2.2 Erstes Programm

2.2.1 Hallo Welt

2.2.2 Eingabe eines Programms

2.3 Speichern und Ausführen

2.3.1 Speichern

2.3.2 Ausführen unter Windows

2.3.3 Ausführen unter Ubuntu Linux und unter macOS

2.3.4 Kommentare

2.3.5 Verkettung von Ausgaben

2.3.6 Lange Ausgaben

3 Programmierkurs

3.1 Ein Spiel programmieren

3.2 Variablen und Operatoren

3.2.1 Berechnung und Zuweisung

3.2.2 Ausgabe mit formatiertem String-Literal

3.2.3 Eingabe einer Zeichenkette

3.2.4 Eingabe einer Zahl

3.2.5 Spiel, Version mit Eingabe

3.2.6 Zufallszahlen

3.3 Verzweigungen

3.3.1 Vergleichsoperatoren

3.3.2 Verzweigung mit »if«

3.3.3 Spiel, Version mit Bewertung der Eingabe

3.3.4 Mehrfache Verzweigung

- 3.3.5 Bedingter Ausdruck
- 3.3.6 Logische Operatoren
- 3.3.7 Mehrere Vergleichsoperatoren
- 3.3.8 Spiel, Version mit genauer Bewertung der Eingabe
- 3.3.9 Verzweigung mit »match«
- 3.3.10 Rangfolge der Operatoren

3.4 Schleifen

- 3.4.1 Schleife mit »for«
- 3.4.2 Schleifenabbruch mit »break«
- 3.4.3 Schleifenfortsetzung mit »continue«
- 3.4.4 Geschachtelte Kontrollstrukturen
- 3.4.5 Spiel, Version mit »for«-Schleife und Abbruch
- 3.4.6 Schleife mit »for« und »range()«
- 3.4.7 Spiel, Version mit »range()«
- 3.4.8 Schleife mit »while«
- 3.4.9 Spiel, Version mit »while«-Schleife und Zähler
- 3.4.10 Kombinierte Zuweisungsausdrücke

3.5 Entwicklung eines Programms

3.6 Fehler und Ausnahmen

- 3.6.1 Basisprogramm
- 3.6.2 Fehler abfangen
- 3.6.3 Eingabe wiederholen
- 3.6.4 Spiel, Version mit Ausnahmebehandlung

3.7 Funktionen und Module

- 3.7.1 Einfache Funktionen
- 3.7.2 Funktionen mit einem Parameter
- 3.7.3 Funktionen mit mehreren Parametern
- 3.7.4 Funktionen mit Rückgabewert
- 3.7.5 Typhinweise

3.7.6 Spiel, Version mit Funktionen

3.8 Das fertige Spiel

4 Datentypen

4.1 Zahlen

- 4.1.1 Ganze Zahlen
- 4.1.2 Zahlen mit Nachkommastellen
- 4.1.3 Typ ermitteln
- 4.1.4 Exponentialoperator **
- 4.1.5 Rundung und Konvertierung
- 4.1.6 Winkelfunktionen
- 4.1.7 Weitere mathematische Funktionen
- 4.1.8 Komplexe Zahlen
- 4.1.9 Bitoperatoren
- 4.1.10 Brüche

4.2 Zeichenketten

- 4.2.1 Eigenschaften
- 4.2.2 Operatoren
- 4.2.3 Slices
- 4.2.4 Änderbarkeit
- 4.2.5 Suchen und Ersetzen
- 4.2.6 Leerzeichen entfernen
- 4.2.7 Text zerlegen
- 4.2.8 Konstanten
- 4.2.9 Datentyp »bytes«

4.3 Listen

- 4.3.1 Eigenschaften und Operatoren
- 4.3.2 Mehrdimensionale Listen

- 4.3.3 Änderbarkeit
- 4.3.4 Methoden
- 4.3.5 List Comprehension

4.4 Tupel

4.5 Dictionarys

- 4.5.1 Eigenschaften, Operatoren und Methoden
- 4.5.2 Dynamische Views

4.6 Sets, Mengen

- 4.6.1 Eigenschaften, Operatoren und Methoden
- 4.6.2 Mengenlehre

4.7 Wahrheitswerte und Nichts

- 4.7.1 Wahrheitswerte »True« und »False«
- 4.7.2 Nichts, »None«

4.8 Referenz, Identität und Kopie

- 4.8.1 Referenz und Identität
- 4.8.2 Ressourcen sparen
- 4.8.3 Objekte kopieren

5 Weiterführende Programmierung

5.1 Allgemeines

- 5.1.1 Kombinierte Zuweisungsoperatoren
- 5.1.2 Anweisung in mehreren Zeilen
- 5.1.3 Eingabe mit Hilfestellung
- 5.1.4 Anweisung »pass«
- 5.1.5 Funktionen »eval()« und »exec()«

5.2 Ausgabe und Formatierung

- 5.2.1 Funktion »print()«
- 5.2.2 Formatierung von Zahlen mit Nachkommastellen
- 5.2.3 Formatierung von ganzen Zahlen
- 5.2.4 Formatierung von Zeichenketten

5.3 Funktionen für Iterables

- 5.3.1 Funktion »zip()«
- 5.3.2 Funktion »map()«
- 5.3.3 Funktion »filter()«

5.4 Verschlüsselung

5.5 Fehler und Ausnahmen

- 5.5.1 Allgemeines
- 5.5.2 Syntaxfehler
- 5.5.3 Laufzeitfehler
- 5.5.4 Logische Fehler und Debugging
- 5.5.5 Fehler erzeugen
- 5.5.6 Unterscheidung von Ausnahmen

5.6 Funktionen

- 5.6.1 Variable Anzahl von Parametern
- 5.6.2 Benannte Parameter
- 5.6.3 Optionale Parameter
- 5.6.4 Mehrere Rückgabewerte
- 5.6.5 Übergabe von Kopien und Referenzen
- 5.6.6 Namensräume
- 5.6.7 Rekursive Funktionen
- 5.6.8 Lambda-Funktion
- 5.6.9 Funktion als Parameter

5.7 Eingebaute Funktionen

- 5.7.1 Funktionen »max()«, »min()« und »sum()«
- 5.7.2 Funktionen »chr()« und »ord()«
- 5.7.3 Funktionen »reversed()« und »sorted()«

5.8 Weitere mathematische Module

- 5.8.1 Funktionsgraphen zeichnen
- 5.8.2 Mehrere Teilzeichnungen
- 5.8.3 Eindimensionale Arrays und Vektoren
- 5.8.4 Mehrdimensionale Arrays und Matrizen
- 5.8.5 Signalverarbeitung
- 5.8.6 Statistikfunktionen

5.9 Eigene Module

- 5.9.1 Eigene Module erzeugen
- 5.9.2 Standard-Import eines Moduls
- 5.9.3 Import eines Moduls mit Umbenennung
- 5.9.4 Import von Funktionen

5.10 Parameter der Kommandozeile

5.11 Programm »Bruchtraining«

- 5.11.1 Der Ablauf des Programms
- 5.11.2 Hauptprogramm
- 5.11.3 Eine leichte Aufgabe
- 5.11.4 Eine mittelschwere Aufgabe
- 5.11.5 Eine schwere Aufgabe

6 Objektorientierte Programmierung

6.1 Was ist OOP?

- 6.2 Klassen, Objekte und eigene Methoden
- 6.3 Besondere Member
- 6.4 Operatormethoden
- 6.5 Referenz, Identität und Kopie

- 6.6 Vererbung
- 6.7 Mehrfachvererbung
- 6.8 Datenklassen
- 6.9 Enumerationen
- 6.10 Spiel, objektorientierte Version

7 Verschiedene Module

7.1 Datum und Uhrzeit

- 7.1.1 Funktionen
- 7.1.2 Rechnen mit Zeitangaben
- 7.1.3 Programm anhalten
- 7.1.4 Spiel, Version mit Zeitmessung
- 7.1.5 Spiel, objektorientierte Version mit Zeitmessung

7.2 Warteschlangen

- 7.2.1 Klasse »SimpleQueue«
- 7.2.2 Klasse »LifoQueue«
- 7.2.3 Klasse »PriorityQueue«
- 7.2.4 Klasse »deque«

7.3 Multithreading

- 7.3.1 Wozu dient Multithreading?
- 7.3.2 Erzeugung eines Threads
- 7.3.3 Identifizierung eines Threads
- 7.3.4 Gemeinsame Daten und Objekte
- 7.3.5 Threads und Exceptions

7.4 Reguläre Ausdrücke

- 7.4.1 Suchen von Teiltexten
- 7.4.2 Ersetzen von Teiltexten

7.5 Audioausgabe

8 Dateien

8.1 Dateitypen

8.2 Öffnen und Schließen einer Datei

8.3 Textdateien

8.3.1 Schreiben einer Textdatei

8.3.2 Lesen einer Textdatei

8.3.3 CSV-Datei schreiben

8.3.4 CSV-Datei lesen

8.4 Dateien mit festgelegter Struktur

8.4.1 Formattiertes Schreiben

8.4.2 Lesen an beliebiger Stelle

8.4.3 Schreiben an beliebiger Stelle

8.5 Serialisierung mit »pickle«

8.5.1 Objekte in Datei schreiben

8.5.2 Objekte aus Datei lesen

8.6 Datenaustausch mit JSON

8.6.1 JSON-Objekte in Datei schreiben

8.6.2 JSON-Objekte aus Datei lesen

8.7 Bearbeitung mehrerer Dateien

8.7.1 Funktion »glob.glob()«

8.7.2 Funktion »os.scandir()«

8.8 Informationen über Dateien

8.9 Dateien und Verzeichnisse verwalten

8.10 Beispielprojekt Morsezeichen

8.10.1 Morsezeichen aus Datei lesen

8.10.2 Ausgabe auf dem Bildschirm

8.10.3 Ausgabe mit Tonsignalen

8.11 Spiel, Version mit Highscore-Datei

8.11.1 Eingabebeispiel

8.11.2 Aufbau des Programms

8.11.3 Code des Programms

8.12 Spiel, objektorientierte Version mit Highscore-Datei

9 Internet

9.1 Laden und Senden von Internetdaten

9.1.1 Daten lesen

9.1.2 Daten kopieren

9.1.3 Daten senden

9.2 Webserver-Programmierung

9.2.1 Erstes Programm

9.2.2 Beantworten einer Benutzereingabe

9.2.3 Formularelemente mit mehreren Werten

9.2.4 Typen von Formularelementen

9.3 Browser aufrufen

9.4 Spiel, Version für das Internet

9.4.1 Eingabebeispiel

9.4.2 Aufbau des Programms

9.4.3 Code des Programms

10 Datenbanken

10.1 Aufbau von Datenbanken

10.2 SQLite

10.2.1 Datenbank, Tabelle und Datensätze

10.2.2 Daten anzeigen

10.2.3 Daten auswählen, Operatoren

10.2.4 Operator »LIKE«

10.2.5 Sortierung der Ausgabe

10.2.6 Auswahl nach Eingabe

10.2.7 Datensätze ändern

10.2.8 Datensätze löschen

10.3 SQLite auf dem Webserver

10.4 MySQL

10.4.1 XAMPP und Connector/Python

10.4.2 Datenbank, Tabelle und Datensätze

10.4.3 Daten anzeigen

10.5 Spiel, Version mit Highscore-Datenbank

10.6 Spiel, objektorientierte Version mit Highscore-Datenbank

11 Benutzeroberflächen

11.1 Einführung

11.1.1 Erstes GUI-Programm

11.1.2 Anordnung von Widgets

11.2 Widget-Typen

11.2.1 Einzeiliges Eingabefeld

11.2.2 Versteckte Eingabe, Widget deaktivieren

11.2.3 Mehrzeiliges Eingabefeld

- 11.2.4 Listbox mit einfacher Auswahl
- 11.2.5 Listbox mit mehrfacher Auswahl
- 11.2.6 Spinbox
- 11.2.7 Radiobutton, Widget-Variable
- 11.2.8 Checkbutton
- 11.2.9 Schieberegler, Scale

11.3 Bilder und Mausereignisse

- 11.3.1 Bild einbetten und ändern
- 11.3.2 Mausereignisse

11.4 Geometrie-Manager »place«

- 11.4.1 Fenstergröße und absolute Position
- 11.4.2 Relative Position
- 11.4.3 Position ändern

11.5 Menüs, Messageboxen und Dialogfelder

- 11.5.1 Menüleisten
- 11.5.2 Kontextmenüs
- 11.5.3 Messageboxen
- 11.5.4 Eigene Dialogfelder

11.6 Zeichnungen und Animationen

- 11.6.1 Verschiedene Zeichnungsobjekte
- 11.6.2 Zeichnungsobjekte steuern
- 11.6.3 Zeichnungsobjekte animieren
- 11.6.4 Kollision von Zeichnungsobjekten

11.7 Spiel, GUI-Version

12 Benutzeroberflächen mit PyQt

- 12.1 Ein erstes Programm
- 12.2 Layout und Größe eines Anwendungsfensters
 - 12.2.1 Grid-Layout
 - 12.2.2 Größe des Anwendungsfensters
- 12.3 Widget-Typen
 - 12.3.1 Einzeiliges Eingabefeld
 - 12.3.2 Versteckte Eingabe, Deaktivieren von Widgets
 - 12.3.3 Mehrzeiliges Eingabefeld
 - 12.3.4 Liste mit einfacher Auswahl
 - 12.3.5 Liste mit mehrfacher Auswahl
 - 12.3.6 Combobox
 - 12.3.7 Spinbox
 - 12.3.8 Radiobutton
 - 12.3.9 Mehrere Gruppen von Radiobuttons
 - 12.3.10 Checkbox
 - 12.3.11 Slider
 - 12.3.12 Bilder, Formate und Hyperlinks

Anhang A

- A.1 Paketverwaltungsprogramm »pip«
- A.2 Erstellen von EXE-Dateien
- A.3 Installation von XAMPP
 - A.3.1 Installation von XAMPP unter Windows
 - A.3.2 Installation von XAMPP unter Ubuntu Linux
 - A.3.3 Installation von XAMPP unter macOS
- A.4 UNIX-Befehle

- A.4.1 Inhalt eines Verzeichnisses
- A.4.2 Verzeichnis anlegen, wechseln und löschen
- A.4.3 Datei kopieren, verschieben und löschen

Stichwortverzeichnis

Rechtliche Hinweise

Über den Autor

Materialien zum Buch

Auf der Webseite zu diesem Buch stehen folgende Materialien für Sie zum Download bereit:

- alle Beispielprogramme
- Lösungen der Übungsaufgaben

Gehen Sie auf <https://www.rheinwerk-verlag.de/5472>. Klicken Sie auf den Reiter MATERIALIEN. Sie sehen die herunterladbaren Dateien samt einer Kurzbeschreibung des Dateiinhalts. Klicken Sie auf den Button HERUNTERLADEN, um den Download zu starten. Je nach Größe der Datei (und Ihrer Internetverbindung) kann es einige Zeit dauern, bis der Download abgeschlossen ist.

1 Einführung

In diesem Kapitel stelle ich Ihnen Python kurz vor. Sie lernen die Vorteile von Python kennen und erfahren, wie Sie Python unter Windows, unter Ubuntu Linux und unter macOS installieren.

1.1 Vorteile von Python

Python ist eine sehr einfach zu erlernende Programmiersprache und für den Einstieg in die Welt der Programmierung ideal geeignet. Trotz ihrer Einfachheit bietet diese Sprache auch die Möglichkeit, komplexe Programme für vielfältige Anwendungsbereiche zu schreiben.

Python eignet sich besonders zur schnellen Entwicklung umfangreicher Anwendungen. Diese Technik ist unter dem Stichwort RAD (*Rapid Application Development*) bekannt geworden. Python vereint zu diesem Zweck folgende Vorteile:

- Eine einfache, eindeutige Syntax: Python ist für alle, die in die Programmierung einsteigen, eine ideale Programmiersprache. Sie beschränkt sich auf einfache, klare Anweisungen und häufig auf einen einzigen möglichen Lösungsweg. Dieser prägt sich schnell ein und wird der Entwicklerin bzw. dem Entwickler vertraut.
- Klare Strukturen: Python verlangt vom Entwickler, in einer gut lesbaren Struktur zu schreiben. Die Anordnung der Programmzeilen ergibt zugleich die logische Struktur des Programms.

- Wiederverwendung von Code: Die Modularisierung, also die Zerlegung eines Problems in Teilprobleme und die anschließende Zusammenführung der Teillösungen zu einer Gesamtlösung, wird in Python sehr leicht gemacht. Die vorhandenen Teillösungen können unkompliziert für weitere Aufgabenstellungen genutzt werden, sodass Sie bald über einen umfangreichen Pool an Modulen verfügen.
- Objektbearbeitung: In Python werden alle Daten als Objekte gespeichert. Dies führt zu einer einheitlichen Behandlung für Objekte unterschiedlichen Typs. Andererseits erfolgt die physikalische Speicherung der Objekte von Python automatisch, also ohne Eingriff des Entwicklers. Dieser muss sich nicht um die Reservierung und Freigabe geeigneter Speicherbereiche kümmern.
- Interpreter/Compiler: Python-Programme werden unmittelbar interpretiert. Sie müssen nicht erst kompiliert und gebunden werden. Dies ermöglicht einen häufigen, schnellen Wechsel zwischen Codierungs- und Testphase.
- Unabhängigkeit vom Betriebssystem: Sowohl Programme, die von der Kommandozeile aus bedient werden, als auch Programme mit grafischen Benutzeroberflächen können auf unterschiedlichen Betriebssystemen (Windows, Linux, macOS) ohne Neuentwicklung und Anpassung eingesetzt werden.

1.2 Verbreitung von Python

Aufgrund seiner vielen Vorzüge gehört Python zu den beliebtesten Programmiersprachen. So wird es zum Beispiel innerhalb des Projekts *100-Dollar-Laptop*, das der Schulausbildung von Kindern in aller Welt dient, für die Benutzeroberfläche verwendet. Aber auch in zahlreichen großen Unternehmen wird Python eingesetzt, hier ein paar Beispiele:

- YouTube wurde zum großen Teil mithilfe von Python entwickelt.
- NASA nutzt Python zur Softwareentwicklung im Zusammenhang mit den Space-Shuttle-Missionen.
- Industrial Light & Magic: Auch Hollywood setzt auf Python – die Produktionsfirma ILM (Star Wars, Indiana Jones, Fluch der Karibik) nutzt es zum Beispiel bei der Entwicklung von Spezialeffekten.
- Honeywell: Python wird weltweit in vielen Firmen zur allgemeinen Hardware- und Softwareentwicklung eingesetzt.

1.3 Aufbau des Buchs

Das vorliegende Buch führt Sie in die Programmiersprache Python in der aktuellen Version 3.10 ein, die im Oktober 2021 erschienen ist. Besonderer Wert wird darauf gelegt, dass Sie selbst praktisch mit Python arbeiten. Daher empfehle ich Ihnen, von Anfang an dem logischen Faden von Erklärungen und Beispielen zu folgen.

Erste Zusammenhänge werden in [Kapitel 2](#), »Erste Schritte«, anhand von einfachen Berechnungen vermittelt. Außerdem lernen Sie, ein Programm einzugeben, zu speichern und es unter den verschiedenen Umgebungen auszuführen.

Sie werden die Sprache spielerisch kennenlernen. Daher begleitet Sie ein selbst programmiertes Spiel durch das Buch. In dem Spiel sollen eine oder mehrere Kopfrechenaufgaben gelöst werden. Es wird mit dem »Programmierkurs« in [Kapitel 3](#) eingeführt und im weiteren Verlauf des Buchs kontinuierlich erweitert und verbessert.

Nach der Vorstellung der verschiedenen Datentypen mit ihren jeweiligen Eigenschaften und Vorteilen in [Kapitel 4](#), »Datentypen«, werden die Programmierkenntnisse in [Kapitel 5](#), »Weiterführende Programmierung«, vertieft. [Kapitel 6](#), »Objektorientierte Programmierung«, widmet sich der objektorientierten Programmierung mit Python. Einige nützliche Module zur Ergänzung der Programme werden in [Kapitel 7](#), »Verschiedene Module«, vorgestellt.

In [Kapitel 8](#), »Dateien«, und in [Kapitel 10](#), »Datenbanken«, lernen Sie, Daten dauerhaft in Dateien oder Datenbanken zu speichern. Python wird zudem in der Internetprogrammierung eingesetzt. Die Zusammenhänge zwischen Python und dem Internet vermittelt [Kapitel 9](#), »Internet«.

Sowohl Windows als auch Ubuntu Linux und macOS bieten komfortable grafische Benutzeroberflächen (GUIs). [Kapitel 11](#), »Benutzeroberflächen«, beschäftigt sich mit der GUI-Erzeugung mithilfe des Moduls `tkinter`. Dieses stellt eine Schnittstelle zwischen dem grafischen Toolkit *Tk* und Python dar. In [Kapitel 12](#) wird die GUI-Erzeugung mithilfe des Moduls `PyQt6` behandelt. Dieses beinhaltet die Elemente von PyQt in der Version 6. PyQt dient als Schnittstelle zwischen der Qt-Bibliothek und Python.

Für die Hilfe bei der Erstellung dieses Buchs bedanke ich mich bei dem gesamten Team des Rheinwerk Verlags, besonders bei Anne Scheibe.

1.4 Übungen

Im Buch finden Sie zahlreiche Übungsaufgaben. Ich empfehle Ihnen, sie unmittelbar zu lösen. Auf diese Weise können Sie Ihre Kenntnisse prüfen, bevor Sie zum nächsten Thema übergehen. Die Lösungen zu den Übungen finden Sie zusammen mit den Beispielprogrammen in den Materialien zum Buch. Beachten Sie dabei Folgendes:

- Es gibt für jedes Problem viele richtige Lösungen. Sieht Ihre Lösung nicht genauso aus wie die angegebene, ist das kein Problem. Betrachten Sie die angegebene Lösung vielmehr als Anregung und als Alternative.
- Bei der eigenen Lösung der Aufgaben wird sicherlich der eine oder andere Fehler auftreten – lassen Sie sich dadurch nicht entmutigen ...
- ... denn nur aus Fehlern kann man lernen. Auf die vorgeschlagene Art und Weise werden Sie Python wirklich erlernen – nicht allein durch das Lesen von Programmierregeln.

1.5 Installation unter Windows

Python ist eine frei verfügbare Programmiersprache, die unter verschiedenen Betriebssystemen eingesetzt werden kann. Die jeweils neuesten Python-Versionen können Sie von der offiziellen Python-Website <https://www.python.org> aus dem Internet laden. Zurzeit (im April 2022) sind das die Dateien *python-3.10.4.exe* für ein 32-Bit-System und *python-3.10.4-amd64.exe* für ein 64-Bit-System. Sie können sie unter Windows 8, Windows 10 und Windows 11 installieren.

Rufen Sie zur Installation unter einem 64-Bit-Windows die ausführbare Datei *python-3.10.4-amd64.exe* auf. Als Erstes müssen Sie bestätigen, dass Sie ein Programm installieren möchten, das nicht aus dem Microsoft Store stammt.

Wählen Sie die Option **CUSTOMIZE INSTALLATION** aus. Lassen Sie alle Häkchen bei den **OPTIONAL FEATURES** gesetzt. Das gilt besonders für das Paketverwaltungsprogramm *pip*, mit dessen Hilfe Sie später zusätzliche Module installieren können, siehe auch [Abschnitt A.1](#), »Paketverwaltungsprogramm ›pip‹«. Setzen Sie bei den **ADVANCED OPTIONS** zusätzlich das Häkchen bei **ADD PYTHON TO ENVIRONMENT VARIABLES**, damit Sie später die Möglichkeit haben, Python-Programme auf Ebene der Kommandozeile aus einem beliebigen Verzeichnis heraus zu starten. Wählen Sie das Installationsverzeichnis *C:\Python*.

Anschließend steht im Startmenü ein Eintrag zu **PYTHON 3.10**, siehe [Abbildung 1.1](#). Möchten Sie bestimmte Einstellungen der Installation im Nachhinein verändern, können Sie die Installationsdatei erneut aufrufen.



Abbildung 1.1 Startmenü mit Eintrag zu Python 3.10

Bei dem Programm IDLE im Startmenü handelt es sich um eine Entwicklungsumgebung, die selbst in Python geschrieben ist und mit der Sie im Folgenden Ihre Programme schreiben werden. Am besten ziehen Sie eine Verknüpfung zu IDLE auf den Desktop.

1.6 Installation unter Ubuntu Linux

Stellvertretend für andere Linux-Distributionen wird in diesem Buch Ubuntu Linux 21.10 genutzt. Python 3 ist unter Ubuntu Linux bereits installiert und darf auch nicht deinstalliert werden. In einem Terminal können Sie mithilfe des Befehls `python3 -v` (mit großem »V«) die aktuelle Versionsnummer ermitteln.

Zur Installation der Entwicklungsumgebung IDLE geben Sie in einem Terminal den folgenden Befehl ein: `sudo apt install idle3`. Das Programm IDLE können Sie anschließend mit dem Befehl `idle` starten.

1.7 Installation unter macOS

Die jeweils neuesten Python-Versionen können Sie von der offiziellen Python-Website <https://www.python.org> aus dem Internet laden. Zurzeit (im April 2022) ist das für macOS die Datei *python-3.10.4-macos11.pkg*.

Nach einem Doppelklick auf diese Datei startet die Installation. Nehmen Sie keine Änderungen vor, landet Python im Verzeichnis *Programme/Python 3.10*. Darin finden Sie einen Eintrag für die Entwicklungsumgebung IDLE, den Sie als Verknüpfung auf den Desktop ziehen können.

2 Erste Schritte

In diesem Kapitel werden Sie Python zum ersten Mal einsetzen – zunächst als Taschenrechner. Außerdem lernen Sie, ein Programm einzugeben, zu speichern und auszuführen.

2.1 Python als Taschenrechner

Sie können Python zunächst so ähnlich wie einen einfachen Taschenrechner benutzen. Dies erleichtert Ihnen den Einstieg in Python.

2.1.1 Eingabe von Berechnungen

Rufen Sie IDLE für Python auf, wie es am Ende der verschiedenen Installationen beschrieben wird. Die Entwicklungsumgebung IDLE kann sowohl als Editor zur Eingabe der Programme als auch als einfacher Taschenrechner genutzt werden.

Eine Darstellung von IDLE unter Windows sehen Sie in [Abbildung 2.1](#).

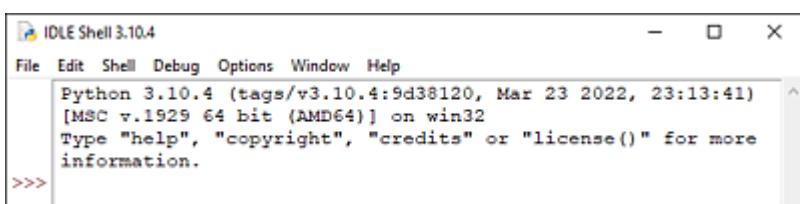


Abbildung 2.1 Python-Entwicklungsumgebung IDLE

Die Abbildungen in diesem Buch sind normalerweise für Python unter Windows erstellt worden. Sie gelten sinngemäß für Python unter Ubuntu Linux und Python unter macOS.

Zur Durchführung kleinerer Berechnungen müssen Sie keine vollständigen Programme schreiben und starten. Sie können die gewünschten Rechenoperationen direkt an der Eingabestelle, erkennbar an der Zeichenfolge `>>>`, eingeben. Eine abschließende Betätigung der Taste führt zur unmittelbaren Berechnung und Ausgabe des Ergebnisses.

Die dabei angewendeten Rechenregeln finden auch bei der Erstellung von Python-Programmen Verwendung. Bereits hier wird das EVA-Prinzip angewendet, so wie auch in vielen einfachen Programmen. Die Abkürzung EVA steht für *Eingabe*, *Verarbeitung* und *Ausgabe*. Der Benutzer gibt die Daten ein, diese werden mithilfe einer Berechnung verarbeitet, und anschließend wird das Ergebnis ausgegeben.

2.1.2 Addition, Subtraktion und Multiplikation

Es folgen die Eingabe, die Verarbeitung und die Ausgabe einiger einfacher Berechnungen:

```
>>> 41 + 7.5  
48.5  
>>> 12 - 18  
-6  
>>> 7 * 3  
21
```

Der Reihe nach werden die Operatoren `+` (Addition), `-` (Subtraktion) und `*` (Multiplikation) eingesetzt. Wenn Sie die Taste betätigen, erscheint das Ergebnis jeweils in der Zeile darunter.

Sie können sowohl mit ganzen Zahlen rechnen als auch mit Zahlen, die Nachkommastellen besitzen. Die Nachkommastellen müssen mithilfe eines Dezimalpunkts abgetrennt werden.

2.1.3 Division, Ganzzahldivision und Modulo

Der Operator / dient der mathematischen Division:

```
>>> 22 / 8  
2.75  
>>> 22 / -8  
-2.75
```

Mithilfe des Operators // wird eine Ganzzahldivision durchgeführt:

```
>>> 22 // 8  
2  
>>> 22 // -8  
-3  
>>> 7 // 2.5  
2.0
```

Bei einer Ganzzahldivision mithilfe des Operators // wird zunächst das Ergebnis der mathematischen Division berechnet. Anschließend wird die nächstkleinere ganze Zahl ermittelt: Aus $22 / 8 = 2.75$ wird 2, aus $22 / -8 = -2.75$ wird -3. Handelt es sich bei mindestens einem der Operanden um eine Zahl mit Nachkommastellen, erscheint auch das Ergebnis als eine solche Zahl: Aus $7 / 2.5 = 2.8$ wird 2.0.

Der Modulo-Operator % berechnet den Rest einer Ganzzahldivision:

```
>>> 22 % 8  
6  
>>> 22.5 % 8.5  
5.5
```

Die Ganzzahldivision $22 // 8$ ergibt »2 Rest 6«. Der Modulo-Operator liefert den Rest 6. Die Ganzzahldivision $22.5 // 8.5$ ergibt »2.0 Rest 5.5«. Der Modulo-Operator liefert den Rest 5.5.

2.1.4 Rangfolge und Klammern

Es gilt, wie in der Mathematik, Punkt- vor Strichrechnung. Multiplikation und Division haben also Vorrang vor Addition und Subtraktion. Das Setzen von Klammern führt dazu, dass die Ausdrücke innerhalb der Klammern zuerst berechnet werden. Zwei Beispiele:

```
>>> 7 + 2 * 3  
13  
>>> (7 + 2) * 3  
27
```

Bei der ersten Rechnung werden zunächst 2 und 3 multipliziert, anschließend wird 7 addiert. Bei der zweiten Rechnung werden zunächst 7 und 2 addiert, anschließend wird mit 3 multipliziert.

Übung »u_grundrechenarten«

Es wird vorausgesetzt, dass Python wie angegeben installiert ist. Rufen Sie die Entwicklungsumgebung IDLE auf. Ermitteln Sie die Ergebnisse der folgenden vier Aufgaben:

```
13 - 5 * 2 + 12/6  
7/2 - 5/4  
(12 - 5 * 2) / 4  
(1/2 - 1/4 + (4 + 3)/8) * 2
```

Es sollten sich folgende Lösungen ergeben: 5.0, 2.25, 0.5, 2.25

2.1.5 Variablen und Zuweisung

Bisher haben wir mit Zahlen nur gerechnet. Werden Zahlen im Verlauf einer Berechnung mehrmals benötigt, können sie in Variablen gespeichert werden.

Dies wird mithilfe der Umrechnung einer Entfernungsangabe von englischen Meilen in Kilometer gezeigt. Dabei gilt: 1 Meile = 1,609344 Kilometer. Zur Erinnerung: Als Dezimaltrennzeichen gilt der Punkt.

```
>>> mi = 1.609344  
>>> 2 * mi  
3.218688  
>>> 5 * mi  
8.04672  
>>> 22.5 * mi  
36.21024  
>>> 2.35 * mi  
3.7819584000000006
```

Der Umrechnungsfaktor wird zunächst in der Variablen `mi` gespeichert. Auf diese Weise können mehrere Umrechnungen

nacheinander ausgeführt werden. Es werden die Werte in Kilometer für 2 Meilen, 5 Meilen, 22,5 Meilen und 2,35 Meilen berechnet und ausgegeben.

Die Speicherung des Umrechnungsfaktors wiederum geschieht mit einer Zuweisung ($\text{mi} = 1.609344$). Dabei erhält die Variable `mi` den Wert, der rechts vom Gleichheitszeichen steht.

Einige Hinweise

In Python-Variablen können Sie ganze Zahlen, Zahlen mit Nachkommastellen, Zeichenketten (also Texte) und andere Objekte eines Programms mithilfe von Zuweisungen in Variablen speichern.

Das Ergebnis der letzten Berechnung ist mathematisch falsch. 2,35 Meilen entsprechen 3,7819584 Kilometern, also ohne eine weitere 6 an der 16. Nachkommastelle. Woher kommt die falsche Anzeige? Zahlen mit Nachkommastellen können im Gegensatz zu ganzen Zahlen nicht mathematisch exakt gespeichert werden. Es gibt einige Stellen im Buch, an denen dies zum Tragen kommt. Bei vielen Berechnungen ist allerdings die Abweichung so gering, dass sie in der Praxis zu vernachlässigen ist.

Es ist üblich, den Wert in Kilometer so auszugeben, dass auf drei Stellen hinter dem Komma gerundet wird, also auf den Meter genau. Auf eine solche formatierte Ausgabe gehe ich später in [Abschnitt 5.2.2, »Formatierung von Zahlen mit Nachkommastellen«](#), noch ein.

Den Namen einer Variablen können Sie unter Einhaltung der folgenden Regeln weitgehend frei wählen:

- Er kann aus den Buchstaben a bis z, A bis Z, Ziffern oder dem Zeichen `_` (Unterstrich) bestehen.
- Er darf nicht mit einer Ziffer beginnen.

- Er darf keinem der reservierten Wörter der Programmiersprache Python entsprechen. Das sind: `and`, `as`, `assert`, `break`, `class`, `continue`, `def`, `del`, `elif`, `else`, `except`, `exec`, `finally`, `for`, `from`, `global`, `if`, `import`, `in`, `is`, `lambda`, `not`, `or`, `pass`, `print`, `raise`, `return`, `try`, `while`, `with`, `yield`.
- Beachten Sie die Groß- und Kleinschreibung: Namen und Anweisungen müssen genau wie vorgegeben geschrieben werden. Die Namen `mi` und `Mi` bezeichnen verschiedene Variablen.

Übung »u_inch«

Für das englische Längenmaß Inch gilt folgende Umrechnung:
1 Inch entspricht 2,54 Zentimetern. Berechnen Sie für die folgenden Angaben den Wert in Zentimeter: 5 Inch, 20 Inch und 92,7 Inch. Vereinfachen Sie Ihre Berechnungen, indem Sie den Umrechnungsfaktor zunächst in einer Variablen speichern.

Wie Sie der obigen Liste entnehmen können, ist `in` als Variablename nicht geeignet. Es sollten sich folgende Lösungen ergeben: 12.7, 50.8, 235.458.

2.2 Erstes Programm

Ein erstes Python-Programm wird eingegeben, abgespeichert und aufgerufen. Dieser Vorgang wird ausführlich erklärt. Die einzelnen Schritte sind später bei jedem Python-Programm auszuführen.

Alle Programme und Erläuterungen beziehen sich zunächst auf Python unter Windows. Gibt es Unterschiede zu Ubuntu Linux oder macOS, werden sie im jeweiligen Abschnitt erwähnt. Sie finden alle Programme auch in den Materialien zum Buch.

2.2.1 Hallo Welt

Die Ausgabe des ersten Python-Programms lautet:

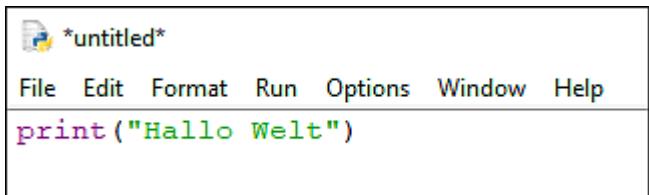
```
Hallo Welt
```

Das Programm gibt den Text »Hallo Welt« auf dem Bildschirm aus. Dies ist häufig das erste Programm, das man beim Erlernen einer beliebigen neuen Programmiersprache schreibt.

2.2.2 Eingabe eines Programms

Zur Eingabe des Programms rufen Sie in der Entwicklungsumgebung IDLE im Menü FILE zunächst den Menübefehl NEW FILE auf. Es öffnet sich ein neues Fenster mit dem Titel UNTITLED. Das Hauptfenster mit dem Titel IDLE SHELL 3.10.4 rückt in den Hintergrund.

In dem neuen Fenster geben Sie das Programm wie in [Abbildung 2.2](#) dargestellt ein.



```
*untitled*
File Edit Format Run Options Window Help
print("Hallo Welt")
```

Abbildung 2.2 Eingabe des Programms in neuem Fenster

Die eingebaute Funktion `print()` gibt Text oder die Werte von Variablen auf dem Bildschirm aus.

2.3 Speichern und Ausführen

Damit Sie das Ergebnis des Programms sehen können, müssen Sie es zunächst in einer Datei speichern und anschließend ausführen. Richten Sie als Erstes ein eigenes Verzeichnis für die Programmbeispiele ein. Ich habe dafür unter Windows das Verzeichnis *C:\Python\Beispiele* erstellt.

2.3.1 Speichern

Zur Speicherung des Programms rufen Sie im aktuellen Fenster mit dem Titel **UNTITLED** im Menü **FILE** den Menübefehl **SAVE** auf. Das Programm wird im Verzeichnis *C:\Python\Beispiele* in der Datei mit dem Namen *hallo.py* gespeichert (siehe [Abbildung 2.3](#)).

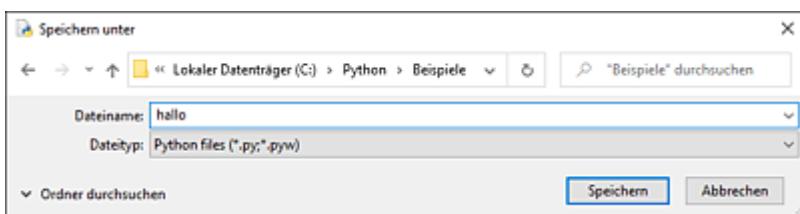


Abbildung 2.3 Speichern des Python-Programms

Nach der Betätigung des Buttons **SPEICHERN** ist die Speicherung vollzogen. Den Namen der Datei (hier *hallo*) können Sie frei wählen. Die Dateiendung *.py* ist für Python-Programme vorgeschrieben.

Das Fenster mit dem Programm sieht nun aus, wie in [Abbildung 2.4](#) dargestellt. In der Titelzeile sehen Sie den Dateinamen bzw. den vollständigen Pfad.

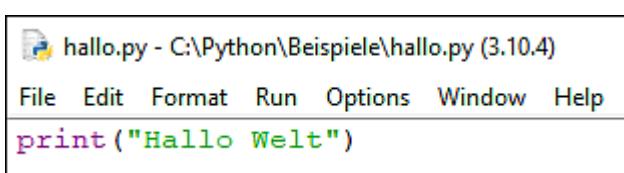


Abbildung 2.4 Dateiname und Pfad in der Titelzeile

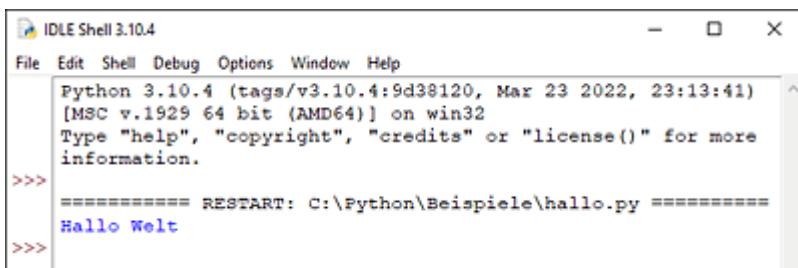
2.3.2 Ausführen unter Windows

Sie können das Programm unter Windows auf zwei verschiedene Arten aufrufen:

1. innerhalb der Entwicklungsumgebung IDLE (das machen wir bei den meisten Programmen dieses Buchs)
2. von der Kommandozeile aus

Innerhalb der Entwicklungsumgebung IDLE

Betrachten wir im Folgenden zunächst den Aufruf innerhalb der Entwicklungsumgebung IDLE. Dazu führen Sie im Menü **RUN** den Menübefehl **RUN MODULE** aus (Taste **F5**). Das Hauptfenster der Entwicklungsumgebung rückt wieder in den Vordergrund, und der Ausgabetext erscheint (siehe [Abbildung 2.5](#)).



The screenshot shows the IDLE Shell 3.10.4 window. The title bar reads "IDLE Shell 3.10.4". The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The main window displays Python version information and a command-line session. The session starts with the Python interpreter's banner: "Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41) [MSC v.1929 64 bit (AMD64)] on win32". It then shows a command-line prompt starting with '>>>'. The command 'RESTART' is shown, followed by the path 'C:\Python\Beispiele\hallo.py'. Finally, the text 'Hallo Welt' is printed in blue, indicating it was output from a previously run program.

Abbildung 2.5 Ergebnis des Programms

Über die Taskleiste können Sie anschließend das Programmfenster erneut in den Vordergrund rücken.

Von der Kommandozeile aus

Das Programm in der Datei *hallo.py* ist ein Kommandozeilenprogramm. Es generiert also nur eine einfache Ausgabe auf dem Bildschirm. In [Kapitel 11](#), »Benutzeroberflächen«, lernen Sie, wie Sie mithilfe von Python Programme mit grafischen Benutzeroberflächen erzeugen.

Für einen Aufruf des Programms aus der Kommandozeile müssen Sie unter Windows zunächst auf die

Kommandozeilenebene wechseln. In Windows rufen Sie dazu im Startmenü das Programm EINGABEAUFGORDERUNG in der Gruppe WINDOWS-SYSTEM auf.

Es erscheint ein Kommandozeilenfenster wie in [Abbildung 2.6](#).

```
Eingabeaufforderung
Microsoft Windows [Version 10.0.19042.1348]
(c) Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\Theis>
```

Abbildung 2.6 Kommandozeilenfenster

Mit Eingabe des Befehls `cd \Python\Beispiele` wechseln Sie nun in das Verzeichnis `C:\Python\Beispiele`, in dem das Programm in der Datei `hallo.py` gespeichert ist. Sie veranlassen die Ausführung des Programms durch Aufruf des Python-Interpreters mit der Anweisung `python hallo.py`. Es erscheint die in [Abbildung 2.7](#) gezeigte Ausgabe.

Dieses Kommandozeilenfenster können Sie für spätere Aufrufe von Python-Programmen geöffnet lassen oder mit Eingabe des Befehls `exit` wieder schließen.

```
Eingabeaufforderung
Microsoft Windows [Version 10.0.19042.1348]
(c) Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\Theis>cd \Python\Beispiele
C:\Python\Beispiele>python hallo.py
Hallo Welt
C:\Python\Beispiele>
```

Abbildung 2.7 Aufruf im Kommandozeilenfenster

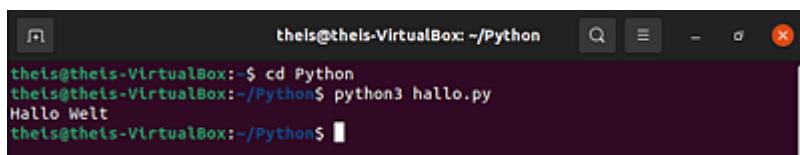
Dank der Tatsache, dass Sie bei der Installation ADD PYTHON TO ENVIRONMENT VARIABLES markiert haben, steht Ihnen der Python-Interpreter in jedem Verzeichnis zur Verfügung.

2.3.3 Ausführen unter Ubuntu Linux und unter macOS

Sie können das Programm unter Ubuntu Linux und unter macOS ebenfalls auf zwei Arten aufrufen, zum einen innerhalb der Entwicklungsumgebung IDLE (siehe den entsprechenden Unterabschnitt im vorherigen [Abschnitt 2.3.2](#), »Ausführen unter Windows«), zum anderen aus einem Terminal heraus.

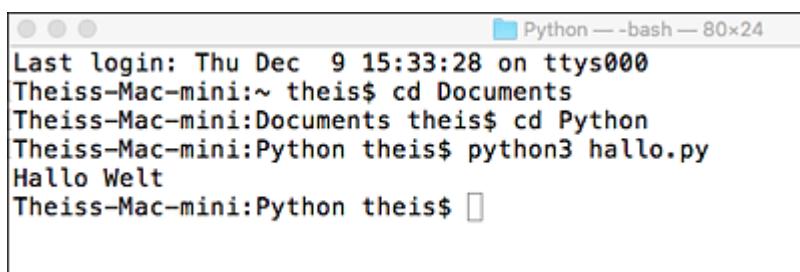
Öffnen Sie ein Terminal, und wechseln Sie in das Verzeichnis, in dem Sie die Datei *hallo.py* gespeichert haben. In Ubuntu Linux finden Sie das Terminal nach Eingabe des Begriffs im DASH. Unter macOS erreichen Sie das Terminal über das DOCK. Öffnen Sie darin das LAUNCHPAD, und wählen Sie die Gruppe ANDERE.

Veranlassen Sie die Ausführung des Programms für Python 3 mit der Anweisung `python3 hallo.py` (siehe [Abbildung 2.8](#) und [Abbildung 2.9](#)).



```
theis@theis-VirtualBox:~/Python$ cd Python
theis@theis-VirtualBox:~/Python$ python3 hallo.py
Hallo Welt
theis@theis-VirtualBox:~/Python$
```

Abbildung 2.8 Aufruf von Python 3 in Ubuntu Linux



```
Last login: Thu Dec  9 15:33:28 on ttys000
Theiss-Mac-mini:~ theis$ cd Documents
Theiss-Mac-mini:Documents theis$ cd Python
Theiss-Mac-mini:Python theis$ python3 hallo.py
Hallo Welt
Theiss-Mac-mini:Python theis$
```

Abbildung 2.9 Aufruf von Python 3 in macOS

Durch die Eingabe von `python3 -v` (großes »V«) können Sie an dieser Stelle auch die genaue Versionsnummer des benutzten Python feststellen.

Hinweis

Sie finden weitere Kommandozeilenbefehle für Ubuntu Linux und macOS in [Abschnitt A.4](#), »UNIX-Befehle«.

2.3.4 Kommentare

Bei umfangreicheren Programmen sollten Sie Kommentare zur Erläuterung in den Programmtext einfügen. Einzeilige Kommentare werden durch das Rautezeichen # eingeleitet und reichen bis zum Zeilenende. Mehrzeilige Kommentare beginnen und enden jeweils mit drei doppelten Anführungszeichen. Kommentare werden vom System nicht als Programmschritte betrachtet und folglich nicht ausgeführt. Im folgenden Listing wird das erste Programm durch Kommentare ergänzt:

```
# Mein erstes Programm
print("Hallo Welt")      # Eine Ausgabe
"""Kommentar in
mehreren Zeilen"""


```

Listing 2.1 Datei »hallo.py«, mit Kommentaren

2.3.5 Verkettung von Ausgaben

Mithilfe der Funktion `print()` können auch mehrere Ausgaben in einer Zeile vorgenommen werden. Das Programm wird weiter verändert:

```
# Mein erstes Programm
print("Hallo", "Welt")
```

Listing 2.2 Datei »hallo.py«, mehrere Ausgaben

Die einzelnen Teile der Ausgabe werden durch Kommata voneinander getrennt. Nach jedem Teil der Ausgabe wird automatisch ein Leerzeichen eingesetzt. Dieses Verhalten können Sie beeinflussen; mehr dazu in [Abschnitt 5.2.1](#), »Funktion `>print()<`«, über die Funktion `print()`.

2.3.6 Lange Ausgaben

Zeichenketten, die mithilfe der Funktion `print()` ausgegeben werden, können sehr lang werden, eventuell sogar über den rechten Rand innerhalb des Editors hinausgehen. Damit wird der Programmcode unübersichtlich. Sie können solche Zeichenketten auf mehrere Zeilen verteilen – dieses Vorgehen wird auch im vorliegenden Buch aufgrund der begrenzten Druckbreite häufig angewendet.

Wenngleich es für dieses kurze Programm nicht notwendig ist, verteilen wir hier zur Verdeutlichung den Code auf mehrere Zeilen:

```
# Mein erstes Programm
print("Hallo",
      "Welt")
print("Hallo "
      "Welt")
```

Listing 2.3 Datei »hallo.py«, lange Ausgaben

Es bietet sich an, den Zeilenumbruch nach einem Komma durchzuführen, wie bei der ersten Ausgabe von »Hallo Welt«. Dabei wird, wie bereits ausgeführt, automatisch ein Leerzeichen gesetzt.

Einzelne lange Texte können in Teiltexte zerlegt werden, wie bei der zweiten Ausgabe von »Hallo Welt«. Teiltextrte müssen nicht durch Kommata voneinander getrennt werden. Sie sind jeweils mit Anführungsstrichen zu begrenzen. Das trennende Leerzeichen zwischen »Hallo« und »Welt« müssen Sie nun von Hand setzen.

Hinweis

In [Abschnitt 5.1.2](#), »Anweisung in mehreren Zeilen«, erläutere ich, wie Sie allgemein lange Programmzeilen aufteilen können.

3 Programmierkurs

Der folgende Programmierkurs mit ausführlichen Erläuterungen führt Sie schrittweise in die Programmierung mit Python ein. Begleitet wird der Kurs von einem Programmierprojekt, das die vielen Teilespekte zu einem Ganzen verknüpft.

3.1 Ein Spiel programmieren

Damit Sie die Programmiersprache Python auf unterhaltsame Weise kennenlernen, werden Sie im Folgenden ein Spiel programmieren. Es wird im Verlauf des Buchs kontinuierlich erweitert und verbessert. Zunächst wird der Ablauf des Spiels beschrieben.

Nach Aufruf des Programms wird dem Benutzer eine Kopfrechenaufgabe gestellt. Er gibt das von ihm ermittelte Ergebnis als Lösungsvorschlag ein, und das Programm bewertet seine Eingabe.

```
Die Aufgabe: 9 + 26
Bitte Lösungsvorschlag eingeben:
34
34 ist falsch
Bitte Lösungsvorschlag eingeben:
35
35 ist richtig
Ergebnis: 35
Anzahl der Versuche: 2
```

Das Spiel wird in mehreren Einzelschritten erstellt. Zunächst entsteht eine einfache Version. Mit zunehmenden Programmierkenntnissen entwickeln Sie immer komplexere Versionen. Die im jeweiligen Abschnitt erlernten

Programmierfähigkeiten setzen Sie unmittelbar zur Verbesserung des Spielablaufs ein.

In späteren Abschnitten des Buchs entstehen weitere Versionen des Spiels. Es begleitet Sie auf diese Weise durch das gesamte Buch. Unter anderem wird es um die folgenden Möglichkeiten erweitert:

- Es werden mehrere Aufgaben gestellt.
- Die benötigte Zeit wird gemessen.
- Der Name des Spielers und die benötigte Zeit werden als Highscore-Liste dauerhaft in einer Datei oder einer Datenbank gespeichert.
- Die Highscore-Liste wird mit neuen Ergebnissen aktualisiert und auf dem Bildschirm dargestellt.
- Es gibt eine Version auf einer grafischen Benutzeroberfläche.
- Es gibt eine Version, die das Spielen im Internet ermöglicht.

3.2 Variablen und Operatoren

Zur Speicherung von Werten werden Variablen benötigt. Operatoren dienen zur Ausführung von Berechnungen.

3.2.1 Berechnung und Zuweisung

Im folgenden Programm wird eine einfache Berechnung mithilfe eines Operators durchgeführt. Das Ergebnis der Berechnung wird mit dem Gleichheitszeichen einer Variablen zugewiesen. Es erfolgt eine Ausgabe. Diese Schritte kennen Sie bereits aus [Abschnitt 2.1.5](#), »Variablen und Zuweisung«.

```
a = 5
b = 3
c = a + b

print("Die Aufgabe:", a, "+", b)
print("Das Ergebnis:", c)
```

Listing 3.1 Datei »zuweisung.py«

Die Ausgabe des Programms lautet:

```
Die Aufgabe: 5 + 3
Das Ergebnis: 8
```

In den beiden Variablen `a` und `b` wird jeweils ein Wert gespeichert. Die beiden Werte werden addiert, das Ergebnis wird der Variablen `c` zugewiesen. Die Aufgabenstellung wird ausgegeben, anschließend das Ergebnis. Der Benutzer des Programms hat noch keine Möglichkeit, in den Ablauf einzugreifen.

3.2.2 Ausgabe mit formatiertem String-Literal

Formatierte String-Literale dienen zur komfortablen Einbettung von einfachen Variablen, aber auch komplexen berechneten Ausdrücken oder Aufrufen von Funktionen und Methoden in Zeichenketten.

Nachfolgend sehen Sie das Programm aus dem vorherigen Abschnitt in einer Version mit String-Literalen:

```
a = 5
b = 3
c = a + b

tx = f"Die Aufgabe: {a} + {b}"
print(tx)
print(f"Das Ergebnis: {c}")
```

Listing 3.2 Datei »literal.py«

Die Ausgabe des Programms lautet:

```
Die Aufgabe: 5 + 3
Das Ergebnis: 8
```

Das Zeichen »f« vor dem Beginn der Zeichenkette leitet ein formatiertes String-Literal ein.

Die erste Zeile wird als Zeichenkette in der Variablen `tx` gespeichert, die anschließend ausgegeben wird. Die Werte der Variablen `a` und `b` werden mithilfe von geschweiften Klammern in die Zeichenkette eingebettet. Diese Zeichen erreichen Sie mithilfe der Taste `Alt` rechts neben dem Leerzeichen.

Der Text der zweiten Zeile wird unmittelbar ausgegeben, inklusive der eingebetteten Werte. Mehr zu den Möglichkeiten von String-Literalen folgt in [Abschnitt 5.2.2](#), »Formatierung von Zahlen mit Nachkommastellen«.

3.2.3 Eingabe einer Zeichenkette

In diesem Abschnitt wird die eingebaute Funktion `input()` zur Eingabe einer Zeichenkette durch die Benutzerin eingeführt. Ein kleines Beispiel:

```
print("Bitte einen Text eingeben")
x = input()
print("Ihre Eingabe:", x)
```

Listing 3.3 Datei »eingabe_text.py«

Die Ausgabe könnte wie folgt aussehen:

```
Bitte einen Text eingeben
Ich schreibe Python-Programme
Ihre Eingabe: Ich schreibe Python-Programme
```

Der Benutzer gibt einen kleinen Satz ein. Dieser Satz wird in der Variablen `x` gespeichert und anschließend ausgegeben.

3.2.4 Eingabe einer Zahl

Im weiteren Verlauf des Spiels ist es notwendig, die Eingabe des Benutzers als Zahl weiterzuverwenden. Dazu muss die Zeichenkette, die die Funktion `input()` liefert, in eine Zahl umgewandelt werden.

Zur Umwandlung gibt es unter anderem die folgenden Funktionen:

- Die eingebaute Funktion `int()` wandelt eine Zeichenkette, die eine gültige ganze Zahl enthält, in eine ganze Zahl um. Enthält die Zeichenkette keine gültige ganze Zahl, bricht das Programm ab.
- Die eingebaute Funktion `float()` wandelt eine Zeichenkette, die eine gültige Zahl enthält, in eine Zahl um. Enthält die Zeichenkette keine gültige Zahl, bricht das Programm ab. Eine Zahl mit einem Komma als Dezimaltrennzeichen ist nicht gültig.

In [Abschnitt 3.6](#), »Fehler und Ausnahmen«, lernen Sie, wie Sie den Abbruch eines Programms abfangen. Bis zu dem genannten Abschnitt gehe ich vereinfacht davon aus, dass der Anwender korrekte Eingaben vornimmt.

Ein Beispiel mit der Funktion `int()`:

```
print("Bitte eine ganze Zahl eingeben")
x = input()
print("Ihre Eingabe:", x)

xganz = int(x)
print("Als ganze Zahl:", xganz)
```

```
xdoppel = xganz * 2
print("Das Doppelte:", xdoppel)
```

Listing 3.4 Datei »eingabe_zahl.py«

Die Ausgabe könnte wie folgt aussehen:

```
Bitte eine ganze Zahl eingeben
6
Ihre Eingabe: 6
Als ganze Zahl: 6
Das Doppelte: 12
```

Der Benutzer gibt eine Zeichenkette ein. Diese Zeichenkette wird mithilfe der eingebauten Funktion `int()` in eine ganze Zahl umgewandelt. Die Zahl und das Doppelte der Zahl werden ausgegeben.

3.2.5 Spiel, Version mit Eingabe

Das Spiel, in dem die Benutzerin eine oder mehrere Kopfrechenaufgaben lösen soll, erhält eine Eingabe. Es wird wie folgt geändert:

```
a = 5
b = 3
c = a + b
print(f"Die Aufgabe: {a} + {b}")

print("Bitte Lösungsvorschlag eingeben:")
z = input()
zahl = int(z)

print("Ihre Eingabe:", z)
print("Das Ergebnis:", c)
```

Listing 3.5 Datei »spiel_eingabe.py«

Eine mögliche Ausgabe des Programms:

```
Die Aufgabe: 5 + 3
Bitte Lösungsvorschlag eingeben:
9
Ihre Eingabe: 9
Das Ergebnis: 8
```

Das Programm gibt die Aufforderung `Bitte Lösungsvorschlag eingeben:` aus und hält an. Die Eingabe der Benutzerin wird in der

Variablen `z` gespeichert. Die Zeichenkette `z` wird mithilfe der Funktion `int()` in eine ganze Zahl verwandelt.

Übung »u_eingabe_inch«

Schreiben Sie ein Programm zur Eingabe und Umrechnung von beliebigen Inch-Werten in Zentimeter. Speichern Sie das Programm in der Datei `u_eingabe_inch.py`. Rufen Sie das Programm auf, und testen Sie es. Die Ausgabe kann zum Beispiel wie folgt aussehen:

```
Bitte geben Sie den Inch-Wert ein:  
3.5  
3.5 Inch sind 8.89 cm
```

Übung »u_eingabe_gehalt«

Schreiben Sie ein Programm zur vereinfachten Berechnung der Steuer. Der Anwender wird dazu aufgefordert, sein monatliches Gehalt einzugeben. Anschließend werden 18 % dieses Betrags berechnet und ausgegeben. Nutzen Sie die Datei `u_eingabe_gehalt.py`. Die Ausgabe kann zum Beispiel wie folgt aussehen:

```
Geben Sie Ihr Gehalt in Euro ein:  
2500  
Es ergibt sich eine Steuer von 450.0 Euro
```

3.2.6 Zufallszahlen

In Python steht mithilfe des Moduls `random` ein Zufallsgenerator zur Verfügung. Er erzeugt zufällige Zahlen, die wir zur Erstellung der Kopfrechenaufgabe nutzen. Die Funktionen des Zufallsgenerators befinden sich in einem zusätzlichen Modul, das zunächst importiert werden muss. Das Programm wird wie folgt verändert:

```
import random  
random.seed()  
  
a = random.randint(1,10)
```

```
b = random.randint(1,10)
c = a + b
print(f"Die Aufgabe: {a} + {b}")

print("Bitte Lösungsvorschlag eingeben:")
zahl = int(input())
print("Ihre Eingabe:", zahl)
print("Das Ergebnis:", c)
```

Listing 3.6 Datei »spiel_zufallszahl.py«

Eine mögliche Ausgabe des Programms, abhängig von den gelieferten zufälligen Werten, sieht wie folgt aus:

```
Die Aufgabe: 8 + 3
Bitte Lösungsvorschlag eingeben:
7
Ihre Eingabe: 7
Das Ergebnis: 11
```

Zusätzliche Module können Sie mithilfe der Anweisung `import` in das Programm einbinden. Die Funktionen dieser Module können Sie anschließend in der Schreibweise `Modulname.Funktionsname` aufrufen.

Der Aufruf der Funktion `seed()` des Moduls `random` führt dazu, dass der Zufallszahlengenerator mit der aktuellen Systemzeit initialisiert wird. Andernfalls könnte es passieren, dass anstelle einer zufälligen Auswahl immer wieder die gleichen Zahlen geliefert würden.

Die Funktion `randint()` des Moduls `random` liefert eine ganze Zufallszahl im angegebenen Bereich. Im vorliegenden Fall ist dies also eine zufällige Zahl von 1 bis 10.

In der Anweisung `zahl = int(input())` werden zwei Funktionen geschachtelt aufgerufen. Zunächst die Funktion `input()`, damit das Programm eine Eingabe des Benutzers entgegennimmt, anschließend die Funktion `int()`, damit die Eingabe in eine ganze Zahl umgewandelt wird.

Hinweis

Die Funktionen des Moduls `random` können für die zufälligen Werte eines Spiels genutzt werden. Benötigen Sie zufällige

Werte zum Zweck der Verschlüsselung oder aus Gründen der Sicherheit, sollten Sie die Funktionen des Moduls `secrets` nutzen, das es seit Python 3.6 gibt, siehe [Abschnitt 5.4, »Verschlüsselung«.](#)

3.3 Verzweigungen

In den bisherigen Programmen werden alle Anweisungen der Reihe nach ausgeführt. Zur Steuerung des Programmablaufs werden allerdings häufig Verzweigungen benötigt. Innerhalb des Programms wird anhand eines Vergleichs entschieden, welcher Zweig des Programms ausgeführt wird.

3.3.1 Vergleichsoperatoren

Ein Vergleich wird mithilfe eines Vergleichsoperators formuliert. Tabelle 3.1 listet die Vergleichsoperatoren mit ihrer Bedeutung auf.

Operator	Bedeutung
>	größer als
<	kleiner als
>=	größer als oder gleich
<=	kleiner als oder gleich
==	gleich
!=	ungleich

Tabelle 3.1 Vergleichsoperatoren

Ein Vergleich liefert einen der beiden Wahrheitswerte `True` oder `False` (»wahr« oder »falsch«). Wahrheitswerte können in

Variablen des Datentyps `bool` gespeichert werden. Es folgt ein Beispiel:

```
x = 12
print("x:", x)

print("x == 12:", x == 12)
z = x != 12
print("x != 12:", z)
```

Listing 3.7 Datei »operator_vergleich.py«

Die Ausgabe des Programms:

```
x: 12
x == 12: True
x != 12: False
```

Das Ergebnis des ersten Vergleichs wird unmittelbar ausgegeben. Das Ergebnis des zweiten Vergleichs wird in der booleschen Variablen `z` gespeichert. Meist werden Wahrheitswerte zur Steuerung von Verzweigungen oder Schleifen genutzt. Mehr zu Wahrheitswerten folgt in [Abschnitt 4.7](#), »Wahrheitswerte und Nichts«.

3.3.2 Verzweigung mit »if«

Im folgenden Beispiel wird untersucht, ob eine zufällig ermittelte ganze Zahl positiv ist. Ist dies der Fall, wird »Diese Zahl ist positiv« ausgegeben, anderenfalls lautet die Ausgabe »Diese Zahl ist 0 oder negativ«. Es wird also nur eine der beiden Anweisungen ausgeführt.

```
import random
random.seed
x = random.randint(-5, 5)
print("x:", x)

if x > 0:
    print("Diese Zahl ist positiv")
else:
    print("Diese Zahl ist 0 oder negativ")
```

Listing 3.8 Datei »verzweigung_if.py«

Eine mögliche Ausgabe des Programms lautet:

```
x: 3  
Diese Zahl ist positiv.
```

Zunächst erhält die Variable `x` einen zufälligen Wert, hier: 3.

Eine Verzweigung wird mithilfe von `if` eingeleitet. Danach wird eine Bedingung formuliert (hier $x > 0$), die entweder *wahr* oder *falsch* ergibt. Anschließend kommt ein Doppelpunkt.

Es folgen eine oder mehrere Anweisungen, die nur ausgeführt werden, falls die Bedingung *wahr* ergibt. Die Anweisungen müssen innerhalb des sogenannten `if`-Zweigs mithilfe der -Taste eingerückt werden, damit Python die Zugehörigkeit zur Verzweigung erkennen kann.

Es kann bei einer Verzweigung einen alternativen Teil geben, der mithilfe der Anweisung `else` eingeleitet wird. Danach folgt wiederum ein Doppelpunkt. Anschließend werden eine oder mehrere Anweisungen notiert, die nur ausgeführt werden, falls die Bedingung *falsch* ergibt. Auch diese Anweisungen müssen eingerückt werden.

Hinweis

Standardmäßig wird eine Einrückung bereits von IDLE vorgenommen. Sie macht das Programm für den Entwickler übersichtlicher. Das gilt nicht nur für Verzweigungen, sondern auch für Schleifen und andere Kontrollstrukturen.

3.3.3 Spiel, Version mit Bewertung der Eingabe

Das Spiel, in dem der Benutzer eine oder mehrere Kopfrechenaufgaben lösen soll, wird um eine Bewertung der Eingabe erweitert. Mithilfe einer einfachen Verzweigung wird untersucht, ob sie richtig oder falsch war. Das Programm verändert sich wie folgt:

```
import random  
random.seed()
```

```

a = random.randint(1,10)
b = random.randint(1,10)
c = a + b
print(f"Die Aufgabe: {a} + {b}")

print("Bitte Lösungsvorschlag eingeben:")
zahl = int(input())

if zahl == c:
    print(zahl, "ist richtig")
else:
    print(zahl, "ist falsch")
    print("Ergebnis:", c)

```

Listing 3.9 Datei »spiel_verzweigung.py«

Eine mögliche Ausgabe des Programms wäre:

```

Die Aufgabe: 2 + 8
Bitte Lösungsvorschlag eingeben:
11
11 ist falsch
Ergebnis: 10

```

Die Eingabe wird in eine Zahl umgewandelt. Entspricht diese dem Ergebnis der Rechnung, so erscheint »... ist richtig«. Entspricht sie nicht dem Ergebnis, so erscheint »... ist falsch« zusammen mit dem korrekten Ergebnis.

Übung »u_verzweigung_if«

Das vereinfachte Programm zur Berechnung der Steuer wird verändert. Der Anwender soll dazu aufgefordert werden, sein monatliches Gehalt einzugeben. Liegt es über 2.500 Euro, sind 22 % Steuern zu zahlen, ansonsten 18 %. Nutzen Sie die Datei *u_verzweigung_if.py*.

Es ist nur *eine* Eingabe erforderlich. Innerhalb des Programms soll anhand des Gehalts entschieden werden, welcher Steuersatz zur Anwendung kommt. Die Ausgabe kann zum Beispiel wie folgt aussehen:

```

Geben Sie Ihr Gehalt in Euro ein:
3000
Es ergibt sich eine Steuer von 660.0 Euro

```

oder sie kann so aussehen:

```

Geben Sie Ihr Gehalt in Euro ein:
2000

```

```
Es ergibt sich eine Steuer von 360.0 Euro
```

3.3.4 Mehrfache Verzweigung

In vielen Anwendungsfällen gibt es mehr als zwei Möglichkeiten, zwischen denen zu entscheiden ist. Dann wird eine mehrfache Verzweigung benötigt.

Im folgenden Beispiel wird untersucht, ob eine Zahl positiv, negativ oder gleich 0 ist. Es wird eine entsprechende Meldung ausgegeben:

```
import random
random.seed
x = random.randint(-5, 5)
print("x:", x)

if x > 0:
    print("x ist positiv")
elif x < 0:
    print("x ist negativ")
else:
    print("x ist gleich 0")
```

Listing 3.10 Datei »verzweigung_mehrfach.py«

Eine mögliche Ausgabe des Programms:

```
x: -3
x ist negativ
```

Die Verzweigung wird mithilfe von `if` eingeleitet. Ist `x` positiv, werden die nachfolgenden, eingerückten Anweisungen ausgeführt.

Nach `elif` wird eine weitere Bedingung formuliert. Sie wird nur untersucht, falls die erste Bedingung (nach dem `if`) nicht zutrifft. Ist `x` negativ, werden die nachfolgenden, eingerückten Anweisungen ausgeführt.

Die Anweisungen nach dem `else` werden nur durchgeführt, falls keine der beiden vorherigen Bedingungen zutraf (nach dem `if` und nach dem `elif`). Im vorliegenden Fall bedeutet das, dass `x` gleich 0 ist, da es nicht positiv und nicht negativ ist.

Hinweis

Innerhalb einer Verzweigung können mehrere `elif`-Anweisungen vorkommen. Sie werden der Reihe nach untersucht, bis das Programm zu einer Bedingung kommt, die zutrifft. Die weiteren `elif`-Anweisungen oder eine `else`-Anweisung werden in diesem Fall nicht mehr beachtet.

Übung »u_verzweigung_mehrfach«

Das Programm zur Berechnung der Steuer soll weiter verändert werden (Datei `u_verzweigung_mehrfach.py`). Der Anwender soll sein monatliches Gehalt eingeben. Anschließend wird seine Steuer nach der folgenden Tabelle berechnet

Gehalt	Steuersatz
mehr als 4.000 Euro	26 %
2.500 bis 4.000 Euro	22 %
weniger als 2.500 Euro	18 %

3.3.5 Bedingter Ausdruck

Wird in allen Fällen einer Verzweigung nur ein Wert zugewiesen, können Sie auch einen *bedingten Ausdruck* (englisch: *conditional expression*) verwenden. Das Ergebnis des Ausdrucks kann gespeichert oder unmittelbar ausgegeben werden. Es folgt ein Programm:

```
import random
random.seed
x = random.randint(-3, 3)
print("x:", x)

ausgabe = "positiv" if x>0 else "negativ"
```

```
print("Diese Zahl ist", ausgabe)

print("Diese Zahl ist", "positiv" if x>0 else "negativ")

print("Diese Zahl ist",
      "positiv" if x>0 else "negativ" if x<0 else "gleich 0")
```

Listing 3.11 Datei »bedingter_ausdruck.py«

Eine mögliche Ausgabe:

```
x: 0
Diese Zahl ist negativ
Diese Zahl ist negativ
Diese Zahl ist gleich 0
```

Mithilfe des ersten bedingten Ausdrucks wird der Variablen ausgabe der Wert »positiv« zugewiesen, falls x größer als 0 ist, anderenfalls der Wert »negativ«. Der zugewiesene Wert von ausgabe wird anschließend ausgegeben.

Das Ergebnis des zweiten bedingten Ausdrucks wird unmittelbar ausgegeben.

Bedingte Ausdrücke können zur Bildung von mehrfachen Verzweigungen geschachtelt werden. Der dritte bedingte Ausdruck ermöglicht drei verschiedene Ausgaben.

3.3.6 Logische Operatoren

Mithilfe der logischen Operatoren `and`, `or` und `not` können mehrere Bedingungen miteinander verknüpft werden.

- Eine Bedingung, die aus einer oder mehreren Einzelbedingungen besteht, die jeweils mit dem Operator `and` (= und) verknüpft sind, ergibt *wahr*, wenn *jede* der Einzelbedingungen *wahr* ergibt.
- Eine Bedingung, die aus einer oder mehreren Einzelbedingungen besteht, die mit dem Operator `or` (= oder) verknüpft sind, ergibt *wahr*, wenn *mindestens eine* der Einzelbedingungen *wahr* ergibt.

- Der Operator `not` (= nicht) kehrt den Wahrheitswert einer Bedingung um, d. h., eine falsche Bedingung wird wahr, eine wahre Bedingung wird falsch.

Der Zusammenhang wird auch in [Tabelle 3.2](#) bis [Tabelle 3.4](#) gezeigt.

Bedingung 1	Bedingung 2	Ergebnis
wahr	wahr	wahr
wahr	falsch	falsch
falsch	wahr	falsch
falsch	falsch	falsch

Tabelle 3.2 Auswirkung des logischen Operators »and«

Bedingung 1	Bedingung 2	Ergebnis
wahr	wahr	wahr
wahr	falsch	wahr
falsch	wahr	wahr
falsch	falsch	falsch

Tabelle 3.3 Auswirkung des logischen Operators »or«

Bedingung	Ergebnis
------------------	-----------------

Bedingung	Ergebnis
wahr	falsch
falsch	wahr

Tabelle 3.4 Auswirkung des logischen Operators »not«

Ein Beispiel:

```
x = 12
y = 15
z = 20
print(f"x:{x}, y:{y}, z:{z}")

if x<y and x<z:
    print("x ist die kleinste Zahl")

if y>x or y>z:
    print("y ist nicht die kleinste Zahl")

if not y<x:
    print("y ist nicht kleiner als x")
```

Listing 3.12 Datei »operator_logisch.py«

Die Ausgabe des Programms:

```
x: 12, y: 15, z: 20
x ist die kleinste Zahl
y ist nicht die kleinste Zahl
y ist nicht kleiner als x
```

Bedingung 1 ergibt *wahr*, wenn *x* kleiner als *y* und kleiner als *z* ist. Dies trifft bei den gegebenen Anfangswerten zu. Bedingung 2 ergibt *wahr*, wenn *y* größer als *x* oder *y* größer als *z* ist. Die erste Bedingung trifft zu, also ist die gesamte Bedingung *wahr*: *y* ist nicht die kleinste der drei Zahlen. Bedingung 3 ist *wahr*, wenn *y* nicht kleiner als *x* ist. Dies trifft hier zu. Zu allen Verzweigungen kann es auch einen *else*-Zweig geben.

Übung »u_operator«

Das Programm zur Berechnung der Steuer soll wiederum verändert werden (Datei *u_operator.py*). Die Tabelle sieht nun

wie folgt aus:

Gehalt	Familienstand	Steuersatz
> 4.000 Euro	ledig	26 %
> 4.000 Euro	verheiratet	22 %
<= 4.000 Euro	ledig	22 %
<= 4.000 Euro	verheiratet	18 %

Übung »u_datum«

Entwickeln Sie ein Programm zur Prüfung einer Datumsangabe (Datei *u_datum.py*). Der Benutzer soll die drei Bestandteile eines Datums einzeln eingeben. Anschließend wird ermittelt, ob es sich um ein falsches oder ein richtiges Datum handelt.

Gehen Sie bei der Entwicklung wie nachfolgend beschrieben vor. Testen Sie Ihr Programm nach jedem Schritt

- Untersuchen Sie den eingegebenen Wert für den Tag. Ist er kleiner als 1 oder größer als 31, handelt es sich um ein falsches Datum.
- Untersuchen Sie den eingegebenen Wert für den Monat. Ist er kleiner als 1 oder größer als 12, handelt es sich um ein falsches Datum.
- Geben Sie den Wert aus, den der letzte Tag des betreffenden Monats hat. Denken Sie daran, dass es nur drei mögliche Fälle gibt: 28, 30 oder 31 Tage. Die Regeln für Schaltjahre werden noch nicht beachtet.
- Untersuchen Sie den eingegebenen Wert für den Tag. Geben Sie aus, ob er kleiner als 1 oder größer als der letzte Tag des

betreffenden Monats ist.

- Untersuchen Sie den eingegebenen Wert für das Jahr. Geben Sie aus, ob es sich um ein Schaltjahr handelt. Die vereinfachte Regel für ein Schaltjahr lautet Lässt sich der Wert ohne Rest durch 4 teilen, handelt es sich um ein Schaltjahr.
- Kombinieren Sie die bisherigen Schritte miteinander. Ist der Wert für den Tag kleiner als 1 oder größer als der letzte Tag des betreffenden Monats (mit Berücksichtigung der Regel für ein Schaltjahr), handelt es sich um ein falsches Datum, ansonsten um ein richtiges Datum.
- Erweitern Sie das Programm. Die vollständige Regel für ein Schaltjahr lautet Lässt sich der Wert ohne Rest durch 4 teilen, aber nicht ohne Rest durch 100, handelt es sich um ein Schaltjahr. Es handelt sich aber auch um ein Schaltjahr, falls sich der Wert ohne Rest durch 400 teilen lässt.

Ein Aufruf des fertigen Programms könnte wie folgt aussehen:

```
Tag des Datums eingeben:  
29  
Monat des Datums eingeben:  
2  
Jahr des Datums eingeben:  
2000  
Letzter Tag: 29  
Richtiges Datum
```

3.3.7 Mehrere Vergleichsoperatoren

Bedingungen können auch mehrere Vergleichsoperatoren enthalten. Manche Verzweigungen sind auf diese Weise verständlicher. Ein Beispiel:

```
x = 12  
y = 15  
z = 20  
print(f"x:{x}, y:{y}, z:{z}")
```

```
if x < y < z:  
    print("y liegt zwischen x und z")
```

Listing 3.13 Datei »operator_mehrere.py«

Die Ausgabe des Programms:

```
x: 12, y: 15, z: 20  
y liegt zwischen x und z
```

Die Bedingung $x < y < z$ entspricht dem Ausdruck $x < y \text{ and } y < z$. In der Kurzform ist sie allerdings besser lesbar.

3.3.8 Spiel, Version mit genauer Bewertung der Eingabe

Nun kann die Eingabe des Benutzers in dem Kopfrechenspiel auf verschiedene Art und Weise genauer bewertet werden:

- mithilfe einer mehrfachen Verzweigung
- anhand logischer Operatoren
- anhand von Bedingungen mit mehreren Vergleichsoperatoren

Das Programm wird wie folgt verändert:

```
import random  
random.seed()  
  
a = random.randint(1,10)  
b = random.randint(1,10)  
c = a + b  
print(f"Die Aufgabe: {a} + {b}")  
  
print("Bitte Lösungsvorschlag eingeben:")  
zahl = int(input())  
  
if zahl == c:  
    print(zahl, "ist richtig")  
elif zahl < 0 or zahl > 100:  
    print(zahl, "ist weit daneben")  
elif c-1 <= zahl <= c+1:  
    print(zahl, "ist nahe dran")  
else:  
    print(zahl, "ist falsch")  
  
print("Ergebnis:", c)
```

Listing 3.14 Datei »spiel_operator.py«

Eine mögliche Ausgabe des Programms wäre:

```
Die Aufgabe: 2 + 1
Bitte Lösungsvorschlag eingeben:
4
4 ist nahe dran
Ergebnis: 3
```

Insgesamt werden vier Möglichkeiten angeboten: über `if`, zweimal `elif` und `else`. Ist die Eingabe kleiner als 0 oder größer als 100, so liegt das weit neben dem richtigen Ergebnis. Dies wird mithilfe des logischen Operators `or` gelöst.

Unterscheidet sich die eingegebene Zahl vom richtigen Ergebnis nur um den Wert 1, ist die Eingabe nahe dran. Dies wird mit einer Bedingung ermittelt, die mehrere Vergleichsoperatoren enthält.

3.3.9 Verzweigung mit »match«

Seit Python 3.10 gibt es mit `match` eine weitere Möglichkeit, mehrfache Verzweigungen in übersichtlicher Form zu gestalten. Ein erstes Beispiel:

```
import random
random.seed()

x = "Paris"
match x:
    case "Paris":
        print("Frankreich")
    case "Rom":
        print("Italien")
    case "Madrid":
        print("Spanien")
    case _:
        print("Unbekanntes Land")
print()
...
```

Listing 3.15 Datei »verzweigung_match.py«, Teil 1 von 3

Die Ausgabe dieses Programmteils:

```
Frankreich
```

Der Zufallsgenerator wird erst im späteren Verlauf des Programms benötigt.

In der Variablen `x` wird eine Zeichenkette gespeichert. Nach dem Schlüsselwort `match` folgt der Ausdruck, mit dem verglichen wird. Dabei kann es sich um eine Variable oder auch um das Ergebnis einer Berechnung handeln. Nach dem Doppelpunkt folgen eingerückt die verschiedenen Fälle.

Ein einzelner Fall beginnt mit dem Schlüsselwort `case`. Es folgt der Wert, mit dem der `match`-Ausdruck verglichen wird, danach ein Doppelpunkt. Wiederum eingerückt folgen die Anweisungen, falls der betreffende Vergleich erfolgreich ist.

Mithilfe des Zeichens `_` kann am Ende ein weiterer Fall formuliert werden. Trifft keiner der zuvor genannten Fälle zu, werden die Anweisungen zu diesem Default-Fall ausgeführt.

Hier geht es um den Vergleich von Zeichenketten. Es kann sich aber auch um einen Vergleich von einzelnen Zeichen oder ganzen Zahlen handeln.

Ein Vergleich von Zahlen mit Nachkommastellen ist ebenfalls möglich. Allerdings werden diese nicht mathematisch exakt gespeichert. Daher könnte bereits eine kleine Abweichung zu einem falschen Ergebnis führen.

Es folgt der zweite Teil des Programms:

```
...
x = random.randint(1,6)
print("x =", x)
match x:
    case 1 | 3 | 5:
        print("ungerade")
    case 2 | 4 | 6:
        print("gerade")
    case _:
        print("Kein Würfelwert")
print()
```

Listing 3.16 Datei »verzweigung_match.py«, Teil 2 von 3

Eine mögliche Ausgabe dieses Programmteils:

```
x = 3
ungerade
```

Mithilfe des Zufallsgenerators wird eine zufällige ganze Zahl von 1 bis 6 ermittelt, also ein Würfelwert. Der Operator `|` ermöglicht die Verknüpfung von mehreren Fällen. Wird also eine 2, 4 oder 6 gewürfelt, erscheint als Ausgabe der Text »gerade«.

Der dritte und letzte Teil des Programms:

```
...
x = random.randint(1,10)
print("x * 1.5 =", x * 1.5)
match x * 1.5:
    case x if x < 5:
        print("kleiner Wert")
    case x if x > 11:
        print("großer Wert")
    case _:
        print("mittlerer Wert")
```

Listing 3.17 Datei »verzweigung_match.py«, Teil 3 von 3

Eine mögliche Ausgabe dieses Programmteils:

```
x * 1.5 = 13.5
großer Wert
```

Ein eingebettetes `if` ermöglicht den Einsatz von Vergleichsoperatoren. Nach dem Schlüsselwort `case` folgt ein Vergleich des untersuchten Ausdrucks. Auch hier gilt: Sobald der erste Vergleich zutrifft, werden die zugehörigen Anweisungen ausgeführt.

Auf diese Weise ist auch ein Vergleich für Zahlen mit Nachkommastellen sinnvoll möglich. Bei dem ersten Vergleich liegt der mögliche Wert 4.5 unterhalb der Grenze, der nächste mögliche Wert 6.0 oberhalb der Grenze.

3.3.10 Rangfolge der Operatoren

In vielen Ausdrücken treten mehrere Operatoren auf. Bisher sind dies Rechenoperatoren, Vergleichsoperatoren und logische Operatoren. Für die Reihenfolge bei der Ausführung ist die Rangfolge der Operatoren wichtig. Die Teilschritte, bei denen

höherrangige Operatoren beteiligt sind, werden zuerst ausgeführt.

Tabelle 3.5 gibt die Rangfolge der bisher verwendeten Operatoren in Python an, beginnend mit den Operatoren, die den höchsten Rang haben. Gleichrangige Operatoren stehen jeweils in einer Zeile. Teilschritte, in denen mehrere Operatoren gleichen Ranges stehen, werden von links nach rechts ausgeführt.

Operator	Bedeutung
+ -	positives Vorzeichen einer Zahl, negatives Vorzeichen einer Zahl
* / % //	Multiplikation, Division, Modulo, Ganzzahldivision
+ -	Addition, Subtraktion
< <= > >=	kleiner, kleiner oder gleich, größer, größer oder gleich
== !=	gleich, ungleich
not	logische Verneinung
and	logisches Und
or	logisches Oder

Tabelle 3.5 Rangfolge der bisher genutzten Operatoren

3.4 Schleifen

Neben der Verzweigung gibt es eine weitere wichtige Struktur zur Steuerung von Programmen: die Schleife. Sie ermöglicht die wiederholte Ausführung von Programmschritten.

Es wird zwischen zwei Typen von Schleifen unterschieden: der `for`-Schleife und der `while`-Schleife. Der jeweilige Anwendungsbereich der beiden Typen wird durch folgende Merkmale definiert:

- Eine `for`-Schleife wird verwendet, wenn ein Programmschritt für eine regelmäßige, zum Zeitpunkt der Anwendung bekannte Abfolge von Werten wiederholt ausgeführt werden soll.
- Eine `while`-Schleife wird verwendet, wenn sich erst durch Eingaben des Anwenders ergibt, ob ein Programmschritt ausgeführt werden soll und wie oft er wiederholt wird.

Eine `for`-Schleife wird auch als *Zählschleife* bezeichnet, eine `while`-Schleife als *bedingungsgesteuerte Schleife*.

3.4.1 Schleife mit »for«

In einer `for`-Schleife werden alle Elemente eines *iterierbaren Objekts* (kurz: *Iterables*) durchlaufen. Ein Iterable ist ein Objekt, das aus mehreren Elementen besteht und durchlaufen werden kann, zum Beispiel eine Liste von mehreren Zahlen oder eine Liste von mehreren Zeichenketten. Eine einzelne Zeichenkette ist ebenfalls iterierbar, da ihre Zeichen als Elemente der Zeichenkette durchlaufen werden können. Im weiteren Verlauf des Buchs lernen Sie noch weitere Iterables kennen.

Innerhalb der Schleife wird mit einer Kopie der Elemente gearbeitet. Eine Veränderung der Kopie hat keine Rückwirkung

auf das Original.

Nachfolgend sehen Sie einige Beispiele:

```
for zahl in 8, 3, 7:  
    print(f"Zahl: {zahl}, Quadrat: {zahl * zahl}")  
print()  
  
for stadt in "Paris", "Rom", "Madrid":  
    print(f"Stadt: {stadt}")  
print()  
  
for zeichen in "Rom":  
    print(f"Zeichen: {zeichen}")  
print()  
  
a = 2  
b = 8  
print(f"a:{a}, b:{b}")  
for i in a, b:  
    i = i + 1  
    print(f"Kopie: {i}")  
print(f"a:{a}, b:{b}")
```

Listing 3.18 Datei »schleife_for.py«

Folgende Ausgabe wird erzeugt:

```
Zahl: 8, Quadrat: 64  
Zahl: 3, Quadrat: 9  
Zahl: 7, Quadrat: 49  
  
Stadt: Paris  
Stadt: Rom  
Stadt: Madrid  
  
Zeichen: R  
Zeichen: o  
Zeichen: m  
  
a:2, b:8  
Kopie: 3  
Kopie: 9  
a:2, b:8
```

Bei der ersten `for`-Schleife wird eine Abfolge von Zahlen durchlaufen. Innerhalb der Schleife ist jede dieser Zahlen über die Variable `zahl` erreichbar und wird zusammen mit ihrem Quadrat ausgegeben.

In der zweiten `for`-Schleife wird eine Abfolge von Zeichenketten durchlaufen. Innerhalb der Schleife ist jede dieser Zeichenketten

über die Variable `stadt` erreichbar und wird zusammen mit einem Text ausgegeben.

Bei der dritten `for`-Schleife wird eine einzelne Zeichenkette durchlaufen. Zeichenketten bestehen aus einer Abfolge von Zeichen. Innerhalb der Schleife ist jedes dieser Zeichen über die Variable `zeichen` erreichbar und wird zusammen mit einem Text ausgegeben.

In der vierten `for`-Schleife wird eine Abfolge von Variablen durchlaufen. Innerhalb der Schleife ist jeweils eine Kopie der aktuellen Variablen über die Variable `i` erreichbar. Die Kopie wird verändert und ausgegeben. Die Werte der Originalvariablen verändern sich nicht.

Wie bei einer Verzweigung mit `if` gilt: Es muss ein Doppelpunkt notiert werden, und die Anweisungen innerhalb der Schleife müssen eingerückt werden. Sie sehen, dass mithilfe eines String-Literals auch das Ergebnis eines berechneten Ausdrucks in eine Zeichenkette eingebettet werden kann.

3.4.2 Schleifenabbruch mit »break«

Das Schlüsselwort `break` führt zu dem unmittelbaren Abbruch einer Schleife. Ein solcher Abbruch wird meist mit einer Bedingung verbunden und häufig bei Sonderfällen eingesetzt. Ein Beispiel:

```
for i in 12, -4, 20, 7:  
    if i*i > 200:  
        break  
    print(f"Zahl: {i}, Quadrat: {i*i}")
```

Listing 3.19 Datei »schleife_break.py«

Diese Ausgabe wird erzeugt:

```
Zahl: 12, Quadrat: 144  
Zahl: -4, Quadrat: 16
```

Die Schleife wird unmittelbar verlassen, wenn das Quadrat der aktuellen Zahl größer als 200 ist. Die Ausgabe innerhalb der

Schleife erfolgt auch nicht mehr.

Das Schlüsselwort `break` kann wie hier bei einer `for`-Schleife eingesetzt werden oder auch bei einer `while`-Schleife, siehe [Abschnitt 3.4.8](#), »Schleife mit ›while‹«.

3.4.3 Schleifenfortsetzung mit »continue«

Das Schlüsselwort `continue` dient zum unmittelbaren Abbruch des aktuellen Durchlaufs einer Schleife. Das Programm wird anschließend mit dem nächsten Durchlauf der Schleife fortgesetzt. Betrachten Sie hierzu das folgende Programm:

```
for i in range (1,7):
    print("Zahl:", i)
    if 3 <= i <= 5:
        continue
    print("Quadrat:", i*i)
```

Listing 3.20 Datei »schleife_continue.py«

Die Ausgabe dieses Programms ist:

```
Zahl: 1
Quadrat: 1
Zahl: 2
Quadrat: 4
Zahl: 3
Zahl: 4
Zahl: 5
Zahl: 6
Quadrat: 36
```

Die Schleife durchläuft alle Zahlen von 1 bis 6. Alle diese Zahlen werden auch ausgegeben. Liegt die aktuelle Zahl zwischen 3 und 5, wird der Rest der Schleife übergangen und unmittelbar der nächste Schleifendurchlauf begonnen. Andernfalls wird das Quadrat der Zahl ausgegeben.

3.4.4 Geschachtelte Kontrollstrukturen

Wie die beiden letzten Programme zeigen, können Kontrollstrukturen (also Verzweigungen und Schleifen)

geschachtelt werden. Dies bedeutet, dass eine Kontrollstruktur eine weitere Kontrollstruktur enthält. Diese kann ihrerseits wiederum eine Kontrollstruktur enthalten usw.

Dazu ein weiteres Beispiel:

```
for x in -2, -1, 0, 1, 2:  
    if x > 0:  
        print(x, "positiv")  
    else:  
        if x < 0:  
            print(x, "negativ")  
        else:  
            print(x, "gleich 0")
```

Listing 3.21 Datei »schachtelung.py«

Es wird diese Ausgabe erzeugt:

```
-2 negativ  
-1 negativ  
0 gleich 0  
1 positiv  
2 positiv
```

Die äußerste Kontrollstruktur ist eine `for`-Schleife. Alle einfach eingerückten Anweisungen werden – gemäß der Schleifensteuerung – mehrmals ausgeführt.

Mit der äußeren `if`-Anweisung wird die erste Verzweigung eingeleitet. Ist `x` größer als 0, wird die folgende zweifach eingerückte Anweisung ausgeführt. Ist `x` nicht größer als 0, wird die innere `if`-Anweisung hinter der äußeren `else`-Anweisung untersucht.

Trifft die Bedingung der inneren `if`-Anweisung zu, werden die folgenden dreifach eingerückten Anweisungen ausgeführt. Trifft sie nicht zu, werden die dreifach eingerückten Anweisungen ausgeführt, die der inneren `else`-Anweisung folgen.

Zu beachten sind besonders die mehrfachen Einrückungen, damit die Tiefe der Kontrollstruktur von Python richtig erkannt werden kann.

Hinweis

Nach einem Doppelpunkt hinter dem Kopf einer Kontrollstruktur wird in der Entwicklungsumgebung IDLE automatisch mit einem Tabulatorsprung eingerückt. Dieser Sprung erzeugt standardmäßig vier Leerzeichen, dadurch werden die Kontrollstrukturen klar erkennbar.

Arbeiten Sie mit einem anderen Editor, müssen Sie darauf achten, dass um mindestens ein Leerzeichen eingerückt wird, damit Python die Kontrollstruktur erkennt. Sinnvoller ist eine Einrückung um zwei oder mehr Leerzeichen, damit die Struktur gut erkennbar ist.

3.4.5 Spiel, Version mit »for«-Schleife und Abbruch

Die `for`-Schleife wird nun dazu genutzt, die Eingabe und die Bewertung des Kopfrechenspiels insgesamt viermal zu durchlaufen. Der Benutzer hat somit vier Versuche, das richtige Ergebnis zu ermitteln. Die Anweisung `break` dient zum Abbruch der Schleife, sobald der Benutzer das richtige Ergebnis eingegeben hat.

```
import random
random.seed()

a = random.randint(1,10)
b = random.randint(1,10)
c = a + b
print(f"Die Aufgabe: {a} + {b}")

for i in 1, 2, 3, 4:
    print("Bitte Lösungsvorschlag eingeben:")
    zahl = int(input())
    if zahl == c:
        print(zahl, "ist richtig")
        break
    else:
        print(zahl, "ist falsch")

print("Ergebnis:", c)
```

Listing 3.22 Datei »spiel_for.py«

Folgende Ausgabe wird erzeugt:

```
Die Aufgabe: 7 + 9
Bitte Lösungsvorschlag eingeben:
12
12 ist falsch
Bitte Lösungsvorschlag eingeben:
16
16 ist richtig
Ergebnis: 16
```

Die Aufgabe wird einmal ermittelt und gestellt. Der Benutzer wird maximal viermal dazu aufgefordert, ein Ergebnis einzugeben. Jede seiner Eingaben wird bewertet. Ist bereits einer der ersten drei Versuche richtig, wird die Schleife vorzeitig abgebrochen.

3.4.6 Schleife mit »for« und »range()«

Meist werden Schleifen für regelmäßige Abfolgen von Zahlen genutzt. Dabei erweist sich der Einsatz der eingebauten Funktion `range()` als sehr nützlich. Ein Beispiel:

```
for i in range(3,11,2):
    print(f"Zahl: {i}, Quadrat: {i*i}")
```

Listing 3.23 Datei »range_drei.py«

Es wird diese Ausgabe erzeugt:

```
Zahl: 3, Quadrat: 9
Zahl: 5, Quadrat: 25
Zahl: 7, Quadrat: 49
Zahl: 9, Quadrat: 81
```

Der englische Begriff *range* bedeutet *Bereich*. Innerhalb der Klammern hinter `range` können maximal drei ganze Zahlen, durch Kommata getrennt, eingetragen werden:

- Die erste ganze Zahl (hier `3`) gibt den Beginn des Bereichs an, für den die folgenden Anweisungen ausgeführt werden.
- Die zweite ganze Zahl (hier `11`) kennzeichnet das Ende des Bereichs. Es ist die erste Zahl, für die die Anweisungen *nicht* mehr ausgeführt werden.

- Die dritte ganze Zahl (hier 2) gibt die Schrittweite für die Schleife an. Die Zahlen, für die die Anweisungen ausgeführt werden, stehen zueinander also jeweils im Abstand von +2.

Der Aufruf der Funktion `range()` mit den Zahlen 3, 11 und 2 ergibt somit die Abfolge: 3, 5, 7, 9.

Wird die Funktion `range()` nur mit zwei Zahlen aufgerufen, so wird eine Schrittweite von 1 angenommen. Ein Beispiel:

```
for i in range(5, 9):
    print("Zahl:", i)
```

Listing 3.24 Datei »range_zwei.py«

Es wird folgende Ausgabe erzeugt:

```
Zahl: 5
Zahl: 6
Zahl: 7
Zahl: 8
```

Wird die Funktion `range()` nur mit einer Zahl aufgerufen, so wird diese Zahl als die obere Grenze angesehen. Als untere Grenze gilt 0. Außerdem wird wiederum eine Schrittweite von 1 angenommen. Zugleich kennzeichnet die Zahl die Anzahl der Durchläufe. Ein Beispiel:

```
for i in range(3):
    print("Zahl:", i)
```

Listing 3.25 Datei »range_eins.py«

Die Ausgabe:

```
Zahl: 0
Zahl: 1
Zahl: 2
```

Hinweis

Bei der Funktion `range()` können eine oder mehrere der drei Zahlen auch negativ sein. Achten Sie auf sinnvolle Zahlenkombinationen. Die Angabe `range(3, -11, 2)` ist nicht sinnvoll, da man von der Zahl +3 in Schritten von +2 nicht zur Zahl -11 gelangt. Python fängt solche Schleifen ab und lässt sie

nicht ausführen. Dasselbe gilt für die Zahlenkombination

```
range(3,11,-2).
```

Für den regelmäßigen Ablauf von Zahlen mit Nachkommastellen sollte die Schleifenvariable passend umgerechnet werden. Ein Beispiel:

```
for x in range(18,22):
    print(x/10)
print()

x = 1.8
for i in range(4):
    print(x)
    x = x + 0.1
```

Listing 3.26 Datei »range_nachkomma.py«

Die Ausgabe lautet:

```
1.8
1.9
2.0
2.1

1.8
1.9000000000000001
2.0
2.1
```

Es werden jeweils die Zahlen von 1,8 bis 2,1 in Schritten von 0,1 erzeugt.

- In der ersten Version werden die ganzen Zahlen von 18 bis 21 erzeugt und anschließend durch 10 geteilt.
- In der zweiten Version wird mit dem ersten Wert begonnen und innerhalb der Schleife jeweils um 0,1 erhöht. Dabei müssen Sie vorher errechnen, wie häufig die Schleife durchlaufen werden muss.

Übung »u_range«

Überlegen Sie, welche Ausgabe das folgende Programm hat (Datei *u_range.py*). Kontrollieren Sie Ihre Ergebnisse anschließend durch einen Aufruf.

```

print("Schleife 1")
for i in 2, 3, 6.5, -7:
    print(i)

print("Schleife 2")
for i in range(3,11,3):
    print(i)

print("Schleife 3")
for i in range(-3,14,4):
    print(i)

print("Schleife 4")
for i in range(3,-11,-3):
    print(i)

```

Übung »u_range_inch«

Schreiben Sie ein Programm, das die folgende Ausgabe erzeugt (Datei *u_range_inch.py*).

```

15 Inch = 38.1 cm
20 Inch = 50.8 cm
25 Inch = 63.5 cm
30 Inch = 76.2 cm
35 Inch = 88.9 cm
40 Inch = 101.6 cm

```

Es handelt sich um eine regelmäßige Liste von Inch-Werten, für die der jeweilige Zentimeter-Wert durch Umrechnung mit dem Faktor 2,54 ermittelt wird. Es ist keine Eingabe durch die Anwenderin notwendig.

3.4.7 Spiel, Version mit »range()«

Im Kopfrechenspiel wird die Schleife zur Wiederholung der Eingabe nun mithilfe von `range()` gebildet. Gleichzeitig haben Sie damit einen Zähler, der die laufende Nummer des Versuchs enthält. Sie können ihn verwenden, um dem Benutzer die Anzahl der Versuche mitzuteilen.

```

import random
random.seed()

a = random.randint(1,10)
b = random.randint(1,10)
c = a + b

```

```

print(f"Die Aufgabe: {a} + {b}")

for versuch in range(1,10):
    print("Bitte Lösungsvorschlag eingeben:")
    zahl = int(input())

    if zahl == c:
        print(zahl, "ist richtig")
        break
    else:
        print(zahl, "ist falsch")

print("Ergebnis:", c)
print("Anzahl der Versuche:", versuch)

```

Listing 3.27 Datei »spiel_range.py«

Die Ausgabe lautet:

```

Die Aufgabe: 10 + 5
Bitte Lösungsvorschlag eingeben:
13
13 ist falsch
Bitte Lösungsvorschlag eingeben:
15
15 ist richtig
Ergebnis: 15
Anzahl der Versuche: 2

```

Der Benutzer hat maximal neun Versuche: `range(1,10)`. Die Variable `versuch` dient als Zähler für die Versuche. Nach Eingabe der richtigen Lösung (oder nach vollständigem Durchlauf der Schleife) wird dem Benutzer die Anzahl der Versuche mitgeteilt.

3.4.8 Schleife mit »while«

Die `while`-Schleife dient zur Steuerung einer Wiederholung mithilfe einer Bedingung. Im folgenden Programm werden zufällige Zahlen addiert und ausgegeben. Solange die Summe der Zahlen kleiner als 30 ist, wird der Vorgang wiederholt. Ist die Summe gleich oder größer als 30, wird das Programm beendet.

```

import random
random.seed()

summe = 0
while summe < 30:
    zzahl = random.randint(1,8)
    summe = summe + zzahl
    print(f"Zahl: {zzahl}, Zwischensumme: {summe}")

```

Listing 3.28 Datei »schleife_while.py«

Eine mögliche Ausgabe des Programms sähe wie folgt aus:

```
Zahl: 3, Zwischensumme: 3
Zahl: 8, Zwischensumme: 11
Zahl: 5, Zwischensumme: 16
Zahl: 8, Zwischensumme: 24
Zahl: 7, Zwischensumme: 31
```

Zunächst wird die Variable für die Summe der Zahlen auf 0 gesetzt.

Die `while`-Anweisung leitet die Schleife ein. Die wörtliche Übersetzung der Zeile lautet: *solange die Summe kleiner als 30 ist*. Dies bezieht sich auf die nachfolgenden eingerückten Anweisungen.

Nach dem Wort `while` folgt eine Bedingung, die mithilfe von Vergleichsoperatoren erstellt wird. Auch hier dürfen Sie den Doppelpunkt am Ende der Zeile nicht vergessen, ähnlich wie bei `if-else` und `for`.

Es wird eine zufällige Zahl ermittelt und zur bisherigen Summe addiert. Die neue Summe errechnet sich also aus der alten Summe plus der neuen zufälligen Zahl. Die neue Summe wird ausgegeben.

Das Ende der Schleife (und hier des Programms) wird erst erreicht, wenn die Summe den Wert 30 erreicht oder überschritten hat.

3.4.9 Spiel, Version mit »while«-Schleife und Zähler

Die `while`-Schleife wird nun auch im Kopfrechenspiel zur Wiederholung der Eingabe genutzt. Die Benutzerin hat beliebig viele Versuche, die Aufgabe zu lösen. Die Variable `versuch`, die als Zähler für die Versuche dient, muss separat gesteuert werden. Sie entspricht nicht mehr automatisch der Schleifenvariablen.

```
import random
random.seed()
```

```

a = random.randint(1,10)
b = random.randint(1,10)
c = a + b
print(f"Die Aufgabe: {a} + {b}")

zahl = c + 1
versuch = 0
while zahl != c:
    versuch = versuch + 1
    print("Bitte Lösungsvorschlag eingeben:")
    zahl = int(input())
    if zahl == c:
        print(zahl, "ist richtig")
    else:
        print(zahl, "ist falsch")

print("Ergebnis:", c)
print("Anzahl der Versuche:", versuch)

```

Listing 3.29 Datei »spiel_while.py«

Die Ausgabe hat sich gegenüber der letzten Version nicht geändert.

Die Benutzerin hat beliebig viele Versuche. Die `while`-Schleife läuft, solange die richtige Lösung nicht ermittelt wird. Die Variable `zahl` wird mit einem Wert vorbesetzt, der dafür sorgt, dass die `while`-Schleife mindestens einmal läuft. Die Variable `versuch` wird mit 0 vorbesetzt und dient als laufende Nummer. Nach Eingabe der richtigen Lösung wird der Benutzerin die Anzahl der Versuche mitgeteilt.

Übung »u_while«

Schreiben Sie ein Programm (Datei `u_while.py`), das den Anwender wiederholt dazu auffordert, einen Wert in Inch einzugeben. Der eingegebene Wert soll anschließend in Zentimeter umgerechnet und ausgegeben werden. Das Programm soll nach der Eingabe des Werts 0 beendet werden.

Bei einer `while`-Schleife wird immer angegeben, gemäß welcher Bedingung wiederholt werden soll, und nicht, gemäß welcher Bedingung beendet werden soll. Daher müssen Sie in

diesem Programm formulieren: *Solange die Eingabe ungleich 0 ist.*

3.4.10 Kombinierte Zuweisungsausdrücke

Mit Python 3.8 wurde der Operator `:=` für kombinierte Zuweisungsausdrücke eingeführt. Er führt auch den Spitznamen *walrus-operator*, aufgrund der Ähnlichkeit mit den Augen und den Zähnen eines Walrosses. Er ermöglicht kürzere Ausdrücke, in denen allerdings eine höhere Komplexität steckt. Nachfolgend sehen Sie, wie Sie mithilfe des Operators das Beispiel aus [Abschnitt 3.4.8](#), »Schleife mit ›while‹«, verkürzen können:

```
import random
random.seed()
summe = 0
while (summe := summe + random.randint(1,8)) < 30:
    print("Zwischensumme:", summe)
```

Listing 3.30 Datei »schleife_zuweisung.py«

Die Zeile, die mit dem Schlüsselwort `while` beginnt, enthält mehrere Abläufe. Zunächst wird die Summe um einen zufällig ermittelten Wert erhöht. Anschließend wird die neue Summe mit der Zahl 30 verglichen. Dieser Vergleich steuert die Schleife. Aufgrund des niedrigen Vorrangs des Operators `:=` muss die Zuweisung in runden Klammern stehen.

Eine mögliche Ausgabe des Programms:

```
Zwischensumme: 1
Zwischensumme: 9
Zwischensumme: 11
Zwischensumme: 17
Zwischensumme: 23
Zwischensumme: 29
```

3.5 Entwicklung eines Programms

Bei der Entwicklung Ihrer eigenen Programme sollten Sie Schritt für Schritt vorgehen. Stellen Sie zuerst einige Überlegungen dazu an, wie das gesamte Programm aufgebaut sein sollte, und zwar auf Papier. Aus welchen Teilen sollte es nacheinander bestehen? Versuchen Sie anschließend nicht, das gesamte Programm mit all seinen komplexen Bestandteilen auf einmal zu schreiben! Dies ist der größte Fehler, den Einsteiger (und manchmal auch Fortgeschrittene) machen können.

Schreiben Sie zunächst eine einfache Version des ersten Programmteils. Anschließend testen Sie sie. Erst nach einem erfolgreichen Test fügen Sie den folgenden Programmteil hinzu. Nach jeder Änderung testen Sie wiederum. Sollte sich ein Fehler zeigen, wissen Sie, dass er aufgrund der letzten Änderung aufgetreten ist. Nach dem letzten Hinzufügen haben Sie eine einfache Version Ihres gesamten Programms erstellt.

Nun ändern Sie einen Teil Ihres Programms in eine komplexere Version ab. Auf diese Weise machen Sie Ihr Programm Schritt für Schritt komplexer, bis Sie schließlich das gesamte Programm so erstellt haben, wie es Ihren anfänglichen Überlegungen auf Papier entspricht.

Manchmal ergibt sich während der praktischen Programmierung noch die eine oder andere Änderung gegenüber Ihrem Entwurf. Das ist kein Problem, solange sich nicht der gesamte Aufbau ändert. Sollte dies allerdings der Fall sein, kehren Sie noch einmal kurz zum Papier zurück und überdenken Sie den Aufbau. Das bedeutet nicht, dass Sie die bisherigen Programmzeilen löschen müssen, sondern möglicherweise nur ein wenig ändern und anders anordnen.

Schreiben Sie Ihre Programme übersichtlich. Falls Sie gerade überlegen, wie Sie drei, vier bestimmte Schritte Ihres Programms auf einmal machen können: Machen Sie daraus besser einzelne Anweisungen, die der Reihe nach ausgeführt werden. Dies vereinfacht eine eventuelle Fehlersuche. Ändern oder erweitern Sie (oder eine andere Person) Ihr Programm später einmal, gelingt der Einstieg in den Aufbau des Programms wesentlich schneller.

Sie können die Funktion `print()` zur Kontrolle von Werten und zur Suche von logischen Fehlern einsetzen. Zusätzlich können Sie einzelne Zeilen Ihres Programms als Kommentar kennzeichnen, um festzustellen, welcher Teil des Programms fehlerfrei läuft und welcher Teil demnach fehlerbehaftet ist.

3.6 Fehler und Ausnahmen

Zur Laufzeit eines Programms können Fehler auftreten, die Sie nicht voraussehen können. Gibt zum Beispiel ein Benutzer nach der Eingabeaufforderung für eine Zahl keine gültige Zahl ein, tritt eine Ausnahme auf, und das Programm wird bei der Umwandlung der Eingabe mit `int()` oder `float()` mit einer Fehlermeldung beendet. Bisher gehe ich vereinfacht davon aus, dass der Anwender korrekte Eingaben vornimmt. In diesem Abschnitt beschreibe ich, wie Sie die Folgen von Fehlern vermeiden oder abfangen.

3.6.1 Basisprogramm

Das folgende Programm dient zur Demonstration eines Fehlers.

```
print("Bitte geben Sie eine ganze Zahl ein")
z = input()
zahl = int(z)
print(f"Sie haben die ganze Zahl {zahl} richtig eingegeben")
```

Listing 3.31 Datei »fehler_basis.py«

Macht der Benutzer eine falsche Eingabe (zum Beispiel »3a«), bricht das Programm bei der Umwandlung ab und erzeugt die folgende Ausgabe:

```
Bitte geben Sie eine ganze Zahl ein
3a
Traceback (most recent call last):
  File "C:\Python\Beispiele\fehler_basis.py", line 3, in <module>
    zahl = int(z)
ValueError: invalid literal for int() with base 10: '3a'
```

Diese Informationen weisen auf die Stelle im Programm hin, an der ein Fehler bemerkt wird (Datei *fehler_basis.py*, Zeile 3). Außerdem wird die Art des Fehlers mitgeteilt (`ValueError`).

3.6.2 Fehler abfangen

In einem ersten Schritt sollen die Folgen einer Fehleingabe abgefangen werden, um einen Abbruch des Programms zu vermeiden. Zu diesem Zweck müssen Sie die Stelle bestimmen, an der ein Fehler auftreten kann, den Python erkennen kann. Hier müssen Sie das Programm verbessern. Dies erreichen Sie in einem ersten Schritt durch die folgende Änderung:

```
print("Bitte geben Sie eine ganze Zahl ein")
z = input()

try:
    zahl = int(z)
    print(f"Sie haben die ganze Zahl {zahl} eingegeben")
except:
    print("Fehler bei Umwandlung der Eingabe")
```

Listing 3.32 Datei »fehler_abfangen.py«

Wenn der Anwender eine richtige ganze Zahl eingibt, läuft das Programm wie bisher. Nach einer falschen Eingabe bricht das Programm bei der Umwandlung nicht ab, sondern gibt eine Meldung aus.

```
Bitte geben Sie eine ganze Zahl ein
3a
Fehler bei Umwandlung der Eingabe
```

Die Anweisung `try` leitet eine Ausnahmebehandlung ein. Ähnlich wie bei einer Verzweigung gibt es verschiedene Zweige, die das Programm durchlaufen kann. Das Programm versucht (englisch: *try*), die Anweisungen durchzuführen, die eingerückt nach `try` stehen. Ist die Umwandlung erfolgreich, wird der `except`-Zweig nicht ausgeführt, ähnlich wie beim `else`-Zweig der `if`-Anweisung.

Ist die Umwandlung dagegen nicht erfolgreich, wird der Fehler oder die Ausnahme (englisch: *exception*) mit der Anweisung `except` abgefangen. In diesem Fall werden alle eingerückten Anweisungen im `except`-Zweig durchgeführt. Das Programm läuft ohne Abbruch zu Ende, da der Fehler zwar auftritt, aber abgefangen wird.

Nach `try` und `except` muss jeweils ein Doppelpunkt gesetzt werden.

Hinweis

Nur der *kritische Bereich* Ihres Programms wird in die Ausnahmebehandlung eingebettet. Sie sollten sich also Gedanken darüber machen, welche Stellen Ihres Programms fehlerträchtig sind. Die Umwandlung einer Eingabe ist solch eine kritische Stelle. Andere Fehlermöglichkeiten sind zum Beispiel die Bearbeitung einer Datei (die möglicherweise nicht existiert) oder die Ausgabe an einen Drucker (der vielleicht nicht eingeschaltet ist).

3.6.3 Eingabe wiederholen

In einem zweiten Schritt wird dafür gesorgt, dass die Anwenderin nach einer falschen Eingabe eine erneute Eingabe machen kann. Der gesamte Eingabevorgang mit Ausnahmebehandlung wird so lange wiederholt, bis die Eingabe erfolgreich war. Betrachten Sie das folgende Programm:

```
while True:
    print("Bitte geben Sie eine ganze Zahl ein")
    z = input()
    try:
        zahl = int(z)
        print(f"Sie haben die ganze Zahl {zahl} eingegeben")
        break
    except:
        print("Fehler bei Umwandlung der Eingabe")
```

Listing 3.33 Datei »fehler_eingabe_neu.py«

Nachfolgend wird eine mögliche Eingabe gezeigt – zunächst mit einem Fehler, anschließend fehlerfrei:

```
Bitte geben Sie eine ganze Zahl ein
3a
Fehler bei Umwandlung der Eingabe
Bitte geben Sie eine ganze Zahl ein
12
Sie haben die ganze Zahl 12 eingegeben
```

Mithilfe des Wahrheitswerts `True` wird eine endlos wiederholte `while`-Schleife formuliert, in die der Eingabevorgang mit der Ausnahmebehandlung eingebettet ist. Ist die Eingabe erfolgreich,

wird die Schleife mithilfe von `break` verlassen. Ist die Eingabe nicht erfolgreich, wird der Eingabevorgang wiederholt.

Übung »u_fehler«

Verbessern Sie das Programm zur Eingabe und Umrechnung eines beliebigen Inch-Werts in Zentimeter. Die Folgen eines Eingabefehlers des Anwenders sollen abgefangen werden. Das Programm soll den Anwender so lange zur Eingabe auffordern, bis sie erfolgreich war (Datei `u_fehler.py`).

3.6.4 Spiel, Version mit Ausnahmebehandlung

Die Ausnahmebehandlung und das Schlüsselwort `continue` werden nun auch im Kopfrechenspiel eingesetzt. Damit können die Folgen eines Eingabefehlers abgefangen und das Programm regulär fortgesetzt werden.

```
import random
random.seed()

a = random.randint(1,10)
b = random.randint(1,10)
c = a + b
print(f"Die Aufgabe: {a} + {b}")

zahl = c + 1
versuch = 0
while zahl != c:
    versuch = versuch + 1
    print("Bitte eine ganze Zahl als Lösungsvorschlag eingeben:")
    z = input()

    try:
        zahl = int(z)
    except:
        print("Sie haben keine ganze Zahl eingegeben")
        continue

    if zahl == c:
        print(zahl, "ist richtig")
    else:
        print(zahl, "ist falsch")

print("Ergebnis:", c)
print("Anzahl der Versuche:", versuch)
```

Listing 3.34 Datei »spiel_ausnahme.py«

Es wird die folgende Ausgabe erzeugt:

```
Die Aufgabe: 8 + 3
Bitte eine ganze Zahl als Lösungsvorschlag eingeben:
12
12 ist falsch
Bitte eine ganze Zahl als Lösungsvorschlag eingeben:
11a
Sie haben keine ganze Zahl eingegeben
Bitte eine ganze Zahl als Lösungsvorschlag eingeben:
11
11 ist richtig
Ergebnis: 11
Anzahl der Versuche: 3
```

Die Umwandlung der Eingabe steht in einem `try-except`-Block. Gelingt die Umwandlung aufgrund einer falschen Eingabe nicht, erscheint eine entsprechende Meldung. Der Rest der Schleife wird übergangen, und die nächste Eingabe wird unmittelbar angefordert.

3.7 Funktionen und Module

Die Modularisierung, also die Zerlegung eines Programms in Funktionen, bietet besonders bei größeren Programmen unübersehbare Vorteile:

- Mehrfach benötigte Programmteile werden nur einmal definiert.
- Programmteile können in mehreren Programmen verwendet werden.
- Umfangreiche Programme werden in übersichtliche Teile zerlegt.
- Pflege und Wartung von Programmen werden erleichtert.
- Der Programmcode ist für den Urheber selbst (zu einem späteren Zeitpunkt) und für andere Programmierer und Programmiererinnen leichter zu verstehen.

Neben den eigenen Funktionen gibt es in Python zahlreiche vordefinierte Funktionen, die dem Entwickler viel Arbeit abnehmen. Sie sind entweder eingebaut oder über die Einbindung spezieller Module verfügbar.

Sie setzen zum Beispiel bereits die eingebaute Funktion `input()` ein. Jede Funktion hat eine spezielle Aufgabe. So hält die Funktion `input()` das Programm an und nimmt eine Eingabe entgegen.

Viele Funktionen haben einen sogenannten Rückgabewert. Sie liefern ein Ergebnis an die Stelle des Programms zurück, von der sie aufgerufen werden. Im Fall von `input()` ist das die eingegebene Zeichenkette.

Hinweis

Für fortgeschrittene Leserinnen und Leser, die bereits mit einer anderen Programmiersprache gearbeitet haben: Funktionen können in Python nicht überladen werden. Definieren Sie eine Funktion mehrfach, gegebenenfalls mit unterschiedlichen Parametern, so gilt nur die jeweils letzte Definition.

3.7.1 Einfache Funktionen

Einfache Funktionen führen bei ihrem Aufruf stets die gleiche Aktion aus. Im folgenden Beispiel führt jeder Aufruf der Funktion `stern()` dazu, dass eine optische Trennung auf dem Bildschirm ausgegeben wird:

```
# Definition der Funktion
def stern():
    print("-----")
    print("*** Trennung ***")
    print("-----")

# Programm
x = 12
y = 5
stern()                      # 1. Aufruf
print("x =", x, ", y =", y)
stern()                      # 2. Aufruf
print("x + y =", x + y)
stern()                      # 3. Aufruf
print("x - y =", x - y)
stern()                      # 4. Aufruf
```

Listing 3.35 Datei »funktion_einfach.py«

Die Ausgabe sehen Sie in [Abbildung 3.1](#).

Zunächst wird die Funktion `stern()` definiert. Nach dem Schlüsselwort `def` folgt der Name der Funktion, anschließend runde Klammern und der bereits bekannte Doppelpunkt. Innerhalb der Klammern könnten Werte an die Funktion übergeben werden. Dazu mehr ab [Abschnitt 3.7.2](#), »Funktionen mit einem Parameter«. Die nachfolgenden eingerückten Anweisungen werden jedes Mal durchgeführt, wenn die Funktion aufgerufen wird.

```
*** Trennung ***  
-----  
x = 12 , y = 5  
-----  
*** Trennung ***  
-----  
x + y = 17  
-----  
*** Trennung ***  
-----  
x - y = 7  
-----  
*** Trennung ***
```

Abbildung 3.1 Einfache Funktionen

Eine Funktion wird zunächst nur definiert und steht zum späteren Gebrauch bereit. Das ausgeführte Programm beginnt mit einigen Rechenoperationen. Mithilfe der vier Aufrufe der Funktion `stern()` werden die Ausgabezeilen optisch voneinander getrennt. Nach Bearbeitung der Funktion fährt das Programm jeweils mit der Anweisung fort, die dem Aufruf der Funktion folgt.

Eine Funktion wird aufgerufen, indem Sie ihren Namen, gefolgt von runden Klammern, notieren. Möchten Sie Informationen an die Funktion übergeben, notieren Sie diese innerhalb der runden Klammern.

Den Namen einer Funktion können Sie weitgehend frei wählen – es gelten die gleichen Regeln wie bei den Namen von Variablen, siehe auch [Abschnitt 2.1.5](#), »Variablen und Zuweisung«: Der Name kann aus den Buchstaben »a« bis »z«, »A« bis »Z«, aus Ziffern und dem Zeichen `_` (Unterstrich) bestehen. Er darf nicht mit einer Ziffer beginnen und keinem reservierten Wort in Python entsprechen.

3.7.2 Funktionen mit einem Parameter

Bei dem Aufruf einer Funktion können auch Informationen übermittelt werden, sogenannte *Parameter*. Diese Parameter

werden innerhalb der Funktion ausgewertet und führen bei jedem Aufruf zu unterschiedlichen Ergebnissen. Ein Beispiel:

```
# Definition der Funktion
def quadrat(x):
    q = x*x
    print(f"Zahl: {x}, Quadrat: {q}")

# Programm
quadrat(4.5)
a = 3
quadrat(a)
quadrat(2*a)
```

Listing 3.36 Datei »parameter.py«

Die Ausgabe lautet:

```
Zahl: 4.5, Quadrat: 20.25
Zahl: 3, Quadrat: 9
Zahl: 6, Quadrat: 36
```

Die Definition der Funktion `quadrat()` enthält eine Variable innerhalb der Klammern. Beim Aufruf wird ein Wert an die Funktion übermittelt und dieser Variablen zugewiesen. Hier sind das die folgenden Werte:

- Die Zahl 4,5 – die Variable `x` erhält in der Funktion den Wert `4.5`.
- Der Wert der Variablen `a` – die Variable `x` erhält in der Funktion den aktuellen Wert von `a`, nämlich 3.
- Das Ergebnis einer Berechnung – die Variable `x` erhält in der Funktion den aktuellen Wert von `2 * a`, also 6.

Hinweis

Die Funktion erwartet genau einen Wert. Sie darf also nicht ohne einen Wert oder mit mehr als einem Wert aufgerufen werden, sonst bricht das Programm mit einer Fehlermeldung ab.

Übung »u_parameter«

Es soll wiederum die Steuer für verschiedene Gehälter berechnet werden (Datei `u_parameter.py`). Liegt das Gehalt

über 2.500 Euro, sind 22 % Steuern zu zahlen, ansonsten 18 %. Die Berechnung und die Ausgabe der Steuer sollen diesmal innerhalb einer Funktion mit dem Namen `steuer()` stattfinden. Die Funktion soll für die folgenden Gehälter aufgerufen werden: 1.800 Euro, 2.200 Euro, 2.500 Euro, 2.900 Euro.

3.7.3 Funktionen mit mehreren Parametern

Sie können einer Funktion auch mehrere Parameter übermitteln. Dabei ist auf die übereinstimmende Anzahl und die richtige Reihenfolge der Parameter zu achten. Ein Beispiel:

```
# Definition der Funktion
def berechnung(x,y,z):
    ergebnis = (x+y) * z
    print("Ergebnis:", ergebnis)

# Programm
berechnung(2,3,5)
berechnung(5,2,3)
```

Listing 3.37 Datei »parameter_mehrere.py«

Die Ausgabe lautet:

```
Ergebnis: 25
Ergebnis: 21
```

Es werden genau drei Parameter erwartet, bei beiden Aufrufen werden auch drei Werte übermittelt. Wie Sie am Ergebnis erkennen, ist die Reihenfolge der Parameter wichtig.

- Beim ersten Aufruf erhält `x` den Wert 2, `y` den Wert 3 und `z` den Wert 5. Dies ergibt die Rechnung: $(2 + 3) * 5 = 25$.
- Beim zweiten Aufruf werden dieselben Zahlen übergeben, aber in anderer Reihenfolge. Es ergibt sich die Rechnung: $(5 + 2) * 3 = 21$.

3.7.4 Funktionen mit Rückgabewert

Funktionen werden häufig zur Berechnung und Rücklieferung von Ergebnissen eingesetzt. Im Programm wird der sogenannte *Rückgabewert* gespeichert oder unmittelbar ausgegeben.

In Python können Funktionen mehr als einen Rückgabewert liefern, siehe [Abschnitt 5.6.4](#), »Mehrere Rückgabewerte«. In diesem Abschnitt werden aber zunächst nur Funktionen betrachtet werden, die genau einen Rückgabewert zur Verfügung stellen.

Nachfolgend wird eine solche Funktion definiert und mehrmals aufgerufen.

```
# Definition der Funktion
def mittelwert(x,y):
    ergebnis = (x+y) / 2
    return ergebnis

# Programm
c = mittelwert(3, 9)
print("Mittelwert:", c)

x = 5
print("Mittelwert:", mittelwert(x,4))

y = -5.1
z = 2.8
print(f"Mittelwert: {mittelwert(y,z)}")
```

Listing 3.38 Datei »rueckgabewert.py«

Diese Ausgabe wird erzeugt:

```
Mittelwert: 6.0
Mittelwert: 4.5
Mittelwert: -1.15
```

Innerhalb der Funktion wird zunächst das Ergebnis berechnet. Es wird anschließend mithilfe der Anweisung `return` an die aufrufende Stelle zurückgeliefert. Die Anweisung `return` beendet außerdem unmittelbar den Ablauf der Funktion.

Beim ersten Aufruf der Funktion wird der Rückgabewert in der Variablen `c` zwischengespeichert. Er kann im weiteren Verlauf des Programms an beliebiger Stelle verwendet werden.

Beim zweiten Aufruf werden mehrere Aktionen durchgeführt:
Die Funktion `mittelwert()` wird aufgerufen und liefert ein Ergebnis. Das Ergebnis wird unmittelbar ausgegeben.

Anhand des dritten Aufrufs sehen Sie, dass der Rückgabewert einer Funktion mithilfe eines String-Literals in eine Zeichenkette eingebettet werden kann.

3.7.5 Typhinweise

Seit Python 3.6 können Sie angeben, welchen Datentyp eine Variable haben soll. Diese Typhinweise (englisch: *type hints*) haben nur provisorischen Charakter. Ihre Regeln können sich ändern und sind nicht bindend. Seit Python 3.5 konnten Sie bereits mithilfe von Typhinweisen angeben, welchen Datentyp die Parameter und der Rückgabewert einer Funktion haben sollen.

Typhinweise sollen die Lesbarkeit des Codes verbessern und bessere Hilfestellungen der Editoren innerhalb einer Entwicklungsumgebung ermöglichen. Allerdings gilt nach wie vor eine Grundregel der Entwickler und Entwicklerinnen von Python: »Python wird eine dynamisch typisierte Sprache bleiben«.

Es folgt ein Programm mit Typhinweisen:

```
a:int = 42
b:float = 42.5
c:str = "Hallo"
d:bool = True
print("Variablen:", a, b, c, d)

def mittelwert(x:float, y:float) -> float:
    ergebnis = (x+y) / 2
    return ergebnis

print("Mittelwert:", mittelwert(3.4, 9.4))
```

Listing 3.39 Datei »typhinweise.py«

Der gewünschte Typ folgt nach dem Namen der Variablen und einem Doppelpunkt. Der Typ des Rückgabewerts einer Funktion folgt nach den Klammern der Parameterliste und der Zeichenfolge `->`, vor dem Doppelpunkt. Die Ausgabe des Programms:

```
Variablen: 42 42.5 Hallo True
Mittelwert: 6.4
```

3.7.6 Spiel, Version mit Funktionen

Die nachfolgende Version des Kopfrechenspiels umfasst zwei Funktionen. Sie dienen zur Ermittlung der Aufgabe und zur Bewertung der Eingabe:

```
# Funktion aufgabe()
def aufgabe():
    a = random.randint(1,10)
    b = random.randint(1,10)
    print(f"Die Aufgabe: {a} + {b}")
    return a + b

# Funktion kommentar()
def kommentar(eingabezah, ergebnis):
    if eingabezah == ergebnis:
        print(eingabezah, "ist richtig")
    else:
        print(eingabezah, "ist falsch")

# Programm
import random
random.seed()
c = aufgabe()
zahl = c + 1
versuch = 0

while zahl != c:
    versuch = versuch + 1
    print("Bitte Lösungsvorschlag eingeben:")
    z = input()
    try:
        zahl = int(z)
    except:
        print("Sie haben keine ganze Zahl eingegeben")
        continue
    kommentar(zahl,c)

print("Ergebnis:", c)
print("Anzahl der Versuche:", versuch)
```

Listing 3.40 Datei »spiel_funktion.py«

In der Funktion `aufgabe()` werden die beiden Zufallszahlen ermittelt, und die Aufgabe wird auf dem Bildschirm ausgegeben. Mithilfe von `return` können Sie auch das Ergebnis eines berechneten Ausdrucks (hier: das Ergebnis der Aufgabe) als Rückgabewert zurückliefern.

Der Funktion `Kommentar()` werden zwei Zahlen als Parameter übermittelt: die Lösung des Anwenders und das richtige Ergebnis. Innerhalb der Funktion wird die eingegebene Lösung untersucht, und ein entsprechender Kommentar wird ausgegeben. Die Funktion hat keinen Rückgabewert.

Übung »u_rueckgabewert«

Schreiben Sie das Programm aus Übung »u_parameter« um. Die Steuer soll innerhalb der Funktion `steuer()` berechnet und an das Hauptprogramm zurückgeliefert werden. Die Ausgabe des Werts soll im Hauptprogramm stattfinden (Datei `u_rueckgabewert.py`).

Übung »u_bedingt«

Schreiben Sie das Programm aus Übung »u_rueckgabewert« um. Die Verzweigung in der Funktion wird mithilfe eines bedingten Ausdrucks erstellt. Dessen Ergebnis wird unmittelbar mithilfe von `return` zurückgeliefert. Die Funktion enthält damit nur noch eine Anweisung (Datei `u_bedingt.py`).

3.8 Das fertige Spiel

Zum Abschluss des Programmierkurses erweitern wir das Kopfrechenspiel noch etwas – dabei nutzen wir viele der Programmiermittel, die Ihnen inzwischen zur Verfügung stehen. Die Erweiterungen:

- Es werden bis zu zehn Aufgaben nacheinander gestellt. Die Benutzerin kann dabei die Anzahl selbst bestimmen.
- Zusätzlich zur Addition kommen die weiteren Grundrechenarten zum Einsatz: Subtraktion, Multiplikation und Division.
- Die Bereiche, aus denen die zufälligen Zahlen gewählt werden, hängen von der Rechenart ab. Bei der Multiplikation wird zum Beispiel mit kleineren Zahlen gerechnet als bei der Addition.
- Die Benutzerin hat maximal drei Versuche pro Aufgabe.
- Die Anzahl der richtig gelösten Aufgaben wird ermittelt.

Die einzelnen Abschnitte des Programms sind nummeriert. Diese Nummern finden sich in der Erläuterung wieder. In späteren Kapiteln kommen weitere Ergänzungen hinzu. Es folgt das Programm:

```
# 1: Zufallsgenerator
import random
random.seed()

# 2: Anzahl Aufgaben
anzahl = -1
while anzahl<1 or anzahl>10:
    try:
        print("Wie viele Aufgaben (1 bis 10):")
        anzahl = int(input())
    except:
        continue

# 3: Anzahl richtige Ergebnisse
richtig = 0

# 4: Schleife mit gewünschter Anzahl an Aufgaben
```

```

for aufgabe in range(1,anzahl+1):

    # 5: Operatorauswahl
    opzahl = random.randint(1, 4)

    # 6: Operandenauswahl
    if opzahl == 1:
        a = random.randint(-10, 30)
        b = random.randint(-10, 30)
        op = "+"
        c = a + b
    elif opzahl == 2:
        a = random.randint(1, 30)
        b = random.randint(1, 30)
        op = "-"
        c = a - b
    elif opzahl == 3:
        a = random.randint(1, 10)
        b = random.randint(1, 10)
        op = "*"
        c = a * b

    # 7: Sonderfall Division
    elif opzahl == 4:
        c = random.randint(1, 10)
        b = random.randint(1, 10)
        op = "/"
        a = c * b

    # 8: Aufgabenstellung
    print(f"Aufgabe {aufgabe} von {anzahl}: {a} {op} {b}")

    # 9: Schleife mit 3 Versuchen
    for versuch in range(1,4):
        # 10: Eingabe
        try:
            print("Bitte Lösungsvorschlag eingeben:")
            zahl = int(input())
        except:
            # Umwandlung war nicht erfolgreich
            print("Sie haben keine ganze Zahl eingegeben")
            # Schleife unmittelbar fortsetzen
            continue

        # 11: Kommentar
        if zahl == c:
            print(zahl, "ist richtig")
            richtig = richtig + 1
            break
        else:
            print(zahl, "ist falsch")

    # 12: Richtiges Ergebnis der Aufgabe
    print("Ergebnis:", c)

# 13: Anzahl richtige Ergebnisse
print(f"Richtig: {richtig} von {anzahl}")

```

Listing 3.41 Datei »spiel_fertig.py«

Es wird die folgende Ausgabe erzeugt:

```
Wie viele Aufgaben (1 bis 10):  
2  
Aufgabe 1 von 2: 26 + 18  
Bitte Lösungsvorschlag eingeben:  
44  
44 ist richtig  
Ergebnis: 44  
Aufgabe 2 von 2: 27 - 2  
Bitte Lösungsvorschlag eingeben:  
24  
24 ist falsch  
Bitte Lösungsvorschlag eingeben:  
23  
23 ist falsch  
Bitte Lösungsvorschlag eingeben:  
22  
22 ist falsch  
Ergebnis: 25  
Richtig: 1 von 2
```

Nach der Initialisierung des Zufallsgenerators (1) wird die gewünschte Anzahl der Aufgaben eingelesen (2). Da der Benutzer einen Fehler bei der Eingabe machen könnte, findet eine Ausnahmebehandlung statt. Der Zähler für die Anzahl der richtig gelösten Aufgaben (`richtig`) wird auf 0 gestellt (3). Es wird eine äußere `for`-Schleife mit der gewünschten Anzahl gestartet (4).

Der Operator wird per Zufallsgenerator ermittelt (5). Für jeden Operator gibt es andere Bereiche, aus denen die Zahlen ausgewählt werden (6). Der Operator selbst und das Ergebnis werden gespeichert.

Eine Besonderheit ist bei der Division zu beachten (7): Es sollen nur ganze Zahlen vorkommen. Die beiden zufälligen Operanden (`a` und `b`) werden daher aus dem Ergebnis einer Multiplikation ermittelt.

Die Aufgabe wird gestellt (8). Dabei werden zur besseren Orientierung des Benutzers auch die laufende Nummer und die Gesamtanzahl der Aufgaben ausgegeben. Es wird eine innere `for`-Schleife für maximal drei Versuche gestartet (9).

Die Eingaben des Benutzers (10) werden kommentiert (11). Nach maximal drei Versuchen wird das richtige Ergebnis ausgegeben (12). Zuletzt wird die Anzahl der richtig gelösten Aufgaben ausgegeben (13).

4 Datentypen

In Python werden alle Daten in Objekten gespeichert. Dieses Kapitel beschäftigt sich mit den Eigenschaften und Vorteilen der verschiedenen Datentypen für die Objekte. Operationen, Funktionen und Operatoren für die jeweiligen Datentypen werden vorgestellt. Ein eigener Abschnitt über Objektreferenzen und Objektidentität vervollständigt die Betrachtung.

Es geht zunächst um Zahlen. Anschließend folgen Strings (= Zeichenketten), Listen, Tupel, Dictionarys und Sets. Gemeinsamkeiten und Unterschiede der Datentypen werden erläutert.

4.1 Zahlen

Ganze Zahlen, Zahlen mit Nachkommastellen, Brüche und Operationen mit Zahlen sind Thema dieses Abschnitts. Es gibt einige eingebaute Funktionen für Zahlen. Das Modul `math` enthält eine Reihe von mathematischen Funktionen zur Durchführung von Berechnungen.

4.1.1 Ganze Zahlen

Als Datentyp für ganze Zahlen dient `int` (von englisch *integer* für ganzzahlig). Mit Zahlen dieses Typs wird mathematisch genau gearbeitet.

Üblicherweise wird das dezimale Zahlensystem mit der Basis 10 benutzt. Außerdem stehen in Python die folgenden

Zahlensysteme zur Verfügung:

- das duale Zahlensystem (mit der Basis 2)
- das oktale Zahlensystem (mit der Basis 8)
- das hexadezimale Zahlensystem (mit der Basis 16)

Ein Beispiel:

```
a = 27
print("Dezimal:", a)
print("Hexadezimal:", hex(a))
print("Oktal:", oct(a))
print("Dual:", bin(a))

b = 0x1a + 12 + 0b101 + 0o67
print("Summe:", b)
```

Listing 4.1 Datei »zahl_ganz.py«

Folgende Ausgabe wird erzeugt:

```
Dezimal: 27
Hexadezimal: 0x1b
Oktal: 0o33
Dual: 0b11011
Summe: 98
```

Die dezimale Zahl 27 wird in die drei anderen Zahlensysteme umgerechnet und ausgegeben.

Die Funktion `hex()` dient zur Umrechnung und Ausgabe der Zahl in das hexadezimale System. Dieses System nutzt neben den Ziffern 0 bis 9 die Buchstaben »a« bis »f« (oder auch »A« bis »F«) als Ziffern für die Werte von 10 bis 15. Die Zahl 0x1b entspricht dem folgenden Wert:

$$1 \times 16^1 + B \times 16^0 = 1 \times 16^1 + 11 \times 16^0 = 16 + 11 = 27$$

Zur Umrechnung und Ausgabe der Zahl in das oktale System dient die Funktion `oct()`. Das oktale System nutzt nur die Ziffern 0 bis 7. Die Zahl 0o33 entspricht dem folgenden Wert:

$$3 \times 8^1 + 3 \times 8^0 = 24 + 3 = 27$$

Die Funktion `bin()` dient zur Umrechnung und Ausgabe der Zahl in das duale System. Dieses System nutzt nur die Ziffern 0 und 1.

Die Zahl 0b11011 entspricht dem folgenden Wert:

$$1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 16 + 8 + 2 + 1 = 27$$

Sie können auch direkt mit Zahlen in anderen Zahlensystemen rechnen. Die Berechnung der Variablen `b` ergibt:

$$0 \times 1a + 12 + 0b101 + 0o67 =$$

$$1 \times 16^1 + A \times 16^0 + 12 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 6 \times 8^1 + 7 \times 8^0 =$$

$$16 + 10 + 12 + 4 + 1 + 48 + 7 = 98$$

Bei der Eingabe oder Zuweisung muss das Präfix `0x`, `0b` bzw. `0o` vor den weiteren Ziffern stehen, damit das Zahlensystem erkannt wird.

Zahlen setzen sich auf der niedrigsten Ebene aus Bits und Bytes zusammen. In [Abschnitt 4.1.9](#), »Bitoperatoren«, werden Sie noch ein wenig intensiver mit Dualzahlen, der Funktion `bin()` und den sogenannten Bitoperatoren arbeiten, die Ihnen den Zugriff auf Bit-Ebene erleichtern.

4.1.2 Zahlen mit Nachkommastellen

Der Datentyp für Zahlen mit Nachkommastellen heißt `float`. Diese sogenannten Fließkommazahlen werden mithilfe eines Dezimalpunkts und gegebenenfalls in Exponentialschreibweise angegeben.

Dazu ein kleines Beispiel:

```
a = 7.5
b = 2e2
c = 3.5E3
d = 4.2e-3
e = 1_250_000.500_001

print(a, b, c, d, e)
```

Listing 4.2 Datei »zahl_nachkomma.py«

Die Ausgabe lautet:

```
7.5 200.0 3500.0 0.0042 1250000.500001
```

Die Variable `a` erhält den Wert 7.5 . Die Nachkommastellen folgen nach dem Dezimalpunkt. Dies gilt auch für die Eingabe einer Zahl mit Nachkommastellen mithilfe der Funktion `input()`. Die Variable `b` erhält den Wert 200 ($= 2 \times 10^2 = 2 \times 100$). Die Variable `c` erhält den Wert 3.500 ($= 3,5 \times 10^3 = 3,5 \times 1.000$), die Variable `d` den Wert $0,0042$ ($= 4,2 \times 10^{-3} = 4,2 \times 0,001$).

Bei der Zuweisung in Exponentialschreibweise wird mithilfe des Zeichens »e« (oder »E«) ausgedrückt, um wie viele Stellen und in welche Richtung der Dezimalpunkt innerhalb der Zahl verschoben wird. Diese Schreibweise eignet sich zum Beispiel für sehr große oder sehr kleine Zahlen, da sie die Eingabe vieler Nullen erspart.

Seit Python 3.6 können Sie einen Unterstrich benutzen, um Zahlen mit vielen Ziffern lesbarer zu machen. Es bietet sich an, ihn nach jeder dritten Ziffer einzufügen, wie es bei der Variablen `e` gemacht wurde.

4.1.3 Typ ermitteln

Es ist häufig nützlich zu wissen, ob es sich bei einer Zahl um eine ganze Zahl (Datentyp `int`) oder eine Fließkommazahl (Datentyp `float`) handelt. Die Funktion `type()` gibt den Typ (die Klasse) eines Objekts aus, nicht nur für Zahlentypen. Hierzu ein Programmbeispiel:

```
a = 2
print("Typ:", type(a))

b = 12/6
print("Typ:", type(b))

c = 12//6
print("Typ:", type(c))

d = 12%6 == 0
print("Vergleich liefert:", d)
print("Typ:", type(d))
```

Listing 4.3 Datei »zahl_type.py«

Das Programm erzeugt die folgende Ausgabe:

```
Typ: <class 'int'>
Typ: <class 'float'>
Typ: <class 'int'>
Vergleich liefert: True
Typ: <class 'bool'>
```

Die Variable `a` enthält den Wert 2 und ist vom Typ `int`. Die Variable `b` enthält den gleichen Wert, allerdings als Ergebnis einer mathematischen Division. Es handelt sich um ein Objekt des Typs `float`. Eine ganzzahlige Division ergibt einen Wert des Typs `int`.

Vergleichen Sie das Ergebnis einer Modulo-Operation mit 0, erhalten Sie die Information, ob eine Zahl durch eine andere Zahl glatt teilbar ist. Der Vergleich selbst liefert ein Objekt der Klasse `bool`.

4.1.4 Exponentialoperator `**`

Der Exponentialoperator `**` dient zur Berechnung von Potenzen, also eines Ausdrucks der Form *Basis hoch Exponent*. Es folgen einige Beispiele:

```
z = 5 ** 3
print("5 hoch 3 =", z)
z = -5.2 ** -3.8
print("-5.2 hoch -3.8 =", z)
z = 9 ** 0.5
print("Quadratwurzel aus 9 = 9 hoch 1/2 =", z)
z = 27 ** (1.0/3.0)
print("Kubikwurzel aus 27 = 27 hoch 1/3 =", z)
```

Listing 4.4 Datei »zahl_hoch.py«

Es wird die folgende Ausgabe erzeugt:

```
5 hoch 3 = 125
-5.2 hoch -3.8 = -0.0019018983172844654
Quadratwurzel aus 9 = 9 hoch 1/2 = 3.0
Kubikwurzel aus 27 = 27 hoch 1/3 = 3.0
```

Sowohl bei der Basis als auch beim Exponenten kann es sich um positive oder negative ganze Zahlen oder Zahlen mit Nachkommastellen handeln. Mithilfe des Exponentialoperators

lassen sich zum Beispiel auch Quadratwurzeln und Kubikwurzeln berechnen.

4.1.5 Rundung und Konvertierung

Die eingebaute Funktion `round()` dient zur Rundung einer Zahl. Im Unterschied dazu schneidet die bereits bekannte Funktion `int()` die Nachkommastellen einer Zahl ab. Einige Beispiele:

```
x = 12/7
print("x:", x)
print("Gerundet auf drei Stellen:", round(x, 3))
print("Gerundet auf null Stellen:", round(x))
print("int(x):", int(x))
print()

x = -12/7
print("x:", x)
print("Gerundet auf drei Stellen:", round(x, 3))
print("Gerundet auf null Stellen:", round(x))
print("int(x):", int(x))
```

Listing 4.5 Datei »zahl_runden.py«

Es wird folgende Ausgabe erzeugt:

```
x: 1.7142857142857142
Gerundet auf drei Stellen: 1.714
Gerundet auf null Stellen: 2
int(x): 1

x: -1.7142857142857142
Gerundet auf drei Stellen: -1.714
Gerundet auf null Stellen: -2
int(x): -1
```

Es werden die beiden Divisionen $12 / 7$ und $-12 / 7$ ausgeführt. Die beiden Ergebnisse werden jeweils auf drei verschiedene Arten umgewandelt:

1. Mithilfe der eingebauten Funktion `round()` wird das Ergebnis auf drei Stellen nach dem Komma gerundet.
2. Mit der gleichen Funktion wird das Ergebnis auf die nächsthöhere bzw. nächstniedrigere ganze Zahl gerundet.
3. Mithilfe der eingebauten Funktion `int()` wird das Ergebnis in eine ganze Zahl umgewandelt.

4.1.6 Winkelfunktionen

Im Modul `math` finden Sie unter anderem die Winkelfunktionen `sin()`, `cos()` und `tan()` und die inversen Winkelfunktionen `asin()`, `acos()` und `atan()`.

Ein Beispielprogramm:

```
import math

x = 30
xbm = math.radians(x)
print(f"Sinus {x} Grad: {math.sin(xbm)}")
print(f"Kosinus {x} Grad: {math.cos(xbm)}")
print(f"Tangens {x} Grad: {math.tan(xbm)}")

z = 0.5
print(f"Arkussinus {z} in Grad: {math.degrees(math.asin(z))}")
z = 0.866
print(f"Arkuskosinus {z} in Grad: {math.degrees(math.acos(z))}")
z = 0.577
print(f"Arkustangens {z} in Grad: {math.degrees(math.atan(z))}")
```

Listing 4.6 Datei »zahl_winkel.py«

Es wird die folgende Ausgabe erzeugt:

```
Sinus 30 Grad: 0.4999999999999994
Kosinus 30 Grad: 0.8660254037844387
Tangens 30 Grad: 0.5773502691896257
Arkussinus 0.5 in Grad: 30.000000000000004
Arkuskosinus 0.866 in Grad: 30.002910931188026
Arkustangens 0.577 in Grad: 29.984946007397852
```

Nach dem Import des Moduls `math` werden zunächst der Sinus, der Kosinus und der Tangens des Winkels 30 Grad berechnet. Alle Funktionen erwarten einen Winkel im Bogenmaß. Daher findet zuvor eine Umwandlung von Grad in Bogenmaß mithilfe der Funktion `radians()` statt.

Anschließend werden der Arkussinus, der Arkuskosinus und der Arkustangens von bestimmten Werten berechnet. Die Funktionen liefern einen Winkel im Bogenmaß. Dieser wird mithilfe der Funktion `degrees()` in Grad umgerechnet.

4.1.7 Weitere mathematische Funktionen

Ebenfalls im Modul `math` finden Sie einige Funktionen und Konstanten, die Ihnen teilweise von Ihrem Taschenrechner bekannt sind.

Ein Beispielprogramm:

```
import math

a = 4.75
print("Variable a:", a)
print("Quadratwurzel von a:", math.sqrt(a))
print("Natürlicher Logarithmus von a:", math.log(a))
print("e hoch a:", math.exp(a))
print("10er-Logarithmus von a:", math.log10(a))
print()

b = 34
print("Ganzzahlige Quadratwurzel:", math.isqrt(b))
print()

print("Kreiszahl pi:", math.pi)
print("Eulersche Zahl e:", math.e)
print()

t = 3, 2, -7
print("Produkt:", math.prod(t))
print("Fakultät von 5:", math.factorial(5))
print("Größter gem. Teiler von 60 und 135:", math.gcd(60, 135))
print("Rest:", math.remainder(10.8, 2.5))
print("Rest:", math.remainder(11.8, 2.5))
print()

for x in 2.96, 2.97:
    if math.isclose(3, x, rel_tol=0.01):
        print("Nahe dran")
    else:
        print("Nicht nahe dran")
```

Listing 4.7 Datei »zahl_rechner.py«

Es wird die folgende Ausgabe erzeugt:

```
Variable a: 4.75
Quadratwurzel von a: 2.179449471770337
Natürlicher Logarithmus von a: 1.55814461804655
e hoch a: 115.58428452718766
10er-Logarithmus von a: 0.6766936096248666

Ganzzahlige Quadratwurzel: 5

Kreiszahl pi: 3.141592653589793
Eulersche Zahl e: 2.718281828459045

Produkt: -42
Fakultät von 5: 120
Größter gem. Teiler von 60 und 135: 15
```

```
Rest: 0.8000000000000007
Rest: -0.6999999999999993
```

```
Nicht nahe dran
Nahe dran
```

Die Quadratwurzel einer positiven Zahl kann mithilfe der Funktion `sqrt()` berechnet werden, aber auch mithilfe des Exponentialoperators, siehe [Abschnitt 4.1.4](#), »Exponentialoperator `**`«. Es werden die Funktionen `log()` und `log10()` zur Berechnung der Logarithmen einer positiven Zahl zur Basis e und zur Basis 10 sowie die Funktion `exp()` zur Berechnung von e^x aufgerufen.

Seit Python 3.8 können Sie mithilfe der Funktion `isqrt()` die ganzzahlige Quadratwurzel einer Zahl berechnen. Das ist der größte ganzzahlige Wert, der kleiner ist als die mathematische Quadratwurzel einer Zahl.

Zudem gibt es die mathematischen Konstanten `pi` und `e`. Solche Konstanten stehen für unveränderbare Werte. Sie werden eingesetzt, weil man sich den Namen einer Konstanten meist besser merken kann als ihren Wert.

Seit Python 3.8 lässt sich mithilfe der Funktion `prod()` das Produkt der Elemente eines Iterables ermitteln, hier eines Tupels. Mehr zum Thema »Tupel« finden Sie in [Abschnitt 4.4](#), »Tupel«. Der Wert der Fakultät darf mathematisch und mithilfe der Funktion `factorial()` nur von positiven ganzen Zahlen berechnet werden.

Seit Python 3.5 gibt es die Funktion `gcd()`. Sie ermittelt den größten gemeinsamen Teiler (GGT, englisch: *greatest common divisor*) zweier ganzer Zahlen. Das ist die größte Zahl, durch die sich beide Zahlen ohne Rest teilen lassen.

Seit Python 3.7 lässt sich mithilfe der Funktion `remainder()` der Rest einer Division gemäß dem IEEE-754-Standard berechnen. Dabei handelt es sich um die Differenz zur nächsten ganzen Zahl. Was bedeutet das? Im vorliegenden Beispiel werden 10.8 bzw. 11.8

durch 2.5 geteilt. Das mathematische Ergebnis liegt jeweils zwischen den beiden ganzen Zahlen 4 ($4 \times 2.5 = 10$) und 5 ($5 \times 2.5 = 12.5$). Im Fall von 10.8 liegt der Wert 10 näher, daher liefert die Funktion `remainder()` den Wert 0.8 (= $10.8 - 10$). Im Fall von 11.8 liegt der Wert 12.5 näher, daher liefert die Funktion `remainder()` den Wert -0.7 (= $11.8 - 12.5$).

Seit Python 3.5 können Sie mithilfe der Funktion `isclose()` feststellen, ob zwei Zahlen einander nahe sind. In den beiden obigen Beispielen wird mithilfe der relativen Toleranz `0.01` festgestellt, ob die beiden Zahlen um maximal 1 % voneinander abweichen. Sie können einen von zwei benannten Parametern nutzen. Neben `rel_tol` gibt es auch `abs_tol` für die Messung mit einer absoluten Toleranz. Mehr zu benannten Parametern in [Abschnitt 5.6.2](#), »Benannte Parameter«.

4.1.8 Komplexe Zahlen

In diesem und den beiden nächsten Abschnitten, [Abschnitt 4.1.9](#) und [Abschnitt 4.1.10](#), werden spezielle Themen behandelt: komplexe Zahlen, Bitoperatoren und Brüche. Sie können zunächst übergangen und bei Bedarf nachgeschlagen werden.

Sie haben in Python die Möglichkeit, komplexe Zahlen zu speichern und mit ihnen zu rechnen. Die mathematischen Grundlagen zum Verständnis von komplexen Zahlen sind nicht Thema dieses Buchs. Informationen finden Sie zum Beispiel unter https://de.wikipedia.org/wiki/Komplexe_Zahl.

Ein Beispiel dazu:

```
a = 2.5 - 4.2j
print("a =", a)
print(f"Realteil: {a.real}, Imaginärteil: {a.imag}")
print("Betrag:", abs(a))
print("Konjugiert komplex:", a.conjugate())
print()

b = 3.7j
print("b =", b)
```

```

print(f"Realteil: {b.real}, Imaginärteil: {b.imag}")
print("Betrag:", abs(b))
print()
...

```

Listing 4.8 Datei »zahl_complex.py«, Teil 1 von 2

Es folgt die Ausgabe des ersten Programmteils:

```

a = (2.5-4.2j)
Realteil: 2.5, Imaginärteil: -4.2
Betrag: 4.887739763939975
Konjugiert komplex: (2.5+4.2j)

b = 3.7j
Realteil: 0.0 Imaginärteil: 3.7
Betrag: 3.7

```

Durch die Zuweisung einer komplexen Zahl wird `a` zu einem Objekt der Klasse `complex`. Ein solches Objekt kann Eigenschaften besitzen. Für ein solches Objekt können Methoden aufgerufen werden. Eine Methode ist eine Funktion, die nur für ein bestimmtes Objekt aufgerufen werden kann. Das trifft hier für die Methode `conjugate()` des Objekts `a` der Klasse `complex` zu. Mehr zu Klassen, Eigenschaften und Methoden erfahren Sie in [Kapitel 6, »Objektorientierte Programmierung«](#).

Eine komplexe Zahl setzt sich aus einem Real- und einem Imaginärteil zusammen. Der Imaginärteil wird durch das Zeichen »`j`« (oder auch »`J`«) gekennzeichnet. Wird nur ein Imaginärteil zugewiesen, wird der Realteil auf 0 gesetzt. Komplexe Zahlen werden in runden Klammern ausgegeben. Ausnahme: Es wurde nur ein Imaginärteil zugewiesen.

Die Eigenschaften `real` und `imag` stellen die jeweiligen Teile der komplexen Zahl zur Verfügung. Die Funktion `abs()` liefert ihren Betrag. Die Methode `conjugate()` liefert die konjugiert komplexe Zahl, also die komplexe Zahl mit geändertem Vorzeichen.

Der zweite Teil des Programms:

```

...
print("a + b =", a + b)
print("a - b =", a - b)
print("a * b =", a * b)
print("a / b =", a / b)
print("a ** 2.5 =", a ** 2.5)

```

```

print("5.1 + a / 3.2j * 2.8 =", 5.1 + a / 3.2j * 2.8)
print()

c = 2.5 - 4.2j
print("c =", c)
print("a == c:", a == c)
print("b != c:", b != c)
print()

c = 1j
print("c =", c)
print("c * c =", c * c)

```

Listing 4.9 Datei »zahl_complex.py«, Teil 2 von 2

Es folgt die Ausgabe des zweiten Programmteils:

```

a + b = (2.5-0.5j)
a - b = (2.5-7.9j)
a * b = (15.540000000000001+9.25j)
a / b = (-1.135135135135-0.6756756756756757j)
a ** 2.5 = (-44.83645966023058-27.915620445612213j)
5.1 + a / 3.2j * 2.8 = (1.4249999999999998-2.1875j)

c = (2.5-4.2j)
a == c: True
b != c: True

c = 1j
c * c = (-1+0j)

```

Sie können gemäß den zugehörigen mathematischen Regeln mithilfe der Operatoren `+`, `-`, `*`, `/` und `**` mit komplexen Zahlen rechnen. Es können auch gemischte Ausdrücke berechnet werden, die neben den komplexen Zahlen auch reelle Zahlen enthalten. Zum Vergleich von komplexen Zahlen können nur die beiden Operatoren `==` und `!=` genutzt werden. Das Quadrat der komplexen Zahl `j`, also `0 + 1j`, entspricht `-1`.

4.1.9 Bitoperatoren

Sämtliche Daten, ob nun Zahlen oder Zeichenketten, setzen sich auf der Hardwareebene aus Bits und Bytes zusammen. Auf dieser Ebene können Sie mit Dualzahlen (siehe auch [Abschnitt 4.1.1](#), »Ganze Zahlen«), der Funktion `bin()` und den sogenannten Bitoperatoren arbeiten. Ein Beispiel dazu:

```

# Nur 1 Bit gesetzt
bit0 = 1           # 0000 0001
bit3 = 8           # 0000 1000
print(bin(bit0), bin(bit3))

# Bitweises AND
a = 5             # 0000 0101
erg = a & bit0     # 0000 0001
if erg:
    print(a, "ist ungerade")

# Bitweises OR
erg = 0           # 0000 0000
erg = erg | bit0  # 0000 0001
erg = erg | bit3  # 0000 1001
print("Bits nacheinander gesetzt:", erg, bin(erg))

# Bitweises Exclusive-OR
a = 21            # 0001 0101
b = 19            # 0001 0011
erg = a ^ b       # 0000 0110
print("Ungleiche Bits:", erg, bin(erg))

# Bitweise Inversion, aus x wird -(x+1)
a = 11            # 0000 1011
erg = ~a          # 1111 0100
print("Bitweise Inversion:", erg, bin(erg))

# Bitweise schieben
a = 11            # 0000 1011
erg = a >> 1      # 0000 0101
print("Um 1 nach rechts geschoben:", erg, bin(erg))
erg = a << 2      # 0010 1100
print("Um 2 nach links geschoben:", erg, bin(erg))

```

Listing 4.10 Datei »operator_bit.py«

Es folgt die Ausgabe:

```

0b1 0b1000
5 ist ungerade
Bits nacheinander gesetzt: 9 0b1001
Ungleiche Bits: 6 0b110
Bitweise Inversion: -12 -0b1100
Um 1 nach rechts geschoben: 5 0b101
Um 2 nach links geschoben: 44 0b101100

```

Zunächst werden die beiden Variablen `bit0` und `bit3` eingeführt, die bei einigen der nachfolgenden Berechnungen benötigt werden. Sie haben die Werte 1 und 8. Am Ende der Programmzeile sehen Sie sie als Dualzahl, also mithilfe von 8 Bit (= 1 Byte) notiert. Das letzte Bit eines Bytes wird Bit 0 genannt, das vorletzte Bit ist Bit 1 usw. Die Werte der beiden Variablen `bit0` und `bit3` sind so

gewählt, dass jeweils nur ein Bit gesetzt ist (= 1), die restlichen Bits sind nicht gesetzt (= 0).

Sie können sich auch eine Reihe von acht Leuchtdioden vorstellen, die entweder *an* oder *aus* sind. Diese Information kann innerhalb eines Bytes gespeichert werden. Ist eines seiner Bits gesetzt, ist die betreffende LED *an*, ansonsten *aus*. Zur Verdeutlichung werden die beiden Variablen `bit0` und `bit3` mithilfe der Funktion `bin()` als Dualzahl ausgegeben.

Sie können den Bitoperator `&` zur bitweisen Und-Verknüpfung zweier Zahlen nutzen. Ähnlich wie beim logischen Operator `and` (siehe [Abschnitt 3.3.6](#), »Logische Operatoren«) wird ein bestimmtes Bit im Ergebnis nur gesetzt, wenn dieses Bit in beiden Zahlen gesetzt ist. Diese Operation wird für jedes einzelne Bit durchgeführt.

Möchten Sie wissen, ob ein bestimmtes Bit innerhalb einer Zahl gesetzt ist, verknüpfen Sie diese Zahl mithilfe des Bitoperators `&` mit einer anderen Zahl, in der nur dieses eine gesuchte Bit gesetzt ist. Handelt es sich um das Bit 0, wissen Sie darüber hinaus, ob die Zahl gerade ($\text{Bit } 0 = 0$) oder ungerade ($\text{Bit } 0 = 1$) ist.

Der Bitoperator `|` dient zur bitweisen Oder-Verknüpfung zweier Zahlen. Das Zeichen für diesen Operator erreichen Sie mithilfe der Taste `Alt`, die sich rechts neben dem Leerzeichen befindet. Ähnlich wie beim logischen Operator `or` (siehe ebenfalls [Abschnitt 3.3.6](#)) wird ein bestimmtes Bit im Ergebnis gesetzt, wenn dieses Bit in einer der beiden Zahlen oder in beiden Zahlen gesetzt ist. Diese Operation wird auch für jedes einzelne Bit durchgeführt.

Möchten Sie einzelne Bits einer Zahl setzen, verknüpfen Sie diese Zahl mithilfe des Bitoperators `|` mit einer anderen Zahl, in der nur dieses eine gesuchte Bit gesetzt ist.

Der Bitoperator `^` dient zur bitweisen Exklusiv-Oder-Verknüpfung zweier Zahlen. Ein bestimmtes Bit im Ergebnis wird gesetzt, wenn

dieses Bit nur in einer der beiden Zahlen gesetzt ist. Ist das Bit in beiden Zahlen gesetzt, wird das Ergebnis-Bit nicht gesetzt. Diese Operation wird ebenfalls für jedes einzelne Bit durchgeführt.

Sie können den Bitoperator `~` zur bitweisen Inversion einer Zahl nutzen. Dabei wird aus der Zahl x die Zahl $-x - 1$, aus 11 wird also -12.

Die beiden Bitoperatoren `>>` und `<<` dienen zum Schieben von Bits innerhalb einer Zahl:

- Mithilfe von `>>` werden alle Bits um eine bestimmte Anzahl von Stellen nach rechts geschoben. Die Bits, die dabei nach rechts »hinausfallen«, sind verloren. Eine Verschiebung um n Bit nach rechts entspricht einer ganzzahligen Division durch 2^n . Eine Verschiebung um 1 Bit nach rechts entspricht also einer ganzzahligen Division durch 2.
- Mithilfe von `<<` werden alle Bits um eine bestimmte Anzahl von Stellen nach links geschoben. Eine Verschiebung um n Bit nach links entspricht einer Multiplikation mit 2^n . Eine Verschiebung um 1 Bit nach links entspricht also einer Multiplikation mit 2.

4.1.10 Brüche

Python kann auch mit Brüchen rechnen bzw. Informationen über Brüche zur Verfügung stellen. Dazu wird das Modul `fractions` (deutsch: Brüche) genutzt. Ein Beispiel:

```
import fractions

z = 12
n = 28
print(f"Bruch: {z}/{n}")

b1 = fractions.Fraction(z, n)
print("Fraction-Objekt:", b1)
print(f"Zähler: {b1.numerator}, Nenner: {b1.denominator}")
print("Wert:", b1.numerator / b1.denominator)
print()
```

```
x = 2.375
print("Zahl:", x)
b2 = fractions.Fraction(x)
print("Fraction-Objekt:", b2)
```

Listing 4.11 Datei »zahl_bruch.py«

Die Ausgabe lautet:

```
Bruch: 12/28
Fraction-Objekt: 3/7
Zähler: 3, Nenner: 7
Wert: 0.42857142857142855
```

```
Zahl: 2.375
Fraction-Objekt: 19/8
```

Zunächst wird ein Beispielbruch in der bekannten Form dargestellt. Er wird aus zwei ganzen Zahlen gebildet, dem Zähler und dem Nenner.

Die Funktion `Fraction()` aus dem Modul `fractions` bietet verschiedene Möglichkeiten, einen Bruch zu erzeugen. Genauer gesagt, handelt es sich bei `Fraction()` um den Konstruktor der Klasse `Fraction`. Damit wird eine Instanz (ein Objekt) der Klasse erzeugt und eine Referenz auf dieses Objekt zurückgeliefert. Klassen, Instanzen, Konstruktoren und andere Begriffe aus der objektorientierten Programmierung werden in [Kapitel 6, »Objektorientierte Programmierung«](#), genauer erläutert.

Das `Fraction`-Objekt, also der Bruch `b1`, der aus 12 und 28 gebildet wird, wird bei der Erzeugung automatisch auf 3/7 gekürzt.

Zähler und Nenner des Bruchs stehen in den Eigenschaften `numerator` und `denominator` einzeln zur Verfügung. Der Wert eines Bruchs lässt sich darüber berechnen: $3/7 = 0,428\dots$

Umgekehrt können Sie auch eine Zahl mit Nachkommastellen in einen Bruch umrechnen. Dazu übergeben Sie die Zahl der Konstruktormethode `Fraction()`: Aus 2.375 wird 19/8.

Im nachfolgenden Programm wird ein Bruch dazu genutzt, eine Zahl mit Nachkommastellen zu approximieren, also anzunähern. Dazu dient die Methode `limit_denominator()`. Das Programm:

```

import fractions

x = 1.84953
print("Zahl:", x)

b3 = fractions.Fraction(x)
print("Fraction-Objekt:", b3)

b4 = b3.limit_denominator(100)
print("Approximiert auf Nenner max. 100:", b4)

wert = b4.numerator / b4.denominator
print("Wert:", wert)
print("rel. Fehler:", abs((x-wert)/x))

```

Listing 4.12 Datei »zahl_bruch_naeahern.py«

Die Ausgabe:

```

Zahl: 1.84953
Fraction-Objekt: 8329542618810553/4503599627370496
Approximiert auf Nenner max. 100: 172/93
Wert: 1.8494623655913978
rel. Fehler: 3.656843014286614e-05

```

Es wird die Zahl 1,84953 untersucht. Sie entspricht dem Bruch 184953/100000. Mithilfe der Methode `limit_denominator()` wird der Nenner auf die Zahl 100 begrenzt. Dann wird der Bruch gesucht, der

- einen Nenner mit dem maximalen Wert 100 hat und
- der Zahl 1,84953 am nächsten kommt.

Im vorliegenden Fall ist das der Bruch 172/93. Er hat den Wert 1,8494623655913978 und kommt der ursprünglichen Zahl recht nahe. Der relative Fehler zwischen diesem Wert und der untersuchten Zahl beträgt nur $3,65684301429 \times 10^{-5}$.

Er wird mithilfe der eingebauten Funktion `abs()` zur Berechnung des Betrags ermittelt. Beim Betrag handelt es sich um den Absolutwert einer Zahl, also der Zahl ohne das Vorzeichen.

Sollten Sie die Methode `gcd()` zur Ermittlung des größten gemeinsamen Teilers vermissen: Seit Python 3.5 gehört sie als Funktion zum Modul `math` (siehe [Abschnitt 4.1.7](#), »Weitere

mathematische Funktionen«) und wird im Modul `fractions` als veraltet bezeichnet.

4.2 Zeichenketten

Zeichenketten sind Sequenzen von einzelnen Zeichen. Auch andere Datentypen gehören zu den Sequenzen. Anhand von Zeichenketten folgt eine Einführung in die Sequenzen.

4.2.1 Eigenschaften

Zeichenketten (Strings) sind Objekte des Datentyps `str`. Sie werden in einfache, doppelte oder dreimal doppelte Hochkommata gesetzt. Mithilfe einer `for`-Schleife können Sie auf die Elemente einer Zeichenkette zugreifen.

Ein Beispiel:

```
tx = "Das"
print("Text:", tx)
print("Typ:", type(tx))
print("Anzahl der Zeichen:", len(tx))
for z in tx:
    print(z)
for i in range(len(tx)):
    print(f"{i}: {tx[ i ]}")

tx = ' Auch das ist eine Zeichenkette'
print(tx)

tx = """Diese Zeichenkette
        steht in
        mehreren Zeilen"""
print(tx)

tx = ' Hier sind "doppelte Hochkommata" gespeichert'
print(tx)
```

Listing 4.13 Datei »text_eigenschaft.py«

Es wird die folgende Ausgabe erzeugt:

```
Text: Das
Typ: <class 'str'>
Anzahl der Zeichen: 3
D
a
s
0: D
1: a
```

```

2: s
Auch das ist eine Zeichenkette
Diese Zeichenkette
    steht in
    mehreren Zeilen
Hier sind "doppelte Hochkommata" gespeichert

```

Für die erste Zeichenkette wird mithilfe der Funktion `type()` der Datentyp und mithilfe der Funktion `len()` die Anzahl der Elemente ausgegeben, also die Länge der Zeichenkette.

Eine Zeichenkette ist ein Iterable, das aus einzelnen Elementen besteht. Mithilfe einer `for`-Schleife kann auf die Elemente zugegriffen werden. Soll zusätzlich der Index, also die laufende Nummer jedes Elements angegeben werden, benötigen Sie eine Schleifenvariable, die die einzelnen Indizes durchläuft.

Die zweite Zeichenkette wird in einfachen Hochkommata angegeben. Die dritte Zeichenkette wird in dreifachen doppelten Hochkommata angegeben, erstreckt sich über mehrere Zeilen und wird auch so ausgegeben.

Die letzte Zeichenkette verdeutlicht den Vorteil, den das Vorhandensein mehrerer Alternativen bietet: Die doppelten Hochkommata sind hier Bestandteil des Texts und werden auch ausgegeben.

4.2.2 Operatoren

Der Operator `+` dient zur Verkettung mehrerer Sequenzen, der Operator `*` zur Vervielfachung einer Sequenz. Mithilfe des Operators `in` stellen Sie fest, ob ein bestimmtes Element in einer Sequenz enthalten ist. Betrachten Sie das folgende Beispiel für diese Operatoren, angewendet für Strings:

```

kreis = "-oooo-"
stern = "***"
linie = stern + kreis * 3 + stern
print(linie)

tx = "Hallo"
print("Text:", tx)
if "a" in tx:

```

```

        print("a ist enthalten")
if "z" not in tx:
    print("z ist nicht enthalten")

```

Listing 4.14 Datei »text_operator.py«

Die Ausgabe lautet:

```

***-oooo--oooo--oooo-***
Text: Hallo
a ist enthalten
z ist nicht enthalten

```

Die Zeichenkette `linie` wird mithilfe des Verkettungsoperators `+` und des Vervielfachungsoperators `*` zusammengesetzt. Der Text `"-oooo-` wird dabei dreimal hintereinander in `tlinie` gespeichert.

Mithilfe des Operators `in` wird festgestellt, ob das Element `a` in der Sequenz enthalten ist. Der logische Operator `not` dient (zusammen mit `in`) der Feststellung, ob das Element `z` nicht enthalten ist.

4.2.3 Slices

Bereiche von Sequenzen werden als *Slices* bezeichnet. Der Einsatz von Slices wird am Beispiel eines Strings verdeutlicht. Auf die gleiche Art und Weise sind Slices auch auf andere Sequenzen anwendbar.

Als Beispiel für eine Sequenz wird die Zeichenkette `Hallo` gespeichert. [Tabelle 4.1](#) stellt die einzelnen Elemente mit dem zugehörigen Index dar. Der Index beginnt bei 0; alternativ können Sie auch einen negativen Index nutzen, der mit -1 endet (siehe unterste Zeile der Tabelle).

Index	0	1	2	3	4
Element	H	a	l	l	o
Negativer Index	-5	-4	-3	-2	-1

Tabelle 4.1 Sequenz mit Index

Ein Slice wird durch die Angabe eines Bereichs in rechteckigen Klammern ([]) erzeugt. Diese erreichen Sie mithilfe der Taste **Alt**, die sich rechts neben dem Leerzeichen befindet.

Ein Slice beginnt mit einem Start-Index, gefolgt von einem Doppelpunkt und einem End-Index. Ein Slice, der nur aus einem einzelnen Element besteht, wird durch die Angabe eines einzelnen Index erzeugt.

```
tx = "Hallo"
print("Text:", tx)

print("[ 1:4] :", tx[ 1:4] )
print("[ :4] :", tx[ :4] )
print("[ 2:] :", tx[ 2:] )
print("[ :] :", tx[ :] )
print("[ 1] :", tx[ 1] )
print("[ 1:-2] :", tx[ 1:-2] )
```

Listing 4.15 Datei »text_slice.py«

Es wird die folgende Ausgabe erzeugt:

```
Text: Hallo
[1:4]: all
[:4]: Hall
[2:] : llo
[:] : Hallo
[1] : a
[1:-2] : al
```

Die Erläuterung der verschiedenen Slices:

- Slice `[1:4] :`: Der Bereich erstreckt sich von dem Element, das durch den Start-Index gekennzeichnet wird, bis vor das Element, das durch den End-Index gekennzeichnet wird.
- Slice `[:4] :`: Wird der Start-Index weggelassen, beginnt der Bereich bei 0.
- Slice `[2:] :`: Wird der End-Index weggelassen, endet der Bereich am Ende der Sequenz.
- Slice `[:] :`: Werden beide Indizes weggelassen, erstreckt sich der Bereich über die gesamte Sequenz. Eine solche Angabe wird zur Zerlegung von mehrdimensionalen Sequenzen benötigt.

- Slice [1] : Wird nur ein Index angegeben, besteht der Bereich nur aus einem einzelnen Element.
- Slice [1:-2] : Wird ein Index mit einer negativen Zahl angegeben, wird für diesen Index vom Ende aus gemessen, beginnend bei -1.

Ein Slice einer Sequenz hat denselben Datentyp wie die Sequenz:
Ein Slice einer Zeichenkette ist eine Zeichenkette, ein Slice einer Liste ist eine Liste.

4.2.4 Änderbarkeit

Anhand des nachfolgenden Beispiels sehen Sie, dass eine Zeichenkette nicht veränderbar ist. Es können keine Zeichen oder Slices durch andere Zeichen oder Slices ersetzt werden:

```
tx = "Das ist ein Text"
print(tx)

try:
    tx[ 4:6] = "war"
except:
    print("Fehler")

tx = tx[ :4] + "war" + tx[ 7:]
print(tx)
```

Listing 4.16 Datei »text_aenderbar.py«

Die Ausgabe lautet:

```
Das ist ein Text
Fehler
Das war ein Text
```

Die einzige Möglichkeit zur Änderung einer Variablen, die eine Zeichenkette enthält, ist die Zuweisung einer neuen Zeichenkette in derselben Variablen. Diese kann aus Teilen der alten Zeichenkette zusammengesetzt werden.

4.2.5 Suchen und Ersetzen

Anhand des folgenden Beispiels werden einige Methoden zum Suchen und Ersetzen in Texten verdeutlicht:

```
tx = "Das ist ein Beispielsatz"
print("Text:", tx)

such = "ei"
print("Suchtext:", such)
print()

anz = tx.count(such)
print(f"count: Der String {such} kommt {anz} mal vor")

pos = tx.find(such)
while pos != -1:
    print("An Position", pos)
    pos = tx.find(such, pos+1)

pos = tx.rfind(such)
if pos != -1:
    print("rfind: Zum letzten Mal an Position", pos)
print()

if tx.startswith("Das"):
    print("Text beginnt mit Das")
if not tx.endswith("Das"):
    print("Text endet nicht mit Das")

tx = tx.replace("ist", "war")
print("Nach Ersetzen:", tx)

z = 48.2
tx = str(z)
tx = tx.replace(".", ",")
print("Zahl mit Komma:", tx)
```

Listing 4.17 Datei »text_suchen.py«

Folgende Ausgabe wird erzeugt:

```
Text: Das ist ein Beispielsatz
Suchtext: ei

count: Der String ei kommt 2 mal vor
An Position 8
An Position 13
rfind: Zum letzten Mal an Position 13

Text beginnt mit Das
Text endet nicht mit Das
Nach Ersetzen: Das war ein Beispielsatz
Zahl mit Komma: 48,2
```

Die Methode `count()` ergibt die Anzahl der Vorkommen eines Suchtextes innerhalb des analysierten Texts.

Die Methode `find()` liefert die Position, an der ein Suchtext innerhalb eines analysierten Texts vorkommt. Kommt der gesuchte Text nicht vor, wird `-1` geliefert. Sie können optional einen zweiten Parameter angeben, der die Position bestimmt, ab der gesucht werden soll. Das können Sie nutzen, um mithilfe einer Schleife alle Vorkommen eines Suchtextes zu finden.

Die Methode `rfind()` ergibt die Position des letzten Vorkommens des Suchtextes innerhalb des analysierten Texts.

Mithilfe der Methoden `startswith()` und `endswith()` untersuchen Sie, ob eine Zeichenkette mit einem bestimmten Text beginnt oder endet. Beide Methoden liefern einen Wahrheitswert, daher kann der Rückgabewert zur Steuerung der Verzweigung genutzt werden.

Die Methode `replace()` ersetzt einen gesuchten Teilstext durch einen anderen und liefert den geänderten Text zurück.

Die eingebaute Funktion `str()` erstellt eine Zeichenkette aus einem Objekt. Das können Sie zum Beispiel nutzen, um eine Zahl mit einem Dezimalkomma auszugeben, indem Sie die Zahl zunächst in eine Zeichenkette umwandeln und anschließend den Dezimalpunkt mithilfe der Methode `replace()` durch ein Komma ersetzen.

4.2.6 Leerzeichen entfernen

Nach dem Einlesen von Texten aus einer Datei können sich Leerzeichen, Tabulatorzeichen (`\t`) und Zeilenende-Zeichen (`\n`) am Anfang oder am Ende eines Texts befinden. Sie können mithilfe der Methode `strip()` entfernt werden. Ein Beispiel:

```
tx = "    \tHallo Welt\n\t    "
print(f"!{tx}!")
print(f"!{tx.strip()}!")
```

Listing 4.18 Datei »text_leerzeichen.py«

Zur Verdeutlichung wird zusätzlich das Zeichen | vor und nach dem Text ausgegeben. Die Ausgabe des Programms sehen Sie in Abbildung 4.1.

```
|      Hallo Welt  
|  
|Hallo Welt|
```

Abbildung 4.1 Löschen von Zeichen am Anfang und am Ende des Textes

4.2.7 Text zerlegen

Die Methode `split()` dient zum Zerlegen von Texten in Teiltexte. Das nachfolgende Beispiel zeigt zwei Anwendungen:

```
tx = "Das ist ein Satz"  
print("Text:", tx)  
wortliste = tx.split()  
for i in range(0, len(wortliste)):  
    print("Element:", i, wortliste[ i ] )  
print()  
  
tx = "Maier; Hans; 6714; 3.500,00; 15.03.62"  
print("Text:", tx)  
wortliste = tx.split("; ")  
for i in range(0, len(wortliste)):  
    print("Element:", i, wortliste[ i ] )
```

Listing 4.19 Datei »text_zerlegen.py«

Die Ausgabe lautet:

```
Text: Das ist ein Satz  
Element: 0 Das  
Element: 1 ist  
Element: 2 ein  
Element: 3 Satz  
  
Text: Maier; Hans; 6714; 3.500,00; 15.03.62  
Element: 0 Maier  
Element: 1 Hans  
Element: 2 6714  
Element: 3 3.500,00  
Element: 4 15.03.62
```

Die Methode `split()` zerlegt einen Text in einzelne Teile, die in einer Liste gespeichert werden. Standardmäßig wird das Leerzeichen als Trennzeichen angesehen. Die eingebaute Funktion `len()` liefert auch für eine Liste die Anzahl der

Elemente. Mehr Informationen zu Listen gibt es in [Abschnitt 4.3, »Listen«](#).

Die einzelnen Teile eines Datensatzes werden häufig mit einem Semikolon voneinander getrennt. Sie können es als Parameter der Methode `split()` verwenden, um den Datensatz aufzuteilen.

4.2.8 Konstanten

Das Modul `string` stellt einige nützliche Zeichenketten-Konstanten bereit, zum Beispiel alle Buchstaben, alle Ziffern oder alle Interpunktionszeichen, wie Sie in folgendem Programm sehen:

```
import string
print("Klein:", string.ascii_lowercase)
print("Groß:", string.ascii_uppercase)
print("Buchstaben:", string.ascii_letters)
print("Ziffern:", string.digits)
print("Interpunktionszeichen:", string.punctuation)
```

Listing 4.20 Datei »text_konstanten.py«

Diese Konstanten können zum Beispiel für eine zufällige Auswahl von Zeichen für ein Passwort dienen, siehe [Abschnitt 5.4, »Verschlüsselung«](#).

Die Ausgabe sieht wie folgt aus:

```
Klein: abcdefghijklmnopqrstuvwxyz
Groß: ABCDEFGHIJKLMNOPQRSTUVWXYZ
Buchstaben: abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
Ziffern: 0123456789
Interpunktionszeichen: !#$%&'()*+,-./:;=>?@[\]^_`{|}~
```

4.2.9 Datentyp »bytes«

Beim Datentyp »bytes« handelt es sich um ein spezielles Thema. Es kann zunächst übergangen und bei Bedarf nachgeschlagen werden.

Die bisher behandelten Zeichenketten, also Objekte des Datentyps `str`, werden aus dem umfangreichen Zeichensatz der

Unicode-Zeichen gebildet.

Objekte des seltener genutzten Datentyps `bytes` entsprechen Zeichen, deren Zeichencode nur im Bereich von 0 bis 255 liegt. Jedes Zeichen kann mithilfe eines Bytes gespeichert werden. Sie können `bytes`-Objekte mithilfe von Byte-Literalen erzeugen oder mithilfe der eingebauten Funktion `bytes()`.

Byte-Literale beginnen mit einem »b« oder einem »B«. Dies ist bei der Eingabe oder Zuweisung zu beachten. Bei der Ausgabe wird ein `b` vorangestellt. Es folgt ein Beispielprogramm:

```
st = "Hallo"
print(st, type(st))

by = b' Hallo'
print(by, type(by))

by = bytes("Hallo", "UTF-8")
print(by, type(by))

by = b' Hallo'
st = by.decode()
print(st, type(st))
```

Listing 4.21 Datei »bytes.py«

Das Programm erzeugt die folgende Ausgabe:

```
Hallo <class 'str'>
b' Hallo' <class 'bytes'>
b' Hallo' <class 'bytes'>
Hallo <class 'str'>
```

Zunächst werden ein `str`-Objekt und ein `bytes`-Objekt per Zuweisung erzeugt und ausgegeben. Beachten Sie beim Byte-Literal das vorangestellte `b`. Zur Umwandlung eines `str`-Objekts in ein `bytes`-Objekt wird die eingebaute Funktion `bytes()` genutzt. Dabei wird die Codierung des `str`-Objekts angegeben, hier UTF-8. Zur Umwandlung eines `bytes`-Objekts in ein `str`-Objekt wird die Methode `decode()` verwendet.

4.3 Listen

Eine Liste ist eine Sequenz von Objekten in ihrer allgemeinsten Form. Sie kann Elemente unterschiedlicher Datentypen enthalten. Eine Liste bietet vielfältige Möglichkeiten, unter anderem die Funktionalität von ein- und mehrdimensionalen Feldern (Arrays), wie man sie aus anderen Programmiersprachen kennt.

4.3.1 Eigenschaften und Operatoren

Eine Liste ist im Gegensatz zu einem String veränderbar. Davon abgesehen ist ein String, vereinfacht gesagt, nur eine Liste zur Speicherung von einzelnen Zeichen. Einige Beispiele für Listen:

```
import random

z = [ 3, 6.2, -12]
print("Liste:", z)
print("Element:", z[ 0] )
print("Slice", z[ :2] )
print()

print("Schleife:")
for element in z:
    print(element)
print()

print("Schleife mit Index:")
for i in range(len(z)):
    print(f"{i}: {z[ i]}")
print()

a = [ "Paris", "Lyon"]
b = [ "Rom", "Pisa"]
c = a + b * 2
print("Addiert und vervielfacht:", c)

print("Zufälliges Element:", random.choice(c))
random.shuffle(c)
print("Nach dem Mischen:", c)
```

Listing 4.22 Datei »liste_eigenschaft.py«

Eine mögliche Ausgabe:

```

Liste: [3, 6.2, -12]
Element: 3
Slice [3, 6.2]

Schleife:
3
6.2
-12

Schleife mit Index:
0: 3
1: 6.2
2: -12

Addiert und vervielfacht: ['Paris', 'Lyon', 'Rom', 'Pisa', 'Rom', 'Pisa']
Zufälliges Element: Rom
Nach dem Mischen: ['Pisa', 'Pisa', 'Rom', 'Paris', 'Rom', 'Lyon']

```

Eine Liste wird innerhalb von rechteckigen Klammern angegeben. Darin werden die einzelnen Elemente durch Kommata getrennt notiert.

Die Variable `z` enthält eine Liste von Zahlen. Wie bei Strings ermitteln Sie ein einzelnes Element einer Liste durch Angabe eines Index. Einen Bereich erhalten Sie mithilfe eines Slice.

Mithilfe einer `for`-Schleife können alle Elemente einer Liste ausgegeben werden, mit oder ohne Index. Die Länge einer Sequenz, also auch einer Liste, ermitteln Sie mit der Funktion `len()`.

Listen können mithilfe des Operators `+` verkettet und mithilfe des Operators `*` vervielfacht werden.

Die Funktion `choice()` aus dem Modul `random` liefert ein zufälliges Element aus der Liste. Die Funktion `shuffle()` aus dem Modul `random` mischt die Elemente der Liste mithilfe eines Zufallsgenerators.

4.3.2 Mehrdimensionale Listen

Eine Liste kann Elemente unterschiedlicher Datentypen enthalten. Diese Elemente können wiederum Listen sein. Das kann man zur Erstellung einer mehrdimensionalen Liste nutzen:

```

z = [[ "Paris", "Fr", 3.5], [ "Rom", "It", 4.2], [ "Madrid", "Sp", 3.2]]
print("Liste:", z)
print("Länge:", len(z))
print()

print("Unter-Liste:", z[ 0])
print("Länge:", len(z[ 0]))
print("Slice von Unter-Listen:", z[ :2])
print()

print("Element:", z[ 2][ 0])
print("Länge:", len(z[ 2][ 0]))
print("Slice von Elementen:", z[ 2][ 0][ :3])
print()

for i in range(len(z)):
    print(f"{i}: {z[ i][ 0]} hat {z[ i][ 2]} Mio. Einwohner")
for stadt in z:
    print(f"{stadt[ 0]} hat {stadt[ 2]} Mio. Einwohner")

```

Listing 4.23 Datei »liste_mehrdimensional.py«

Die Ausgabe lautet:

```

Liste: [['Paris', 'Fr', 3.5], ['Rom', 'It', 4.2], ['Madrid', 'Sp', 3.2]]
Länge: 3

Unter-Liste: ['Paris', 'Fr', 3.5]
Länge: 3
Slice von Unter-Listen: [['Paris', 'Fr', 3.5], ['Rom', 'It', 4.2]]

Element: Madrid
Länge: 6
Slice von Elementen: Mad

0: Paris hat 3.5 Mio. Einwohner
1: Rom hat 4.2 Mio. Einwohner
2: Madrid hat 3.2 Mio. Einwohner
Paris hat 3.5 Mio. Einwohner
Rom hat 4.2 Mio. Einwohner
Madrid hat 3.2 Mio. Einwohner

```

Die mehrdimensionale Liste `z` besitzt drei Elemente, und zwar drei Unter-Listen. Diese bestehen jeweils aus zwei Zeichenketten und einer Zahl. Auf eine oder mehrere Unter-Listen greifen Sie mithilfe eines Index oder Slice zu.

Die Länge einer Unter-Liste wird mithilfe der Funktion `len()`, bezogen auf die Unter-Liste, ermittelt. Einzelne Elemente einer Unter-Liste erreichen Sie durch Angabe mehrerer Indizes. Der erste Index steht für die Unter-Liste, der zweite für das Element innerhalb der Unter-Liste.

Handelt es sich bei einem Element wiederum um eine Sequenz, können Sie:

- mithilfe eines Index oder eines Slice auf einzelne Unter-Elemente der Sequenz zugreifen
- die Länge der Sequenz mithilfe der Funktion `len()` ermitteln

Mithilfe von `for`-Schleifen können Sie die mehrdimensionale Liste durchlaufen.

4.3.3 Änderbarkeit

Listen können, im Gegensatz zu Strings, verändert werden. Sie können also ein Element oder einen Slice hinzufügen, ändern oder löschen. Ersetzen Sie ein einzelnes Element durch einen Slice, wird eine neue Unter-Liste erzeugt. Es folgt ein Beispiel:

```
z = [ "Paris", "Lyon", "Reims", "Nice"]
print("Liste:", z)

z[ 2] = "Lens"
print("Element ersetzt:", z)

z[ 1:3] = [ "Nancy", "Metz", "Gap"]
print("Slice durch Liste ersetzt:", z)

del z[ 2:]
print("Slice entfernt:", z)

z[ 0] = [ "Paris-Nord", "Paris-Sud"]
print("Element durch Liste ersetzt:", z)
```

Listing 4.24 Datei »liste_aendern.py«

Die Ausgabe der Liste in den verschiedenen Zuständen:

```
Liste: ['Paris', 'Lyon', 'Reims', 'Nice']
Element ersetzt: ['Paris', 'Lyon', 'Lens', 'Nice']
Slice durch Liste ersetzt: ['Paris', 'Nancy', 'Metz', 'Gap', 'Nice']
Slice entfernt: ['Paris', 'Nancy']
Element durch Liste ersetzt: [['Paris-Nord', 'Paris-Sud'], 'Nancy']
```

Die Liste besteht zunächst aus vier Zeichenketten und wird wie folgt geändert:

- Ein einzelnes Element wird durch eine andere Zeichenkette ersetzt.
- Ein Slice wird durch eine Liste ersetzt, die eine andere Länge besitzt.
- Ein Slice wird mithilfe des Schlüsselworts `del` aus der Liste entfernt.
- Ein einzelnes Element wird durch eine Liste ersetzt; dadurch wird eine neue Unter-Liste erzeugt.

4.3.4 Methoden

Im nachfolgenden Programm wird eine Reihe von weiteren Methoden zum Ändern und Untersuchen von Listen verdeutlicht:

```

z = [ "Paris", "Lyon", "Metz"]
print("Liste:", z)

z.insert(1, "Nantes")
print("Einsetzen:", z)

z.sort()
print("Sortieren:", z)

z.reverse()
print("Umdrehen:", z)

such = "Nantes"
if such in z:
    z.remove(such)
    print("Entfernen:", z)

z.append("Paris")
print("Am Ende hinzufügen:", z)
print()

such = "Paris"
print("Anzahl gefunden:", z.count(such))

startpos = 0
while such in z[startpos:]:
    pos = z.index("Paris", startpos)
    print("Position:", pos)
    startpos = pos + 1

```

Listing 4.25 Datei »liste_methoden.py«

Die Ausgabe der Liste in den verschiedenen Zuständen:

```
Liste: ['Paris', 'Lyon', 'Metz']
Einsetzen: ['Paris', 'Nantes', 'Lyon', 'Metz']
Sortieren: ['Lyon', 'Metz', 'Nantes', 'Paris']
Umdrehen: ['Paris', 'Nantes', 'Metz', 'Lyon']
Entfernen: ['Paris', 'Metz', 'Lyon']
Am Ende hinzufügen: ['Paris', 'Metz', 'Lyon', 'Paris']

Anzahl gefunden: 2
Position: 0
Position: 3
```

Die Originalliste, bestehend aus drei Zeichenketten, wird erstellt. Danach wird ein neues Element mit der Methode `insert()` an Position 1 eingefügt.

Die Liste wird mit der Methode `sort()` sortiert. Handelt es sich um eine Liste von Zeichenketten, wird alphabetisch sortiert. Eine Liste von Zahlen wird nach Größe sortiert. Bei anderen Typen von Elementen oder bei gemischten Listen ist der Einsatz der Methode `sort()` nur bedingt sinnvoll.

Die Liste wird mithilfe der Methode `reverse()` intern umgedreht.

Ein bestimmtes Element wird in der Liste gesucht. Ist es mindestens einmal vorhanden, wird das erste Vorkommen mit `remove()` gelöscht. Wäre es nicht vorhanden, würde eine Ausnahme ausgelöst werden.

Ein Element wird am Ende der Liste mithilfe der Methode `append()` angefügt.

Die Anzahl der Vorkommen eines bestimmten Elements wird mithilfe der Methode `count()` ermittelt.

Die Position des Vorkommens eines bestimmten Elements kann mithilfe der Methode `index()` bestimmt werden. Mithilfe eines zweiten Parameters kann festgelegt werden, ab welcher Position die Suche beginnen soll. Kommt das Element nicht vor, wird eine Ausnahme ausgelöst.

Zur Bestimmung aller Positionen eines bestimmten Elements wird eine `while`-Schleife genutzt. Ihre Bedingung lautet: »Solange

das Element in der (restlichen) Liste vorkommt«. Beim ersten Mal wird mit dieser Bedingung die gesamte Liste geprüft. Danach wird nur noch derjenige Teil der Liste geprüft, der nach der letzten gefundenen Position liegt.

4.3.5 List Comprehension

Die Technik der *List Comprehension* ermöglicht die schnelle Erzeugung einer Liste aus einer anderen Liste. Dabei können Sie die Elemente der ersten Liste filtern und verändern. Es folgen drei Beispiele:

```
x = [ 3, -6, -8, 9, 15]
print(x)
y = [ 2, 13, 4, -8, 4]
print(y)
print()

a = [ z+1 for z in x]
print(a)

b = [ z+1 for z in x if z>0]
print(b)

c = [ x[ i]+y[ i] for i in range(len(x))]
print(c)

d = [ x[ i]+y[ i] for i in range(len(x)) if x[ i]>0 and y[ i]>0]
print(d)
```

Listing 4.26 Datei »liste_comprehension.py«

Die Ausgabe lautet:

```
[3, -6, -8, 9, 15]
[2, 13, 4, -8, 4]

[4, -5, -7, 10, 16]
[4, 10, 16]
[5, 7, -4, 1, 19]
[5, 19]
```

Zunächst werden die beiden Beispiellisten `x` und `y` erstellt und ausgegeben.

Die Liste `a` entspricht der Liste `x`. Der Wert jedes Elements wird allerdings um 1 erhöht. Der Ausdruck `z+1 for z in x` bedeutet: »Übernehme die Elemente der Liste und addiere 1«.

Bei der Liste `b` verhält es sich ähnlich. Allerdings werden nur die Elemente mit positiven Werten herausgefiltert. Der Ausdruck `z+1 for z in x if z>0` bedeutet: »Übernehme die Elemente der Liste und addiere 1, aber nur falls sie größer als 0 sind«.

Für die Liste `c` wird auch der Index einbezogen. Die Elemente der beiden Listen werden paarweise addiert. Der Ausdruck `x[i] +y[i] for i in range(len(x))` bedeutet: »Addiere die i-ten Elemente der beiden Listen über die Länge der Liste `x`«. Die Liste `y` könnte also auch größer sein.

Bei der Liste `d` verhält es sich wiederum ähnlich. Allerdings werden nur die Paare mit zwei positiven Elementen herausgefiltert. Der Ausdruck `x[i] +y[i] for i in range(len(x)) if x[i]>0 and y[i]>0` bedeutet: »Addiere die i-ten Elemente der beiden Listen über die Länge der Liste `x`, aber nur falls beide größer als 0 sind«.

4.4 Tupel

Ein Tupel entspricht einer Liste, deren Elemente nicht verändert werden dürfen. Ansonsten gelten die gleichen Regeln, und es können die gleichen Operationen und Methoden auf Tupel wie auf Listen angewendet werden, sofern sie keine Veränderung des Tupels hervorrufen.

Nachfolgend werden einige Besonderheiten von Tupeln verdeutlicht:

```
z = (3, 6, -8)
print("Tupel 1 verpackt:", z)
z = 3, 6, -8
print("Tupel 2 verpackt:", z)

for i in 3, 6, -8:
    print(i)

a, b, c = z
print("Tupel entpackt:", a, b, c)

a, b, c = 3, 6, -8
print("Mehrfache Zuweisung:", a, b, c)

a, b, c = c, a, a+b
print("Auswirkung später:", a, b, c)

a, *b, c = 3, 6, 12, -28, -8
print("Rest in Liste:", a, b, c)
```

Listing 4.27 Datei »tupel.py«

Die Ausgabe des Programms:

```
Tupel 1 verpackt: (3, 6, -8)
Tupel 2 verpackt: (3, 6, -8)
3
6
-8
Tupel entpackt: 3 6 -8
Mehrfache Zuweisung: 3 6 -8
Auswirkung später: -8 3 9
Rest in Liste: 3 [6, 12, -28] -8
```

Ein Tupel enthält mehrere Elemente, die durch Kommata voneinander getrennt sind. Sie können innerhalb von runden

Klammern notiert werden. Die Zuweisung eines Tupels zu einer einzelnen Variablen wird auch *Verpacken* eines Tupels genannt.

Die Ihnen bereits bekannte `for`-Schleife arbeitet mit einem Tupel.

Ein Tupel kann mithilfe einer Zuweisung in mehrere Variablen *entpackt* werden. Dabei muss darauf geachtet werden, dass die Anzahl der Variablen mit der Anzahl der Elemente des Tupels übereinstimmt.

Mithilfe einer mehrfachen Zuweisung können mehrere Werte gleichzeitig mehreren Variablen zugewiesen werden. Auch hier muss auf die passende Anzahl geachtet werden.

Werden bei einer mehrfachen Zuweisung die zugewiesenen Werte verändert, wirkt sich das erst nach dem Ende der gesamten Zuweisung aus. Im vorliegenden Beispiel erhält `a` den ursprünglichen Wert von `c`, `b` den ursprünglichen Wert von `a` und `c` die Summe der ursprünglichen Werte von `a` und `b`.

Mithilfe eines *Ausdrucks mit Stern* (englisch: *starred expression*) kann eine mehrfache Zuweisung flexibler gestaltet werden. Sie können vor einer der Variablen den Operator `*` setzen. Damit verweist diese Variable auf eine Liste, die nach der Zuweisung die überzähligen Werte enthält.

Im vorliegenden Beispiel werden der erste und der letzte Wert den Variablen `a` und `c` zugewiesen. Die mittleren Werte (hier sind es drei) werden in der Liste `b` gespeichert. Werden nur zwei Elemente zugewiesen, ist die Liste `b` leer.

4.5 Dictionaries

Ein Dictionary oder assoziativer Array ist mit einem Wörterbuch zu vergleichen. In einem Wörterbuch finden Sie unter einem Schlüsselbegriff die zugeordnete Information. So steht etwa in einem englisch-deutschen Wörterbuch unter dem Eintrag *house* der zugeordnete deutsche Begriff *Haus*.

4.5.1 Eigenschaften, Operatoren und Methoden

In Python stellen Dictionarys veränderliche Objekte dar und enthalten Paare. Jedes Paar besteht aus einem eindeutigen Schlüssel und einem zugeordneten Wert. Über den Schlüssel greifen Sie auf den Wert zu. Als Schlüssel werden meist Strings verwendet, es können aber auch Zahlen genutzt werden. Die Paare eines Dictionarys sind ungeordnet.

Im folgenden Beispiel werden die Namen und Altersangaben mehrerer Personen in einem Dictionary erfasst und bearbeitet. Der Name dient als Schlüssel. Über diesen Schlüssel kann auf die Altersangabe (also auf den Wert) zugegriffen werden.

```
dc = {"Peter":31, "Julia":28, "Werner":35}
print("Dictionary:", dc)
print("Anzahl:", len(dc))
print()

dc_vergleich = {"Peter":31, "Werner":35, "Julia":28}
if dc == dc_vergleich:
    print("Gleich")

dc[ "Julia"] = 27
print("Wert ersetzt:", dc)

dc[ "Moritz"] = 22
print("Element hinzugefügt:", dc)
print()

if "Julia" in dc:
    print(dc[ "Julia"])
del dc[ "Julia"]
if "Julia" not in dc:
```

```

        print("Nicht enthalten")
print("Element entfernt:", dc)

dc_hinzu = {"Moritz": 18, "Karin": 29}
dc.update(dc_hinzu)
print("Update:", dc)

```

Listing 4.28 Datei »dictionary.py«

Folgende Ausgabe wird erzeugt:

```

Dictionary: {'Peter': 31, 'Julia': 28, 'Werner': 35}
Anzahl: 3

Gleich
Wert ersetzt: {'Peter': 31, 'Julia': 27, 'Werner': 35}
Element hinzugefügt: {'Peter': 31, 'Julia': 27, 'Werner': 35, 'Moritz': 22}

27
Nicht enthalten
Element entfernt: {'Peter': 31, 'Werner': 35, 'Moritz': 22}
Update: {'Peter': 31, 'Werner': 35, 'Moritz': 18, 'Karin': 29}

```

Das Dictionary `dc` wird mit drei Paaren erzeugt und ausgegeben. Dictionarys werden mithilfe von geschweiften Klammern erzeugt. Die Paare werden durch Kommata voneinander getrennt, ein Paar wird in der folgenden Form notiert: *Schlüssel:Wert*.

Die Funktion `len()` liefert die Länge, also die Anzahl der Paare des Dictionarys.

Sie können Dictionarys nur mithilfe der beiden Operatoren `==` und `!=` miteinander vergleichen. Zwei Dictionarys stimmen überein, wenn Sie dieselben Paare enthalten, also dieselben Schlüssel mit denselben Werten. Da die Paare eines Dictionarys ungeordnet sind, ist beim Vergleich die Reihenfolge der Elemente nicht relevant.

Auf einen einzelnen Wert greifen Sie mithilfe des Schlüssels in rechteckigen Klammern zu. Bei der Zuweisung eines Paars wird geprüft, ob der Schlüssel bereits existiert. Ist das der Fall, wird nur der alte Wert durch den neuen Wert ersetzt. Ist das nicht der Fall, wird das Paar zum Dictionary hinzugefügt.

Mithilfe des Operators `in` prüfen Sie, ob ein bestimmter Schlüssel im Dictionary existiert. Das ist zum Beispiel vor der Ausgabe

eines Werts oder vor dem Entfernen eines Paars mithilfe des Operators `del` notwendig. Beim Versuch, auf einen nicht existierenden Schlüssel zuzugreifen, tritt eine Ausnahme auf.

Die Methode `update()` dient zum Aktualisieren der Paare eines neuen Dictionarys zu einem vorhandenen Dictionary. Bei der Zuweisung der neuen Paare wird jeweils geprüft, ob der Schlüssel bereits existiert. Ist das der Fall, wird nur der alte Wert durch den neuen Wert ersetzt. Ist das nicht der Fall, wird das Paar zum Dictionary hinzugefügt.

4.5.2 Dynamische Views

Die Methoden `keys()`, `items()` und `values()` erzeugen sogenannte dynamische Views eines Dictionarys. Ändert sich das Dictionary, ändern sich diese Views ebenfalls. Nachfolgend wird mit diesen Views gearbeitet:

```
dc = {"Peter":31, "Julia":28, "Werner":35}
print("Dictionary:", dc)

k = dc.keys()
print("Schlüssel:", k)
for schluessel in k:
    print(schluessel)
if "Werner" in k:
    print("Schlüssel Werner ist enthalten")
print()

v = dc.values()
print("Werte:", v)
for wert in v:
    print(wert)
if 31 in v:
    print("Wert 35 ist enthalten")
print()

i = dc.items()
print("Items:", i)
for k, v in i:
    print(f"Schlüssel {k}, Wert {v}")
```

Listing 4.29 Datei »dictionary_view.py«

Das Programm erzeugt die folgende Ausgabe:

```
Dictionary: {'Peter': 31, 'Julia': 28, 'Werner': 35}
Schlüssel: dict_keys(['Peter', 'Julia', 'Werner'])
Peter
Julia
Werner
Schlüssel Werner ist enthalten

Werte: dict_values([31, 28, 35])
31
28
35
Wert 35 ist enthalten

Items: dict_items([('Peter', 31), ('Julia', 28), ('Werner', 35)])
Schlüssel Peter, Wert 31
Schlüssel Julia, Wert 28
Schlüssel Werner, Wert 35
```

Mithilfe der Methode `keys()` wird eine View der Schlüssel des Dictionarys erzeugt, die anschließend ausgegeben wird. Die einzelnen Schlüssel werden mithilfe einer `for`-Schleife ausgegeben. Mithilfe des Operators `in` wird geprüft, ob ein bestimmter Schlüssel in der View existiert.

Die Methode `values()` dient zur Erzeugung einer View der Werte des Dictionarys, die anschließend ausgegeben wird. Die einzelnen Werte werden mithilfe einer `for`-Schleife ausgegeben. Mithilfe des Operators `in` wird geprüft, ob ein bestimmter Wert in der View existiert.

Mithilfe der Methode `items()` wird eine View der Schlüssel-Wert-Paare des Dictionarys erzeugt, die anschließend ausgegeben wird. Die einzelnen Paare werden mithilfe einer `for`-Schleife ausgegeben. Dabei wird jeweils das Tupel aus Schlüssel und Wert in zwei Variablen entpackt.

4.6 Sets, Mengen

Sets (deutsch: Mengen) unterscheiden sich von Listen und Tupeln dadurch, dass jedes Element nur einmal existiert.

Sets sind ungeordnet, daher ist auch die Reihenfolge bei der Ausgabe eines Sets nicht festgelegt. Einzelne Elemente können nicht anhand eines Slice bestimmt werden. Allerdings können Sie mit Sets einige interessante Operationen durchführen, die aus der Mengenlehre bekannt sind.

4.6.1 Eigenschaften, Operatoren und Methoden

Im folgenden Beispiel wird ein Set erzeugt und bearbeitet:

```
st = set([ 8, 5, 5, 2, 5 ])
print("Set:", st)
print("Anzahl:", len(st))

for x in st:
    print(x)
if 5 in st:
    print("Wert ist enthalten")

su = set([ 2, 8 ])
if su < st:
    print("Echte Teilmenge")
sv = set([ 2, 8, 5 ])
if sv <= st:
    print("Teilmenge")

st.add(-12)
st.add(-12)
print("Element hinzugefügt:", st)

st.discard(5)
print("Element entfernt:", st)

sk = st.copy()
print("Kopie:", sk)

sk.clear()
print("Geleert:", sk)

sf = frozenset([ 8, 5, 5, 2, 5 ])
print("Frozenset:", sf)
```

Listing 4.30 Datei »set_eigenschaft.py«

Die Ausgabe lautet:

```
Set: {8, 2, 5}
Anzahl: 3
8
2
5
Wert ist enthalten
Echte Teilmenge
Teilmenge
Element hinzugefügt: {8, 2, -12, 5}
Element entfernt: {8, 2, -12}
Kopie: {8, 2, -12}
Geleert: set()
FrozenSet: frozenset({8, 2, 5})
```

Ein Set erzeugen Sie mithilfe der Funktion `set()`. Ihr wird als einziger Parameter eine Liste oder ein anderes Iterable übergeben. In der hier übergebenen Liste sind Werte mehrfach enthalten, im Set nur noch einmal.

Die Funktion `len()` liefert die Länge, also die Anzahl der Elemente eines Sets. Mithilfe einer `for`-Schleife können Sie das Set durchlaufen. Mit `in` prüfen Sie, ob ein bestimmtes Element im Set enthalten ist.

Mit den vier Operatoren `<`, `<=`, `>` und `>=` stellen Sie fest, ob ein Set eine Teilmenge oder eine echte Teilmenge eines anderen Sets ist. Dabei ist die Reihenfolge der Elemente im Set unerheblich:

- Das Set `su` ist eine echte Teilmenge des Sets `st`: Alle Elemente von `su` sind in `st` enthalten, und `su` hat weniger Elemente als `st`.
- Das Set `sv` ist eine einfache Teilmenge des Sets `st`: Alle Elemente von `sv` sind in `st` enthalten, aber `sv` hat ebenso viele Elemente wie `st`.

Mithilfe der Methode `add()` fügen Sie ein Element hinzu, falls es noch nicht enthalten ist. Die Methode `discard()` dient zum Löschen eines bestimmten Elements. Es stellt kein Problem dar, falls es nicht enthalten ist. Die Methode `copy()` erzeugt ein neues

Set als Kopie des alten Sets. Die Methode `clear()` entfernt alle Elemente aus dem Set.

Der englische Begriff *frozen* bedeutet »eingefroren«. Ein Frozenset stellt die unveränderliche Variante eines Sets dar. Es wird mithilfe der Funktion `frozenset()` erzeugt und bei der Ausgabe durch den Begriff `frozenset`. Mithilfe einer `for`-Schleife kann es durchlaufen und mithilfe des Operators `in` geprüft werden. Der Versuch, es zu verändern, führt zu einer Ausnahme.

4.6.2 Mengenlehre

Im nachfolgenden Programm werden mithilfe der Operatoren `|`, `&`, `-` und `^` einige Operationen aus der mathematischen Mengenlehre durchgeführt:

```
st = set([ 8, 15, "x" ])
su = set([ 4, "x", "abc", 15 ])
print("st:", st)
print("su:", su)

print("Vereinigungsmenge:", st | su)
print("Schnittmenge:", st & su)
print("Differenzmenge st-su:", st - su)
print("Differenzmenge su-st:", su - st)
print("Symmetrische Differenzmenge:", st ^ su)
```

Listing 4.31 Datei »set_mengenlehre.py«

Die Ausgabe lautet:

```
st: {8, 'x', 15}
su: {'x', 4, 15, 'abc'}
Vereinigungsmenge: {'x', 'abc', 4, 8, 15}
Schnittmenge: {'x', 15}
Differenzmenge st-su: {8}
Differenzmenge su-st: {'abc', 4}
Symmetrische Differenzmenge: {4, 8, 'abc'}
```

Der Operator `|` dient zur Vereinigung zweier Sets. Das entstehende Set enthält alle Elemente, die in einem der beiden Sets enthalten sind. Auch im neuen Set kommt jedes Element nur einmal vor.

Die Elemente, die in beiden Sets enthalten sind, bilden die Schnittmenge. Sie wird mithilfe des Operators & ermittelt.

Bei einer Differenzmenge müssen Sie beachten, welches Set von welchem anderen Set abgezogen wird. Mithilfe des Operators - werden zwei verschiedene Differenzmengen erstellt. Die Operation $st - su$ zieht vom Set st alle Elemente ab, die auch in su enthalten sind. Bei der Operation $su - st$ verhält es sich umgekehrt.

Bei der symmetrischen Differenzmenge werden mit dem Operator ^ die Elemente ermittelt, die nur in einem der beiden Sets enthalten sind.

4.7 Wahrheitswerte und Nichts

Objekte und Ausdrücke können wahr oder falsch sein, außerdem gibt es auch das Nichts-Objekt. Betrachten wir einige Zusammenhänge.

4.7.1 Wahrheitswerte »True« und »False«

Alle Objekte in Python besitzen einen Wahrheitswert: `True` oder `False`. Bei Vergleichen zur Steuerung von Verzweigungen und Schleifen wird ein solcher Wahrheitswert ermittelt, siehe auch [Abschnitt 3.3.1](#), »Vergleichsoperatoren«. Wahrheitswerte können in Variablen des Datentyps `bool` gespeichert werden. Die Funktion `bool()` liefert den Wahrheitswert eines Ausdrucks oder eines Objekts.

Folgende Objekte ergeben `True`:

- eine Zahl ungleich 0, also größer als 0 oder kleiner als 0
- eine nicht leere Sequenz (String, Liste, Tupel)
- ein nicht leeres Dictionary
- eine nicht leere Menge

Folgende Objekte ergeben `False`:

- eine Zahl, die den Wert 0 hat
- eine leere Sequenz (String "", Liste [], Tupel ())
- ein leeres Dictionary: {}
- eine leere Menge: `set()`, `frozenset()`
- eine Objektsammlung `x`, für die gilt `len(x) == 0`
- die Konstante `None` (siehe [Abschnitt 4.7.2](#), »Nichts, ›None‹«)

Für Liste, Dictionary und Set gilt: Solange die Objektsammlung Elemente enthält, ergibt sich der Wahrheitswert `True`. Werden die Elemente entfernt, ergibt sich der Wahrheitswert `False`. Im folgenden Programm wird der Wahrheitswert verschiedener Objekte überprüft und ausgegeben.

```
print("5>3:", 5>3)
print("5<3:", 5<3)
print("Typ von 5>3:", type(5>3))

print("Zahl -0.1:", bool(-0.1))
print("Zahl 0:", bool(0))

print("String 'Hallo':", bool("Hallo"))
print("String '' :", bool("")))

print("Liste [ 2,8] :", bool([ 2,8]))
print("Liste []:", bool([]))

print("Tupel (2,8):", bool((2,8)))
print("Tupel ():", bool(()))

print("Dictionary {'Julia':28, 'Werner':32}:",
      bool({'Julia':28, 'Werner':32}))
print("Dictionary {}:", bool({}))

print("Set (2,8,2):", bool(set([ 2,8,2])))
print("Set ():", bool(set([])))
```

Listing 4.32 Datei »wahrheitswert.py«

Das Programm erzeugt die Ausgabe:

```
5>3: True
5<3: False
Typ von 5>3: <class 'bool'>
Zahl -0.1: True
Zahl 0: False
String 'Hallo': True
String '' : False
Liste [2,8]: True
Liste []: False
Tupel (2,8): True
Tupel (): False
Dictionary {'Julia':28, 'Werner':32}: True
Dictionary {}: False
Set (2,8,2): True
Set (): False
```

String, Liste, Tupel, Dictionary und Set ergeben `False`, wenn sie leer sind, und `True`, wenn sie nicht leer sind. Dies können Sie zur Prüfung der betreffenden Objekte nutzen.

4.7.2 Nichts, »None«

Das Schlüsselwort `None` bezeichnet das Nichts-Objekt. `None` ist das einzige Objekt des Datentyps `NoneType`. Funktionen ohne Rückgabewert liefern `None` zurück. Dies kann auf Folgendes hinweisen:

- dass Sie eine Funktion falsch einsetzen, bei der Sie einen Rückgabewert erwarten
- dass eine Funktion kein Ergebnis liefert, obwohl dies erwartet wird.

Ein Beispiel:

```
def summe(a, b):
    c = a + b

# Programm
erg = summe(7, 12)
if not erg:
    print("Fehler")
if erg is None:
    print("Fehler")
print("Ergebnis:", erg)
print("Wahrheitswert:", bool(erg))
print("Typ:", type(erg))
```

Listing 4.33 Datei »nichts.py«

Die Ausgabe lautet:

```
Fehler
Fehler
Ergebnis: None
Wahrheitswert: False
Typ: <class 'NoneType'>
```

Bei der Definition der Funktion `summe()` wird die Rückgabe des Ergebnisses »vergessen«. Daher erhält `erg` den Wert `None`. Das Nichts-Objekt hat den Wahrheitswert `False`. Damit können Sie feststellen, dass ein Fehler vorliegt.

4.8 Referenz, Identität und Kopie

In diesem Abschnitt erläutere ich den Zusammenhang zwischen Objekten und Referenzen. Wir untersuchen die Identität von Objekten und erzeugen Kopien von Objekten.

4.8.1 Referenz und Identität

Der Name eines Objekts entspricht einer Referenz auf das Objekt.

Weisen Sie diese Referenz einem anderen Namen zu, erzeugen Sie eine zweite Referenz auf dasselbe Objekt. Mithilfe des Identitätsoperators `is` stellen Sie fest, dass beide Referenzen auf dasselbe Objekt verweisen.

Wird das Objekt über die zweite Referenz geändert, zeigt sich eine der beiden folgenden Verhaltensweisen:

- Im Fall eines einfachen Objekts wie Zahl oder String wird ein zweites Objekt erzeugt, in dem der neue Wert gespeichert wird. Die beiden Referenzen verweisen danach auf zwei verschiedene Objekte.
- Im Fall eines nicht einfachen Objekts wie Liste, Tupel, Dictionary, Set usw. wird das Originalobjekt geändert. Es gibt nach wie vor ein Objekt mit zwei verschiedenen Referenzen.

Mithilfe des Operators `==` stellen Sie fest, ob zwei Objekte den gleichen Inhalt haben, ob also zum Beispiel zwei Listen die gleichen Elemente enthalten.

Im folgenden Beispiel werden nacheinander eine Zahl, ein String und eine Liste erzeugt und zweimal referenziert. Anschließend wird der zweiten Referenz jeweils ein neuer Inhalt zugewiesen. Identität und Inhalt werden anhand der beiden Operatoren `is` und `==` festgestellt.

```

print("Zahl:")
x = 12.5
y = x
print("dasselbe Objekt:", x is y)
y = 15.8
print("dasselbe Objekt:", x is y)
print("gleicher Inhalt:", x == y)
print()

print("String:")
x = "Robinson"
y = x
print("dasselbe Objekt:", x is y)
y = "Freitag"
print("dasselbe Objekt:", x is y)
print("gleicher Inhalt:", x == y)
print()

print("Liste:")
x = [ 23, "hallo", -7.5]
y = x
print("dasselbe Objekt:", x is y)
y[1] = "welt"
print("dasselbe Objekt:", x is y)

```

Listing 4.34 Datei »referenz.py«

Es wird die folgende Ausgabe erzeugt:

```

Zahl:
dasselbe Objekt: True
dasselbe Objekt: False
gleicher Inhalt: False

String
dasselbe Objekt: True
dasselbe Objekt: False
gleicher Inhalt: False

Liste:
dasselbe Objekt: True
dasselbe Objekt: True

```

Die Ausgabe zeigt, dass die Objekte zunächst jeweils identisch sind.

Durch die Zuweisung eines neuen Werts wird bei einer Zahl oder einem String ein neues Objekt erzeugt. Die Inhalte sind danach unterschiedlich.

Die Liste (hier stellvertretend auch für andere Objekte) existiert insgesamt nur einmal, auch wenn einzelne Elemente der Liste

verändert werden. Sie können über beide Referenzen auf dieselbe Liste zugreifen.

4.8.2 Ressourcen sparen

Python spart gern Ressourcen. Dies kann zu einem ungewöhnlichen Verhalten führen: Wird einem Objekt über eine Referenz ein Wert zugewiesen *und* wird auf denselben Wert bereits von einer anderen Referenz verwiesen, kann es geschehen, dass die beiden Referenzen anschließend auf dasselbe Objekt verweisen. Python spart also Speicherplatz.

Das Schlüsselwort `del` dient zur Löschung von nicht mehr benötigten Referenzen. Ein Objekt, auf das zwei Referenzen verweisen, wird durch das Löschen der ersten Referenz nicht gelöscht.

Ein Beispiel:

```
x = 42
y = 56
print(f"x:{x}, y:{y}, identisch:{x is y}")

y = 42
print(f"x:{x}, y:{y}, identisch:{x is y}")

del y
print("x:", x)

del x
try:
    print("x:", x)
except:
    print("Fehler")
```

Listing 4.35 Datei »ressourcen.py«

Es wird die folgende Ausgabe erzeugt:

```
x:42, y:56, identisch:False
x:42, y:42, identisch:True
x: 42
Fehler
```

Zunächst erhalten die Variablen `x` und `y` unterschiedliche Werte. Damit handelt es sich bei ihnen um zwei Referenzen auf zwei

verschiedene Objekte. Anschließend erhält die Variable `y` denselben Wert wie die Variable `x`. Nun gibt es nur noch ein Objekt, auf das zwei Referenzen verweisen.

Die Referenz `y` wird gelöscht. Das Objekt existiert weiterhin und kann über die Referenz `x` erreicht werden. Anschließend wird die Referenz `x` gelöscht. Die Ausgabe über diese Referenz führt zu einem Fehler, da der Name nicht mehr existiert.

4.8.3 Objekte kopieren

Echte Kopien von nicht einfachen Objekten wie Liste, Tupel, Dictionary, Set usw. können Sie wie folgt erzeugen: Sie erzeugen ein leeres Objekt und fügen die einzelnen Elemente hinzu.

Für umfangreiche Objekte, die wiederum andere Objekte enthalten, kann das sehr aufwendig werden. Sie können daher auch die Funktion `deepcopy()` aus dem Modul `copy` verwenden.

Beide Vorgehensweisen werden in folgendem Programm gezeigt.

```
import copy

x = [ 23, "hallo", -7.5]
y = []
for i in x:
    y.append(i)
print("dasselbe Objekt:", x is y)
print("gleicher Inhalt:", x == y)
print()

x = [ 23,[ "Berlin", "Hamburg"], -7.5, 12, 67]
y = copy.deepcopy(x)
print("dasselbe Objekt:", x is y)
print("gleicher Inhalt:", x == y)
```

Listing 4.36 Datei »kopieren.py«

Das Programm erzeugt die Ausgabe:

```
dasselbe Objekt: False
gleicher Inhalt: True

dasselbe Objekt: False
gleicher Inhalt: True
```

Die Ausgabe zeigt, dass in beiden Fällen jeweils ein neues Objekt erzeugt wird. Die Inhalte der beiden Objekte sind allerdings gleich.

5 Weiterführende Programmierung

In diesem Kapitel werden die Kenntnisse aus dem Programmierkurs im Zusammenhang mit den verschiedenen Datentypen durch nützliche Praxistipps erweitert.

5.1 Allgemeines

Im ersten Teil des Kapitels erläutere ich einige nützliche Techniken, die keinem bestimmten Thema zuzuordnen sind.

5.1.1 Kombinierte Zuweisungsoperatoren

Neben der einfachen Zuweisung eines Werts zu einer Variablen gibt es auch die kombinierten Zuweisungsoperatoren. Diese verbinden die normalen Operatoren für Zahlen oder Zeichenketten mit der Zuweisung eines Werts. Die betreffende Variable wird also unmittelbar um den genannten Wert verändert. Dies ist besonders bei umfangreichen Ausdrücken oder bei längeren Variablennamen sinnvoll. Ein Beispiel:

Der Ausdruck `TemperaturInCelsius += 5` ist überschaubarer als der Ausdruck `TemperaturInCelsius = TemperaturInCelsius + 5`. Beide Ausdrücke erhöhen den Wert der Variablen `TemperaturInCelsius` um 5.

Es folgt ein Beispiel, in dem kombinierte Zuweisungsoperatoren für Zahlen und für Zeichenketten eingesetzt werden.

```

x = 12
print(x)
x += 3      # Erhöhen von x
print(x)
x -= 9      # Vermindern von x
print(x)
x **= 2      # Quadrieren von x
print(x)
x *= 3      # Verdreifachen von x
print(x)
x //= 7      # Ganzzahliges Teilen von x
print(x)
x /= 4      # Teilen von x
print(x)
x %= 2      # Dividieren, Rest berechnen
print(x)

t = "Hallo "
print(t)
t += "Python "  # Anhängen an t
print(t)
t *= 3          # Verdreifachen von t
print(t)

```

Listing 5.1 Datei »zuweisung_kombiniert.py«

Die Ausgabe:

```

12
15
6
36
108
15
3.75
1.75
Hallo
Hallo Python
Hallo Python Hallo Python Hallo Python

```

Die Variable `x` erhält zunächst den Zahlenwert 12. Durch die nachfolgenden Anweisungen wird ihr Wert jedes Mal verändert. Der Wert wird als Erstes um 3 erhöht (= 15), anschließend um 9 vermindert (= 6), danach hoch 2 gerechnet (= 36) und mit 3 multipliziert (= 108).

Es folgt eine Ganzzahldivision, dabei werden die Nachkommastellen abgeschnitten. Die verbleibende Zahl (15) wird durch 4 geteilt (= 3,75). Zuletzt wird der Rest der Division durch 2 berechnet (= 1,75).

Die Variable `t` erhält den Zeichenkettenwert `Hallo`. Anschließend wird sie verlängert und vervielfacht.

Bei den Änderungen ist darauf zu achten, dass die betreffende Variable vorher bereits einen Wert hat, sonst tritt ein Fehler auf.

5.1.2 Anweisung in mehreren Zeilen

In [Abschnitt 2.3.6](#), »Lange Ausgaben«, wurde erläutert, wie Sie eine lange Zeichenkette bei Einsatz der Funktion `print()` über mehrere Zeilen verteilen. In diesem Abschnitt zeige ich, wie Sie lange Anweisungen allgemein zerlegen können, unter anderem mithilfe des Zeichens `\`. Ein Beispiel:

```
print("Bitte geben Sie eine"
      " Temperatur in Celsius ein:")
TemperaturInCelsius = float(input())

TemperaturInFahrenheit = TemperaturInCelsius \
    * 9 / 5 + 32

print(TemperaturInCelsius, "Grad Celsius entsprechen",
      TemperaturInFahrenheit, "Grad Fahrenheit")
```

Listing 5.2 Datei »zeile_lang.py«

Die Ausgabe lautet:

```
Bitte geben Sie eine Temperatur in Celsius ein:
5.2
5.2 Grad Celsius entsprechen 41.36 Grad Fahrenheit
```

Zunächst wird eine längere Zeichenkette auf zwei Zeilen verteilt. Jeder Teil der Zeichenkette wird in Anführungszeichen gesetzt. Ein trennendes Leerzeichen zwischen den beiden Teilen muss von Hand eingegeben werden.

Bei einer längeren Anweisung, die innerhalb einer Berechnung getrennt werden muss, kommt das Zeichen `\` zum Einsatz. Es zeigt an, dass die Anweisung in der nächsten Zeile fortgesetzt wird.

Sie können eine Anweisung, die Kommata enthält, auf einfache Weise nach einem der Kommata trennen.

5.1.3 Eingabe mit Hilfestellung

Die eingebaute Funktion `input()` zur Eingabe von Zeichenketten hat einen optionalen Parameter. Damit können Sie eine hilfreiche Information für die Eingabe übergeben. Im folgenden Beispiel soll der Benutzer die Namen von drei Städten eingeben, die in einer Liste abgespeichert werden:

```
li = []
for i in range(3):
    tx = input(f"Bitte {i+1}. Stadt eingeben: ")
    li.append(tx)
print(li)
```

Listing 5.3 Datei »eingabe_hilfe.py«

Die Ausgabe lautet:

```
Bitte 1. Stadt eingeben: Paris
Bitte 2. Stadt eingeben: Rom
Bitte 3. Stadt eingeben: Madrid
['Paris', 'Rom', 'Madrid']
```

Bei der Eingabe wird als Hilfestellung die laufende Nummer der einzugebenden Stadt ausgegeben.

5.1.4 Anweisung »pass«

Die Anweisung `pass` bewirkt, dass nichts ausgeführt wird. Einige mögliche Einsatzzwecke dafür sind:

- Sie entwickeln ein Programm, in dem eine Funktion aufgerufen wird, die erst später entwickelt wird. Zunächst soll das Hauptprogramm geschrieben werden, das auch einen Aufruf der Funktion enthält. In der Definition der Funktion steht zunächst nur die Anweisung `pass`.
- Das Programm enthält eine Verzweigung, bei der in einem bestimmten Zweig nichts ausgeführt werden soll. Dieser Zweig soll aber dennoch erscheinen, um den Programmablauf klarer zu machen.

Ein Programm mit Beispielen zu diesen Fällen sähe wie folgt aus:

```

def QuadraturDesKreises():
    pass
print("Als nächster Schritt wird ein Kreis quadriert")
QuadraturDesKreises()
print("Der Kreis wurde quadriert")

a = -12
b = 6
c = 6.2
if a >= 0 and b >= 0 and c >= 0:
    pass
else:
    print("Eine der Zahlen ist negativ")

if a == 1:
    print("Fall 1")
elif a == 2:
    print("Fall 2")
elif a < 0:
    pass
else:
    print("Ansonsten")

```

Listing 5.4 Datei »pass.py«

Die Ausgabe lautet:

```

Als nächster Schritt wird ein Kreis quadriert
Der Kreis wurde quadriert
Eine der Zahlen ist negativ

```

Die Funktion `QuadraturDesKreises()` dient zunächst nur als Dummy und wird zu einem späteren Zeitpunkt mit Inhalten gefüllt. Sie ist aber bereits im Programm eingebaut und kann aufgerufen werden.

Bei der einfachen Verzweigung erfolgt im `else`-Zweig eine Ausgabe. Bei der mehrfachen Verzweigung erfolgt nur eine Ausgabe, falls der untersuchte Wert größer oder gleich 0 ist.

5.1.5 Funktionen »eval()« und »exec()«

Die Funktionen `eval()` und `exec()` dienen zum Zusammensetzen von Python-Code aus Zeichenketten:

- Die Funktion `eval()` evaluiert einen zusammengesetzten Ausdruck, ermittelt also den Wert (englisch: *value*) des Ausdrucks.

- Die Funktion `exec()` führt eine zusammengesetzte Anweisung aus.

Ein Beispiel:

```
def rechnung1(a, b):
    return a + b

def rechnung2(a, b):
    return a * b

for i in 1, 2:
    print(eval("rechnung" + f"{i}" + "(3,4)"))

for i in 1, 2:
    exec("print(rechnung" + f"{i}" + "(3,4))")
```

Listing 5.5 Datei »eval_exec.py«

Folgende Ausgabe wird erzeugt:

```
7
12
7
12
```

Es werden zunächst die beiden Funktionen `rechnung1()` und `rechnung2()` zur Berechnung der Summe und des Produkts von zwei Zahlen definiert. Die Namen der beiden Funktionen unterscheiden sich nur in der Ziffer.

In beiden Schleifen werden mehrere Zeichenketten zu einer längeren Zeichenkette zusammengesetzt.

In der ersten Schleife enthält die Zeichenkette einen Ausdruck, und zwar den Aufruf einer Funktion: `"rechnung1(3,4)"` bzw. `"rechnung2(3,4)"`. Der Ausdruck wird mithilfe von `eval()` evaluiert. Das bedeutet, dass der Aufruf der Funktion ausgeführt wird. Anschließend enthält der Ausdruck den Rückgabewert der Funktion. Dieser wird ausgegeben.

In der zweiten Schleife enthält die Zeichenkette eine Anweisung, und zwar: `"print(rechnung1(3,4))"` bzw. `"print(rechnung2(3,4))"`. Die Anweisung wird mithilfe von `exec()` ausgeführt. Es wird eine Funktion aufgerufen, deren Rückgabewert ausgegeben wird.

5.2 Ausgabe und Formatierung

In diesem Abschnitt erläutere ich einige Möglichkeiten zur Ausgabe mithilfe der Funktion `print()` und zur Formatierung von Ausgaben.

5.2.1 Funktion »print()«

Die Funktion `print()` haben wir bereits mehrfach eingesetzt. Sie bietet noch weitere Möglichkeiten:

- Das Trennzeichen (englisch: *Separator*), das die ausgegebenen Objekte voneinander trennt, kann mithilfe des benannten Parameters `sep` geändert werden. Es enthält standardmäßig ein Leerzeichen.
- Das Zeilenende, das standardmäßig nach einer Ausgabe folgt, kann mithilfe des benannten Parameters `end` geändert werden.

Ein Beispiel:

```
a = 23
b = 7.5
c = a + b
print("Ergebnis: ", end="")
print(a, "+", b, "=", c, sep="")  
  
li = [ "Hamburg", "Berlin", "Augsburg"]
for stadt in li:
    print("Stadt", stadt, sep="=>", end=" # ")
```

Listing 5.6 Datei »print.py«

Die Ausgabe sieht wie folgt aus:

```
Ergebnis: 23+7.5=30.5
Stadt=>Hamburg # Stadt=>Berlin # Stadt=>Augsburg #
```

Zunächst zur Ausgabe der Berechnung. Zwei Aufrufe der Funktion `print()` erzeugen normalerweise zwei Ausgabezeilen. Hier sieht es anders aus:

- Beim ersten Aufruf wird der Parameter `end` auf eine leere Zeichenkette gesetzt. Das bedeutet, dass die nächste Ausgabe in derselben Zeile erfolgt.
- Beim zweiten Aufruf wird der Parameter `sep` auf eine leere Zeichenkette gesetzt. Dies führt dazu, dass die einzelnen Teile der Ausgabe direkt aneinandergefügt werden, ohne Leerzeichen.

Für die Ausgabe der Liste wurden sowohl der Separator als auch das Zeilenende verändert.

5.2.2 Formatierung von Zahlen mit Nachkommastellen

Formatierte String-Literale gibt es seit Python 3.6. Wir haben sie bereits häufig zur komfortablen Einbettung von Variablen, berechneten Ausdrücken oder Funktionsaufrufen in Zeichenketten genutzt.

In diesem Abschnitt und in den beiden nächsten Abschnitten, [Abschnitt 5.2.3](#) und [Abschnitt 5.2.4](#), werden weitergehende Möglichkeiten gezeigt. Sie können zum Beispiel die Spalten von Tabellen in einer einheitlichen Formatierung übersichtlich ausgeben. Dazu bestimmen Sie unter anderem die Mindestausgabebreite der Zahlen und die Anzahl der Nachkommastellen.

Zunächst werden zwei Zahlen mit Nachkommastellen mithilfe von formatierten String-Literalen ausgegeben:

```
x = 100/7
y = 2/7
print("Zahlen:", x, y)
print()

print(f"Format f, Standard: {x:f} {x:f} {y:f}")
print(f"Format f, nach Komma: {x:.25f}")
print(f"Format f, gesamt: {x:15.10f}")
print()
```

```

print(f"Format e, Standard: {x:e}")
print(f"Format e, nach Komma: {x:.3e}")
print(f"Format e, gesamt: {x:12.3e}")
print()

print(f"Format %, Standard: {y:%}")
print(f"Format %, nach Komma: {y:.3%}")
print(f"Format %, gesamt: {y:12.3%}")

```

Listing 5.7 Datei »literal_nachkomma.py«

Die Ausgabe sieht aus wie in [Abbildung 5.1.](#)

```

Zahlen: 14.285714285714286 0.2857142857142857

Format f, Standard: 14.285714 14.285714 0.285714
Format f, nach Komma: 14.2857142857142864755815026
Format f, gesamt: 14.2857142857

Format e, Standard: 1.428571e+01
Format e, nach Komma: 1.429e+01
Format e, gesamt: 1.429e+01

Format %, Standard: 28.571429%
Format %, nach Komma: 28.571%
Format %, gesamt: 28.571%

```

Abbildung 5.1 Formatierung von Zahlen mit Nachkommastellen

Die beiden Variablen `x` und `y` erhalten die Werte von $100/7$ bzw. von $2/7$. Sie werden zum Vergleich unformatiert ausgegeben.

Wie gewohnt wird ein formatiertes String-Literal mithilfe des Zeichens `f` und der geschweiften Klammern gebildet. Allerdings folgen dem einzubettenden Element jetzt ein Doppelpunkt und die Formatierungszeichen.

Das Formatierungszeichen »`f`« steht für die Ausgabe einer Zahl mit einer bestimmten Anzahl von Nachkommastellen, standardmäßig sechs. Die Angabe `.25f` erzeugt 25 Nachkommastellen. Die Angabe `15.10f` steht für die rechtsbündige Ausgabe einer Zahl auf einer Mindestgesamtbreite von 15 Stellen, davon 10 nach dem Komma. Ein solches Format ist besonders für Tabellen geeignet.

Das Formatierungszeichen »`e`« steht für die Ausgabe einer Zahl im Exponentialformat, standardmäßig mit sechs Nachkommastellen und Exponent. Die Angabe `.3e` erzeugt drei

Nachkommastellen. Die Angabe `12.3e` steht für die rechtsbündige Ausgabe auf einer Mindestgesamtbreite von 12 Stellen, davon 3 nach dem Komma.

Das Formatierungszeichen »%« steht für die Ausgabe einer Zahl im Prozentformat, standardmäßig mit sechs Nachkommastellen und einem Prozentzeichen. Die Zahl wird (nur für die Ausgabe) mit 100 multipliziert, intern bleibt sie unverändert. Die Angabe `.3%` erzeugt drei Stellen nach dem Komma. Die Angabe `12.3%` steht für die rechtsbündige Ausgabe auf einer Mindestgesamtbreite von 12 Stellen, davon 3 nach dem Komma.

Hinweis

Achten Sie auf die richtige Schreibweise bei der Formatierung. Zum Beispiel darf vor der schließenden geschweiften Klammer kein Leerzeichen stehen.

5.2.3 Formatierung von ganzen Zahlen

Es folgt eine Ausgabe von ganzen Zahlen mithilfe von formatierten String-Literalen:

```
for z in range(61, 67):
    print(f"{{z:4d} {{z:9b} {z:4o} {z:4x}}")
```

Listing 5.8 Datei »literal_ganz.py«

Die Ausgabe sieht aus wie in [Abbildung 5.2](#).

61	111101	75	3d
62	111110	76	3e
63	111111	77	3f
64	1000000	100	40
65	1000001	101	41
66	1000010	102	42

Abbildung 5.2 Formatierung von ganzen Zahlen

Es wird eine einheitlich formatierte Zahlentabelle erzeugt. Die Zahlen von 61 bis 66 werden nacheinander ausgegeben als:

- Dezimalzahl (Zeichen `d`, hier in der Mindestgesamtbreite 4)

- Dualzahl (Zeichen `b`, hier in der Mindestgesamtbreite 9)
- Oktalzahl (Zeichen `o`, hier in der Mindestgesamtbreite 4)
- Hexadezimalzahl (Zeichen `x`, hier in der Mindestgesamtbreite 4)

5.2.4 Formatierung von Zeichenketten

In der nachfolgenden Tabelle kommt die Ausgabe von Zeichenketten mithilfe von formatierten String-Literalen hinzu:

```
artname = {23:"Apfel", 8:"Banane", 42:"Pfirsich"}
anzahl = {23:1, 8:3, 42:5}
epreis = {23:2.95, 8:1.45, 42:3.05}

print(f"{' Nr' :>4}{' Name' :>12}{' Anz' :>4}{' EP' :>13}{' GP' :>13} ")
for x in 23, 8, 42:
    print(f"{x:04d}{artname[ x ] :>12}{anzahl[ x ] :4d} "
          f"{epreis[ x ] :8.2f} Euro{anzahl[ x ] * epreis[ x ] :8.2f} Euro")
```

Listing 5.9 Datei »literal_zeichenkette.py«

Die Ausgabe sieht aus wie in [Abbildung 5.3.](#)

Nr	Name	Anz	EP	GP
0023	Apfel	1	2.95 Euro	2.95 Euro
0008	Banane	3	1.45 Euro	4.35 Euro
0042	Pfirsich	5	3.05 Euro	15.25 Euro

Abbildung 5.3 Formatierung von Zeichenketten

Sie können die Formatierung nicht nur für Variablen oder Rechenausdrücke, sondern auch für Werte nutzen, wie hier für die Zeichenketten in der Überschrift der Artikeltabelle. Die Zeichenketten müssen innerhalb von einfachen Anführungsstrichen notiert werden.

Standardmäßig gilt für Zahlen die Rechtsbündigkeit und für Zeichenketten die Linksbündigkeit. Das Formatierungszeichen `>` steht für eine rechtsbündige Ausgabe, die nachfolgende Zahl für die Mindestgesamtbreite der Ausgabe. Das Zeichen `<` steht analog für linksbündig.

Die Artikeltabelle basiert auf den drei Dictionarys für den Artikelnamen, die Anzahl und den Einzelpreis. Die Elemente der Dictionarys werden, zusammen mit dem ermittelten Gesamtpreis, einheitlich ausgegeben.

Steht vor der Angabe der Gesamtbreite eine Null, wird die Zahl mit führenden Nullen aufgefüllt. Dies ist bei der ersten Spalte der Fall. Bei der Aufteilung eines einzelnen String-Literals auf mehrere String-Literale müssen Sie das führende `f` vor jeder Zeichenkette notieren.

Übung »u_literal«

Schreiben Sie das Programm aus der Übung »u_range_inch« so um, dass eine Ausgabe wie in [Abbildung 5.4](#) erzeugt wird. Die Beträge für die Spalten »Inch« und »cm« sollen jeweils mit einer Nachkommastelle angezeigt und rechtsbündig ausgerichtet werden (Datei `u_literal.py`).

Inch	cm
15.0	38.1
20.0	50.8
25.0	63.5
30.0	76.2
35.0	88.9
40.0	101.6

Abbildung 5.4 Übung »u_literal«

5.3 Funktionen für Iterables

Es gibt eine Reihe von Funktionen, die mit Iterables arbeiten. Sie können dem Entwickler viel Arbeit abnehmen. Als Beispiele erläutere ich im Folgenden die Funktionen `zip()`, `map()` und `filter()`.

Python ist eine sehr vielseitige Programmiersprache. Sie nutzt auch Prinzipien der funktionalen Programmierung, und zwar beim Einsatz der nachfolgenden Funktionen bzw. Technik:

- der Funktion `map()`, siehe [Abschnitt 5.3.2](#)
- der Funktion `filter()`, siehe [Abschnitt 5.3.3](#)
- der *List Comprehension*, siehe [Abschnitt 4.3.5](#)
- der *Lambda-Funktion*, siehe [Abschnitt 5.6.8](#)

5.3.1 Funktion »zip()«

Die Funktion `zip()` liefert ein Iterable, in dem die Elemente anderer Iterables miteinander verbunden werden. Ein Beispiel:

```
plz = [ 49808, 78224, 55411]
stadt = [ "Lingen", "Singen", "Bingen"]
bundesland = [ "NS", "BW", "RP"]

kombi = zip(plz, stadt, bundesland)
for element in kombi:
    print(element)
```

Listing 5.10 Datei »iterable_zip.py«

Die Ausgabe lautet:

```
(49808, 'Lingen', 'NS')
(78224, 'Singen', 'BW')
(55411, 'Bingen', 'RP')
```

Zunächst werden verschiedene Iterables erzeugt – in diesem Fall drei Listen, die die Postleitzahlen, die Namen und die zugehörigen Bundesländer dreier Städte enthalten.

Die Funktion `zip()` erhält als Parameter die drei Iterables. In der Funktion werden sie miteinander verbunden. Es wird das Objekt `kombi` zurückgeliefert, das aus Tupeln besteht. Sie werden mithilfe einer `for`-Schleife ausgegeben.

5.3.2 Funktion »map()«

Die Funktion `map()` gibt Ihnen die Möglichkeit, eine Funktion in einer Anweisung mehrfach aufzurufen, jeweils mit anderen Parametern. Sie liefert ein Iterable, das aus den Ergebnissen besteht.

Es folgt ein Programm mit zwei Beispielen:

```
def quad(x):
    return x * x

def summe(a,b,c):
    return a + b + c

z = map(quad, [ 4, 2.5, -1.5] )
print("Quadrat:")
for element in z:
    print(element)
print()

z = map(summe, [ 3, 1.2, 2], [ 4.8, 2], [ 5, 0.1, 9])
print("Summe:")
for element in z:
    print(element)
```

Listing 5.11 Datei »iterable_map.py«

Die Ausgabe lautet:

```
Quadrat:
16
6.25
2.25

Summe:
12.8
3.3000000000000003
```

Zunächst werden zwei Funktionen definiert:

- Die Funktion `quad()` hat einen Parameter und liefert dessen Quadrat.

- Die Funktion `summe()` hat drei Parameter und liefert deren Summe.

Beim ersten Aufruf der Funktion `map()` wird Folgendes übergeben:

- Der erste Parameter ist der Name der Funktion `quad()`.
- Der zweite Parameter ist ein Iterable, in dem die Werte für die verschiedenen Aufrufe der Funktion `quad()` stehen.
- Es wird das Iterable `z` zurückgeliefert. Es enthält die Rückgabewerte der Funktion `quad()` für die verschiedenen Aufrufe.
- Diese Werte werden mithilfe einer `for`-Schleife ausgegeben.

Beim zweiten Aufruf der Funktion `map()` wird Folgendes übergeben:

- Der erste Parameter ist der Name der Funktion `summe()`.
- Der zweite und alle folgenden Parameter sind Iterables, in denen die Werte für die verschiedenen Aufrufe der Funktion stehen.
- Es wird das Iterable `z` zurückgeliefert. Es enthält die Rückgabewerte der Funktion `summe()` für die verschiedenen Aufrufe.
- Diese Werte werden mithilfe einer `for`-Schleife ausgegeben.
- Für die Bildung der ersten Summe wird aus jedem Iterable das erste Element (hier: 3, 4, 8 und 5) verwendet. Für die Bildung der zweiten Summe wird aus jedem Iterable das zweite Element (hier: 1, 2, 2 und 0,1) verwendet usw.
- Das kürzeste Iterable bestimmt die Anzahl der Aufrufe.

5.3.3 Funktion »filter()«

Die Funktion `filter()` untersucht Elemente eines Iterables mithilfe einer Funktion. Sie liefert ein Iterable mit denjenigen Elementen, für die die Funktion den Wahrheitswert `True` zurückliefert.

Ein Beispiel:

```
def positiv(a):
    return True if a>0 else False

z = filter(positiv, [ 5, -6, -2, 0, 12, 3, -5] )
for element in z:
    print(element)
```

Listing 5.12 Datei »iterable_filter.py«

Es wird diese Ausgabe erzeugt:

```
5
12
3
```

Der erste Parameter der Funktion `filter()` ist der Name der Funktion, die für einen untersuchten Wert `True` oder `False` liefert. Der zweite Parameter ist das Iterable. Zurückgeliefert wird ein Iterable, das diejenigen Elemente enthält, für die die Funktion `True` ergibt.

5.4 Verschlüsselung

Die Funktionen des Moduls `random` reichen für die zufälligen Werte eines Spiels aus. Benötigen Sie aber zufällige Werte zum Zweck der Verschlüsselung oder aus Gründen der Sicherheit, sollten Sie die Funktionen des Moduls `secrets` nutzen, das sichere zufällige Zahlen liefert und seit Python 3.6 zur Verfügung steht.

Ein erstes Beispiel:

```
import string, secrets

li = []
for i in range(10):
    li.append(secrets.randrange(6) + 1)
print("Zehnmal würfeln:", li)

tx = ""
for i in range(10):
    tx += secrets.choice(string.ascii_lowercase)
print("Text mit zehn zufälligen kleinen Buchstaben:", tx)
```

Listing 5.13 Datei »secrets_funktionen.py«

Die Funktion `randbelow()` liefert eine zufällige Zahl zwischen 0 und dem Wert des übergebenen Parameters. Hier wird zehnmal gewürfelt.

Die Funktion `choice()` liefert einen zufälligen Wert aus einer Sequenz. Bei der vorliegenden Sequenz handelt es sich um die Zeichenketten-Konstante `ascii_lowercase` aus dem Modul `string`, in dem die kleinen Buchstaben aus dem ASCII-Code stehen, siehe auch [Abschnitt 4.2.8](#), »Konstanten«. Es wird eine Folge von zehn zufälligen kleinen Buchstaben ermittelt und ausgegeben.

Die Ausgabe des Programms:

```
Zehnmal würfeln: [3, 1, 2, 4, 4, 1, 1, 6, 3]
Text mit zehn zufälligen kleinen Buchstaben: oagubwscsc
```

Im nachfolgenden Programm wird mithilfe der Funktion `choice()` ein zufälliges Passwort erstellt, das aus sechs Zeichen besteht. Im Passwort muss jeweils mindestens ein Zeichen aus

den vier nachfolgenden Gruppen vorkommen: kleine Buchstaben, große Buchstaben, Ziffern und Sonderzeichen:

```
import string, secrets

sonder = "!#$%&()*+-/:;<=>?@[ \] _{| }"
print("Sonderzeichen:", sonder)
alle = string.ascii_letters + string.digits + sonder

while True:
    tx = ""
    anz = [ 0, 0, 0, 0]
    for i in range(6):
        zeichen = secrets.choice(alle)
        tx += zeichen
        if zeichen in string.ascii_lowercase:
            anz[ 0] += 1
        elif zeichen in string.ascii_uppercase:
            anz[ 1] += 1
        elif zeichen in string.digits:
            anz[ 2] += 1
        else:
            anz[ 3] += 1
    if 0 not in anz:
        break
print("Werte aus Liste: ", tx)
print("Anzahl:", anz)
```

Listing 5.14 Datei »secrets_password.py«

Zunächst wird eine Zeichenkette erstellt, die aus allen möglichen Sonderzeichen besteht. Als Nächstes wird eine Zeichenkette zusammengefügt, die aus allen erlaubten Zeichen für das Passwort besteht, also aus allen kleinen und großen Buchstaben, allen Ziffern und allen möglichen Sonderzeichen.

Das Passwort wird in einer Endlosschleife erstellt. Diese wird nur verlassen, wenn das Passwort die oben genannten Bedingungen erfüllt. Für das Passwort werden mithilfe der Funktion `choice()` nacheinander sechs Zeichen aus der Menge der erlaubten Zeichen gezogen und zusammengefügt.

Es gibt eine Liste von Zählern, für jede Gruppe einen. Für jedes Zeichen wird die Gruppenzugehörigkeit bestimmt. Der entsprechende Zähler wird erhöht. Gibt es keinen Zähler mehr, der auf 0 steht, wird die Endlosschleife verlassen.

5.5 Fehler und Ausnahmen

Dieser Abschnitt bietet weitergehende Erläuterungen und Techniken im Zusammenhang mit Fehlern und Ausnahmen.

5.5.1 Allgemeines

Während Sie ein Programm entwickeln und testen, treten häufig Fehler auf. Das ist normal und auch wichtig für den Lernprozess. Es gibt drei Arten von Fehlern, Syntaxfehler, Laufzeitfehler und logische Fehler:

- Syntaxfehler bemerken Sie spätestens beim Start eines Programms.
- Laufzeitfehler, also Fehler zur Laufzeit des Programms, die einen Programmabsturz zur Folge haben, können Sie mit einem `try-except`-Block behandeln.
- Logische Fehler treten auf, wenn das Programm vollständig abläuft, aber nicht die erwarteten Ergebnisse liefert. Hier hat der Entwickler den Ablauf nicht richtig durchdacht. Diese Fehler sind erfahrungsgemäß am schwersten zu finden. Dabei bietet das Debugging eine gute Hilfestellung.

5.5.2 Syntaxfehler

Syntaxfehler treten zur Entwicklungszeit des Programms auf und haben ihre Ursache in falsch oder unvollständig geschriebenem Programmcode. Spätestens beim Start eines Programms macht Python auf Syntaxfehler aufmerksam. Die Programmiererin erhält eine Meldung und einen Hinweis auf die Fehlerstelle. Das Programm wird nicht weiter ausgeführt.

Ein Beispiel für einen fehlerhaften Code:

```
x = 12
if x > 10
    print(x)
```

Listing 5.15 Datei »fehler_code.py«

Nach dem Programmstart erscheint eine Fehlermeldung, da ein Doppelpunkt erwartet wird. Seit den neuesten Versionen von Python gibt es eine verbesserte Fehleranalyse mit Hinweisen für Entwickler, siehe [Abbildung 5.5](#).

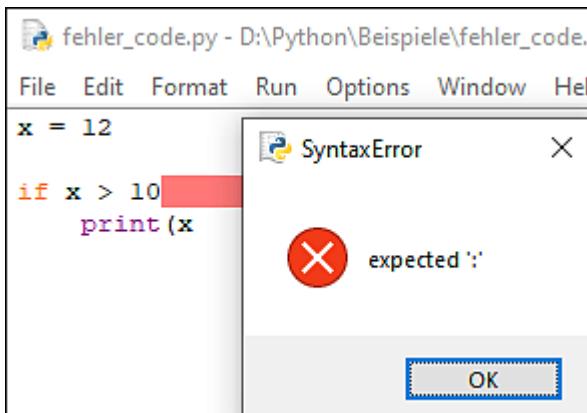


Abbildung 5.5 Anzeige des ersten Fehlers

Das Programm läuft nicht weiter. Es kann erst nach der Verbesserung des Programms erneut gestartet werden. Es erscheint eine weitere Fehlermeldung, da die runde Klammer der Funktion `print()` nicht geschlossen wurde, siehe [Abbildung 5.6](#). Erst nachdem der letzte Fehler beseitigt ist, läuft das Programm bis zum Ende.

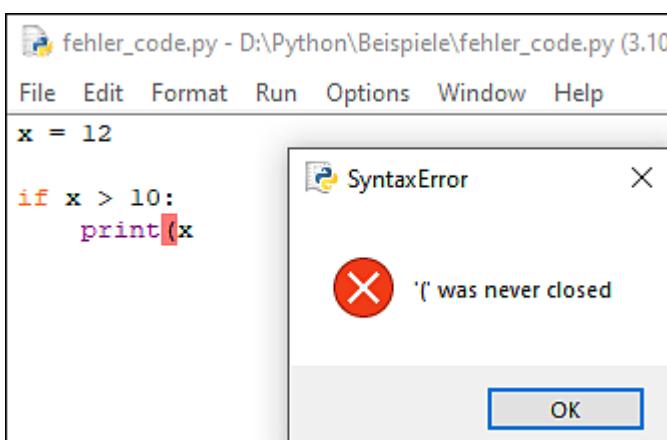


Abbildung 5.6 Anzeige des zweiten Fehlers

5.5.3 Laufzeitfehler

Ein `try-except`-Block dient zum Auffangen von Laufzeitfehlern, die zu Ausnahmen führen, wie bereits in [Abschnitt 3.6, »Fehler und Ausnahmen«](#), angesprochen. Laufzeitfehler treten auf, wenn das Programm versucht, eine unzulässige Operation durchzuführen, zum Beispiel eine Division durch 0 oder das Öffnen einer nicht vorhandenen Datei.

Es ist nicht möglich, Laufzeitfehler vollständig zu vermeiden, da es Vorgänge gibt, auf die die Entwicklerin keinen Einfluss hat. Das kann die fehlerhafte Eingabe eines Benutzers oder eine nicht vorhandene Datei mit Eingabedaten sein. Weitere Möglichkeiten zum Auffangen von Laufzeitfehlern werden in [Abschnitt 5.5.6, »Unterscheidung von Ausnahmen«](#), erläutert.

5.5.4 Logische Fehler und Debugging

Logische Fehler treten auf, wenn eine Anwendung zwar ohne Syntaxfehler übersetzt und ohne Laufzeitfehler ausgeführt wird, aber nicht das geplante Ergebnis liefert. Ursache hierfür ist ein Fehler in der Programmlogik.

Die Ursache logischer Fehler zu finden ist oft schwierig und erfordert intensives Testen und Analysieren der Abläufe und Ergebnisse. Die Entwicklungsumgebung IDLE stellt zu diesem Zweck einen einfachen Debugger zur Verfügung.

Einzelschrittverfahren

Sie können ein Programm im Einzelschrittverfahren ablaufen lassen. Bei jedem dieser Einzelschritte können Sie sich die aktuellen Inhalte von Variablen anschauen. Als Beispiel dient ein einfaches Programm mit einer Schleife und einer Funktion:

```
def summe(a,b):  
    c = a + b  
    return c
```

```

for i in range(5):
    erg = summe(10,i)
    print(erg)

```

Listing 5.16 Datei »fehler_debuggen.py«

Dieses Programm schreibt nacheinander die Zahlen von 10 bis 14 auf den Bildschirm. Öffnen Sie zuerst die **IDLE SHELL** und von dort aus das Programmfenster mit dem obigen Programm.

Sie starten den Debugger, indem Sie in der **IDLE SHELL** im Menü **DEBUG** den Menüpunkt **DEBUGGER** aufrufen. Es erscheint das Dialogfeld **DEBUG CONTROL** und in der **IDLE SHELL** die Meldung [DEBUG ON] , siehe [Abbildung 5.7](#).

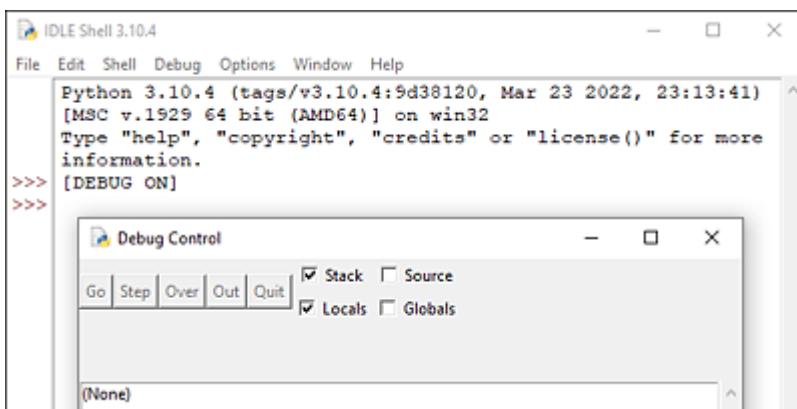


Abbildung 5.7 Dialogfeld »Debug Control« und Meldung

Starten Sie nun das Programm wie gewohnt im Programmfenster über den Menüpfad **RUN** • **RUN MODULE** oder die Taste **F5**. Im Dialogfeld **DEBUG CONTROL** wird auf die erste Zeile des Programms hingewiesen, wie in [Abbildung 5.8](#) zu sehen.

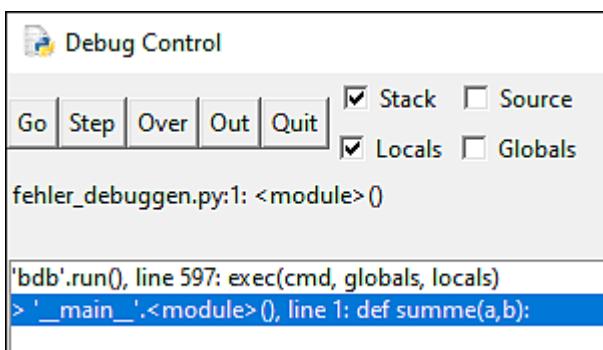


Abbildung 5.8 Nach dem Start des Programms

Jetzt können Sie auch die Buttons im Dialogfeld DEBUG CONTROL betätigen. Mit dem Button STEP gehen Sie schrittweise durch das Programm. Mit dem nächsten Schritt gelangen Sie direkt hinter die Funktionsdefinition zur ersten ausgeführten Programmzeile. Die Funktion wird erst beim Aufruf durchlaufen, siehe Abbildung 5.9.

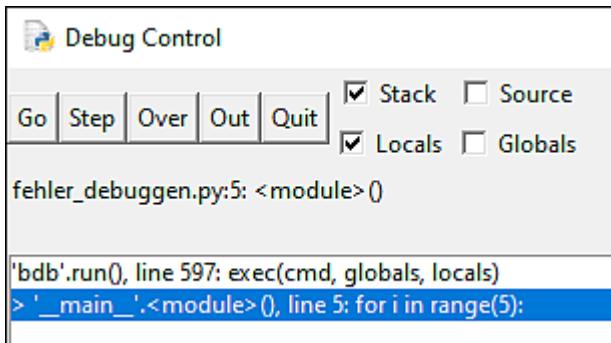


Abbildung 5.9 Erste ausgeführte Programmzeile

Durch wiederholtes Drücken des Buttons STEP können Sie nun die Schleife mehrmals durchlaufen. Dabei wird jedes Mal auch die Funktion `summe()` durchlaufen. Im unteren Bereich des Dialogfelds DEBUG CONTROL sehen Sie die jeweils gültigen Variablen und ihre sich ständig verändernden Werte. Befinden Sie sich bei einem der Schritte im Hauptprogramm in der Schleife, sehen Sie die Werte von `i` und `erg`, siehe Abbildung 5.10.

<code>_package_</code>	None
<code>_spec_</code>	None
<code>erg</code>	10
<code>i</code>	0
<code>summe</code>	<function sum...001F5827324D0>

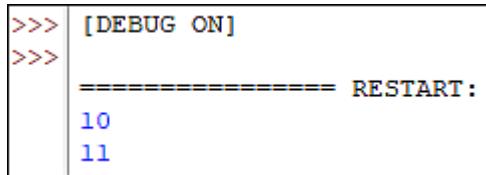
Abbildung 5.10 Hauptprogramm: aktuelle Werte von »i« und »erg«

Wenn Sie sich bei einem der Schritte in der Funktion `summe()` befinden, sehen Sie die Werte von `a`, `b` und `c`, siehe Abbildung 5.11.

Locals	
<code>a</code>	10
<code>b</code>	2
<code>c</code>	12

Abbildung 5.11 Funktion: aktuelle Werte von »a«, »b« und »c«

In der **IDLE SHELL** werden parallel dazu die ersten Ergebnisse ausgegeben, siehe [Abbildung 5.12](#).



```
>>> [DEBUG ON]
>>> ===== RESTART:
10
11
```

Abbildung 5.12 Ausgabe der ersten Ergebnisse in der »Idle Shell«

Nach dem Durchlauf der letzten Programmzeile wird noch der Hinweis `[DEBUG ON]` in der **IDLE SHELL** ausgegeben. IDLE befindet sich nach wie vor im Debug-Modus, aber die Buttons können Sie erst nach einem erneuten Programmstart wieder betätigen.

Weitere Möglichkeiten

Um das Programm in etwas größeren Schritten zu durchlaufen, klicken Sie auf den Button `OVER`. Die Funktionen werden in diesem Fall nicht in Einzelschritten, sondern als Ganzes durchlaufen. Der Debugger springt also über die Funktionen hinweg.

Sie können auch zwischen den beiden Möglichkeiten (Button `STEP` und Button `OVER`) flexibel hin und her wechseln – je nachdem, welchen Programmteil Sie sich genau ansehen möchten.

Wenn Sie sich gerade in Einzelschritten durch eine Funktion bewegen, führt die Betätigung des Buttons `OUT` dazu, dass der Rest der Funktion übersprungen und mit dem ersten Schritt nach dem Funktionsaufruf fortgefahren wird.

Der Button `Go` lässt das Programm, das Sie gerade debuggen, in einem Schritt bis zum Ende laufen. Der Button `QUIT` bricht den Lauf des Programms sofort ab, ohne es zu Ende laufen zu lassen. In beiden Fällen ist der Debug-Modus noch eingeschaltet.

Der Debug-Modus lässt sich an derselben Stelle ausschalten, an der Sie ihn eingeschaltet haben: in der **IDLE SHELL** im Menü `DEBUG` •

DEBUGGER. In der **IDLE SHELL** erscheint anschließend die Meldung
[DEBUG OFF] .

Auf weniger elegante Weise können Sie den Debugger beenden, indem Sie das Dialogfeld **DEBUG CONTROL** einfach schließen.

Andere Entwicklungsumgebungen für Python bieten weitere Möglichkeiten. Das Setzen von *Breakpoints* (deutsch: Haltepunkte) ist sehr nützlich. Diese Haltepunkte werden auf bestimmte Programmzeilen gesetzt. Das Programm läuft dann in einem Zug bis zu einer solchen Programmzeile, und Sie können die aktuellen Werte überprüfen. Anschließend durchlaufen Sie entweder im Einzelschrittverfahren einen Programmreich, in dem Sie Fehler vermuten, oder Sie gehen direkt zum nächsten vorher gesetzten Haltepunkt usw.

5.5.5 Fehler erzeugen

»Wieso sollte man Fehler erzeugen?«, werden Sie sich angesichts dieser Überschrift fragen. Hierfür gibt es, besonders im Zusammenhang mit der Eingabe von Daten durch einen Anwender, durchaus sinnvolle Gründe.

Im folgenden Beispiel wird der Anwender dazu aufgefordert, eine Zahl einzugeben, deren Kehrwert anschließend berechnet wird. Diese Zahl soll allerdings positiv sein. Diese Einschränkung kann mithilfe der Anweisung `raise` bearbeitet werden:

```
while True:
    try:
        zahl = float(input("Eine positive Zahl: "))
        if zahl < 0:
            raise
        kw = 1.0 / zahl
        break
    except:
        print("Fehler")

print(f"Der Kehrwert von {zahl} ist {kw}")
```

Listing 5.17 Datei »fehler_erzeugen.py«

Ist die eingegebene Zahl kleiner als 0, wird die Anweisung `raise` ausgeführt. Dadurch wird eine Ausnahme erzeugt, so als ob der Anwender einen Fehler gemacht hätte. Das Programm verzweigt unmittelbar zur Anweisung `except` und führt die dort angegebenen Anweisungen aus.

In diesem Fall handelt es sich zwar nur um einen logischen Eingabefehler, aber er wird genauso behandelt wie ein Fehler im Programm. Der Anwender wird somit veranlasst, nur positive Zahlen einzugeben. Das folgende Listing zeigt eine mögliche Eingabe:

```
Eine positive Zahl: 0
Fehler
Eine positive Zahl: abc
Fehler
Eine positive Zahl: -6
Fehler
Eine positive Zahl: 6
Der Kehrwert von 6.0 ist 0.1666666666666666
```

Der Benutzer macht verschiedene Fehler:

- Er gibt die Zahl 0 ein. Dies würde bei der Berechnung des Kehrwerts zu einer Ausnahme des Typs `ZeroDivisionError` führen.
- Er gibt einen Text ein. Dies würde beim Aufruf der Funktion `float()` zu einer Ausnahme des Typs `ValueError` führen.
- Er gibt eine negative Zahl ein. Dies führt wegen der vorgenommenen Einschränkung zu einer selbst erzeugten Ausnahme.

Mithilfe der Anweisungen `try`, `raise` und `except` lassen sich also auch nicht sinnvolle Eingaben des Anwenders abfangen und behandeln. Die vorgeführte Technik hat den Nachteil, dass alle Fehler gleichbehandelt werden und die Informationen für den Anwender im Fehlerfall noch nicht sehr genau sind. Dies wird im folgenden Abschnitt verbessert.

5.5.6 Unterscheidung von Ausnahmen

Im folgenden Programm werden unterschiedliche Arten von Fehlern, die zu Ausnahmen führen, spezifisch abgefangen. Damit erfährt die Benutzerin mehr über den Fehler, und es wird eine komfortablere Programmbedienung ermöglicht. Wiederum wird der Kehrwert einer eingegebenen Zahl ermittelt:

```
while True:
    try:
        zahl = float(input("Eine positive Zahl: "))
        if zahl == 0:
            raise RuntimeError("Zahl gleich 0")
        if zahl < 0:
            raise RuntimeError("Zahl zu klein")
        kw = 1.0 / zahl
        break
    except ValueError:
        print("Fehler: keine Zahl")
    except ZeroDivisionError:
        print("Fehler: Zahl 0 eingegeben")
    except RuntimeError as e:
        print("Fehler:", e)

print(f"Der Kehrwert von {zahl} ist {kw}")
```

Listing 5.18 Datei »fehler_unterscheiden.py«

Das Programm enthält zu einem Versuch mehrere spezifische Möglichkeiten des Abfangens. Nachfolgend wird eine mögliche Eingabe gezeigt:

```
Eine positive Zahl: 0
Fehler: Zahl gleich 0
Eine positive Zahl: abc
Fehler: keine Zahl
Eine positive Zahl: -6
Fehler: Zahl zu klein
Eine positive Zahl: 6
Der Kehrwert von 6.0 ist 0.1666666666666666
```

Der Benutzer macht verschiedene Fehler:

- Er gibt die Zahl 0 ein. Dies führt wegen der vorgenommenen Einschränkung zu einem Laufzeitfehler. Dieser wird als allgemeiner `RuntimeError` abgefangen mit der Meldung `Fehler: Zahl gleich 0.`

- Würde die Eingabe von 0 nicht auf diese Weise abgefangen, käme es später bei der Berechnung des Kehrwerts zu einem Laufzeitfehler, einem `ZeroDivisionError`. Dieser würde abgefangen mit der Meldung `Fehler: Zahl 0 eingegeben.` Zu Demonstrationszwecken wird der Fehler zweimal abgefangen.
- Der Benutzer gibt einen Text ein. Dies führt beim Aufruf der Funktion `float()` zu einem Laufzeitfehler, einem `ValueError`. Dieser wird abgefangen mit der Meldung `Fehler: keine Zahl.`
- Er gibt eine negative Zahl ein. Dies führt wegen der zweiten Einschränkung wiederum zu einem Laufzeitfehler. Dieser wird auch als allgemeiner `RuntimeError` abgefangen mit der Meldung `Fehler: Zahl zu klein.`

Beim Erzeugen des Fehlers mit der Anweisung `raise` werden ein Fehler und eine Meldung übergeben. Beim Auffangen des genannten Fehlers mit der Anweisung `except` wird die Meldung mithilfe des Schlüsselworts `as` an die Variable `e` übergeben.

5.6 Funktionen

Python bietet zum Thema »Funktionen« noch einige sehr nützliche Erweiterungen, die in diesem Abschnitt erläutert werden.

5.6.1 Variable Anzahl von Parametern

Bisher wird darauf geachtet, dass die Anzahl der Parameter einer Funktion bei Definition und Aufruf übereinstimmt. Sie können aber auch Funktionen mit einer variablen Anzahl von Parametern definieren.

Bei der Definition einer solchen Funktion notieren Sie vor dem letzten (gegebenenfalls einzigen) Parameter das Zeichen *. Dieser Parameter enthält ein Tupel mit den bis dahin nicht zugeordneten Werten der Parameterkette.

Nachfolgend wird eine Funktion definiert und zweimal aufgerufen, die die Summe aller Parameter berechnet und zurückliefert:

```
def summe(*summanden):
    erg = 0
    for s in summanden:
        erg += s
    return erg

print("Summe:", summe(3, 4))
print("Summe:", summe(3, 8, 12, -5))
```

Listing 5.19 Datei »parameter_variabel.py«

Folgende Ausgabe wird erzeugt:

```
Summe: 7
Summe: 18
```

Die Funktion wird mit zwei bzw. vier Werten aufgerufen. Die for-Schleife dient zur Summierung der Werte des Tupels.

5.6.2 Benannte Parameter

Die definierte Reihenfolge der Parameter muss beim Aufruf nicht eingehalten werden, falls mit *benannten Parametern* gearbeitet wird.

Im nachfolgenden Beispiel sind einige Varianten zum Aufruf der Funktion `volumen()` dargestellt. Diese Funktion berechnet das Volumen eines Quaders und gibt es aus. Zudem wird die übergebene Farbe ausgegeben:

```
def volumen(breite, laenge, tiefe, farbe):
    print("Werte:", breite, laenge, tiefe, farbe)
    print("Volumen:", breite * laenge * tiefe, "Farbe:", farbe)

volumen(4, 6, 2, "rot")
volumen(laenge=2, farbe="gelb", tiefe=7, breite=3)
volumen(5, tiefe=2, laenge=8, farbe="blau")
# volumen(3, tiefe=4, laenge=5, "schwarz")
```

Listing 5.20 Datei »parameter_benannt.py«

Die Ausgabe lautet:

```
Werte: 4 6 2 rot
Volumen: 48 Farbe: rot
Werte: 3 2 7 gelb
Volumen: 42 Farbe: gelb
Werte: 5 8 2 blau
Volumen: 80 Farbe: blau
```

Der erste Aufruf findet in der bekannten Form statt. Es wird ausschließlich mit *positionalen Parametern* gearbeitet. Das heißt, die Zuordnung gelingt über die Position der Parameter.

Beim zweiten Aufruf werden vier benannte Parameter übergeben. In diesem Fall ist die Reihenfolge beim Aufruf nicht wichtig.

Beim dritten Aufruf werden ein positionaler und drei benannte Parameter übergeben. Es sind also auch Mischformen möglich.

Sobald allerdings der erste benannte Parameter beim Aufruf erscheint, müssen alle nachfolgenden Parameter auch benannt sein. Daher würde beim vierten Aufruf ein Fehler auftreten.

5.6.3 Optionale Parameter

Optionale Parameter ermöglichen ebenfalls eine variable Parameterzahl. Sie müssen einen Vorgabewert besitzen und am Ende der Parameterliste stehen. Zusätzlich können benannte Parameter eingesetzt werden.

Die Funktion zur Volumenberechnung des Quaders wird geändert. Es müssen nur noch zwei Parameter angegeben werden. Die anderen beiden Parameter sind optional, es werden gegebenenfalls die Vorgabewerte verwendet.

```
def volumen(breite, laenge, tiefe=1, farbe="schwarz"):
    print("Werte:", breite, laenge, tiefe, farbe)
    print("Volumen:", breite * laenge * tiefe, "Farbe:", farbe)

volumen(4, 6, 2, "rot")
volumen(2, 12, 7)
volumen(5, 8)
volumen(4, 7, farbe="rot")
```

Listing 5.21 Datei »parameter_optional.py«

Das Programm erzeugt die Ausgabe:

```
Werte: 4 6 2 rot
Volumen: 48 Farbe: rot
Werte: 2 12 7 schwarz
Volumen: 168 Farbe: schwarz
Werte: 5 8 1 schwarz
Volumen: 40 Farbe: schwarz
Werte: 4 7 1 rot
Volumen: 28 Farbe: rot
```

Die beiden Parameter `tiefe` und `farbe` sind optional und stehen am Ende der Parameterliste. Es folgen die vier Aufrufe:

- Beim ersten Aufruf werden alle vier Parameter übergeben.
- Beim zweiten Aufruf wird nur der erste optionale Parameter übergeben. Der zweite optionale Parameter erhält daher den Vorgabewert.
- Beim dritten Aufruf werden beide optionalen Parameter nicht übergeben und erhalten ihren Vorgabewert.
- Beim vierten Aufruf wird nur der zweite optionale Parameter übergeben. Da dieser Parameter nicht positional zugeordnet

werden kann, muss er benannt werden.

5.6.4 Mehrere Rückgabewerte

Funktionen können in Python ein Tupel von Rückgabewerten liefern. Im folgenden Beispiel werden in der Funktion `kreis()` die Fläche und der Umfang eines Kreises berechnet und als Tupel zurückgegeben.

```
import math
def kreis(radius):
    flaeche = math.pi * radius * radius
    umfang = 2 * math.pi * radius
    return flaeche, umfang

f, u = kreis(3)
print(f"Fläche: {round(f, 3)}, Umfang: {round(u, 3)}")
x = kreis(3)
print(f"Fläche: {round(f, 3)}, Umfang: {round(u, 3)})")
```

Listing 5.22 Datei »rueckgabe_tupel.py«

Die Ausgabe lautet:

```
Fläche: 28.274, Umfang: 18.85
Fläche: 28.274, Umfang: 18.85
```

Das Schlüsselwort `return` liefert ein Tupel mit den beiden Ergebnissen der Funktion. An der Aufrufstelle muss ein Tupel der passenden Größe zum Empfang bereitstehen, wie beim ersten Aufruf. Es kann aber auch eine Variable bereitstehen, die damit zum Tupel wird, wie beim zweiten Aufruf.

5.6.5 Übergabe von Kopien und Referenzen

Werden Parameter, die an eine Funktion übergeben werden, innerhalb der Funktion verändert, wirkt sich dies unterschiedlich aus:

- Bei der Übergabe eines einfachen Objekts (Zahl oder Zeichenkette) wird eine Kopie des Objekts angelegt. Eine Veränderung der Kopie hat keine Auswirkungen auf das Original.

- Bei der Übergabe eines Objekts, zum Beispiel des Typs Liste, Dictionary oder Set, wird mit einer Referenz auf das Originalobjekt gearbeitet. Eine Veränderung über die Referenz verändert auch das Original.

Zur Verdeutlichung dieses Zusammenhangs werden im folgenden Beispiel insgesamt fünf Parameter an eine Funktion übergeben: eine Zahl, eine Zeichenkette, eine Liste, ein Dictionary und ein Set. Die Objekte werden jeweils dreimal ausgegeben:

- vor dem Aufruf der Funktion
- nach einer Veränderung innerhalb der Funktion
- nach der Rückkehr aus der Funktion

```
def aendern(za, tx, li, di, st):
    za = 8
    tx = "ciao "
    li[ 0] = 7
    di[ "x"] = 7
    st.discard(3)
    print(f"Funktion: {za} {tx} {li} {di} {st}")

hza = 3
htx = "hallo"
hli = [ 3,"abc"]
hdi = {"x":3, "y":"abc"}
hst = set([ 3, "abc"])

print(f"Vorher: {hza} {htx} {hli} {hdi} {hst}")
aendern(hza, htx, hli, hdi, hst)
print(f"Nachher: {hza} {htx} {hli} {hdi} {hst}")
```

Listing 5.23 Datei »parameter_uebergabe.py«

Die Ausgabe lautet:

```
Vorher: 3 hallo [3, 'abc'] {'x': 3, 'y': 'abc'} {'abc', 3}
Funktion: 8 ciao  [7, 'abc'] {'x': 7, 'y': 'abc'} {'abc'}
Nachher: 3 hallo [7, 'abc'] {'x': 7, 'y': 'abc'} {'abc'}
```

Es zeigt sich, dass nur bei Liste, Dictionary und Set eine dauerhafte Veränderung durch die Funktion erfolgte. Dies ist je nach Problemstellung ein erwünschter oder ein unerwünschter Effekt.

Im nachfolgenden Programm wird auch die Veränderung von einfachen Objekten durch eine Funktion dauerhaft gemacht. Dabei wird die Tatsache genutzt, dass Python-Funktionen mehr als einen Rückgabewert haben können. Eine Funktion dient zur aufsteigenden Sortierung von zwei Variablen. Sie werden als Tupel zurückgeliefert:

```
def sortieren(eins, zwei):
    if eins > zwei:
        return zwei, eins
    else:
        return eins, zwei

x = 12
y = 7
print(f"x: {x}, y: {y}")
x, y = sortieren(x, y)
print(f"x: {x}, y: {y}")
```

Listing 5.24 Datei »werte_aendern.py«

Die Ausgabe lautet:

```
x: 12, y: 7
x: 7, y: 12
```

An die Funktion werden zwei Zahlenwerte übergeben. Es wird geprüft, ob der erste Wert größer als der zweite Wert ist:

- Trifft das zu, werden beide Werte in umgekehrter Reihenfolge an die aufrufende Stelle zurückgeliefert.
- Trifft das nicht zu, werden die beiden Werte in unveränderter Reihenfolge an die aufrufende Stelle zurückgeliefert.

5.6.6 Namensräume

Die Definition einer Funktion in Python erzeugt einen lokalen Namensraum. In diesem lokalen Namensraum stehen alle Namen der Variablen, denen innerhalb der Funktion ein Wert zugewiesen wird, und die Namen der Variablen aus der Parameterliste.

Wird bei der Ausführung der Funktion auf eine Variable zugegriffen, so wird diese Variable zunächst im lokalen Namensraum gesucht. Wird der Name der Variablen dort nicht gefunden, wird in dem bisher bekannten globalen Namensraum gesucht, also in den bisher bearbeiteten Programmzeilen außerhalb der Funktion. Wird die Variable auch dort nicht gefunden, tritt ein Fehler auf. Ein erstes Beispiel:

```
def func():
    try:
        print(x)
    except:
        print("Fehler")

# Programm
func()
x = 42
func()
```

Listing 5.25 Datei »global_ohne.py«

Die Ausgabe lautet:

```
Fehler
42
```

Der erste Aufruf von `func()` führt zu einem Fehler, da in der Funktion der Wert der Variablen `x` ausgegeben werden soll. Sie ist nicht im lokalen Namensraum vorhanden, aber auch nicht in den bisher bearbeiteten Programmzeilen außerhalb der Funktion.

Der zweite Aufruf von `func()` führt nicht zu einem Fehler, denn die Variable `x` hat vorher außerhalb der Funktion einen Wert erhalten und ist somit im globalen Namensraum bekannt.

Das Schlüsselwort `global` dient dazu, eine Variable direkt dem globalen Namensraum zuzuordnen. Das ist im nachfolgenden Beispiel notwendig:

```
def eingabe():
    global x
    x = float(input("Zahl: "))

def ausgabe():
    print("Zahl:", x)

# Programm
```

```
eingabe()  
ausgabe()
```

Listing 5.26 Datei »global_mit.py«

In der Funktion `eingabe()` wird ein Wert für die Variable `x` eingelesen und in eine Zahl umgewandelt. Sie wird mithilfe des Schlüsselworts `global` direkt dem globalen Namensraum zugeordnet. Ansonsten wäre sie nur im lokalen Namensraum bekannt und würde daher in der Funktion `ausgabe()` weder im lokalen noch im globalen Namensraum gefunden werden. Die Zuordnung muss vor der Nutzung der Variablen erfolgen.

Testen Sie das Verhalten, indem Sie die Zeile mit dem Schlüsselwort `global` einmal kurzfristig als Kommentar setzen.

Die Ausgabe des Programms sieht zum Beispiel wie folgt aus:

```
zahl: 3.6  
zahl: 3.6
```

5.6.7 Rekursive Funktionen

Bestimmte Abläufe lassen sich am besten rekursiv programmieren. Eine rekursive Funktion ruft sich immer wieder selbst auf. Damit dies nicht zu einer endlosen Menge an Funktionsaufrufen führt, muss es eine Bedingung geben, die der Rekursion ein Ende setzt. Zudem wird ein erster Aufruf benötigt, der die Rekursion einleitet.

Das Prinzip der Rekursion lässt sich bereits anhand des folgenden einfachen Programms verdeutlichen. Sie finden darin die rekursive Funktion `halbieren()`, die bei jedem Aufruf den ihr übergebenen Wert halbiert. Anschließend ruft sie sich selbst wieder auf, und zwar mit dem halbierten Wert. Die Rekursion endet, wenn der Wert, der ständig halbiert wird, eine bestimmte Grenze unterschreitet. Der erste Aufruf der rekursiven Funktion erfolgt mit einem Startwert aus dem Hauptprogramm heraus.

Das Programm:

```

def halbieren(wert):
    print(wert)
    wert = wert / 2
    if wert > 0.05:
        halbieren(wert)

# Programm
halbieren(3)

```

Listing 5.27 Datei »rekursion.py«

Die Ausgabe des Programms:

```

3
1.5
0.75
0.375
0.1875
0.09375

```

Der rekursive Aufruf erfolgt gemäß der Bedingung `wert > 0.05`. Ohne diese Bedingung würde die Rekursion endlos weiterlaufen.

5.6.8 Lambda-Funktion

Lambda-Funktionen werden mithilfe des Schlüsselworts `lambda` erstellt. Sie werden auch *anonyme Funktionen* genannt, im Gegensatz zu den bisher genutzten *benannten Funktionen*. Sie haben im Vergleich zu einer benannten Funktion eine kürzere Definition.

Sie können einer Lambda-Funktion Parameter übergeben. Sie liefert ihr Ergebnis als Ausdruck zurück, der in der gleichen Zeile stehen muss. Darin dürfen keine Mehrfachanweisungen, Ausgaben oder Schleifen vorkommen.

Sie können eine anonyme Funktion einer Variablen zuweisen. Über diese Referenz kann die anonyme Funktion aufgerufen werden.

Ein Beispielprogramm sieht wie folgt aus:

```

mal = lambda x,y: x*y
plus = lambda x,y: x+y

print(mal(5,3))
print(plus(4,7))

```

Listing 5.28 Datei »funktion_lambda.py«

Das Programm erzeugt die folgende Ausgabe:

```
15  
11
```

Die erste Lambda-Funktion wird der Variablen `mal` zugewiesen und hat zwei Parameter. Das Ergebnis der Funktion ist die Multiplikation dieser beiden Parameter. Die zweite Lambda-Funktion wird der Variablen `plus` zugewiesen und ist nach demselben Muster aufgebaut:

[Ergebnis] = `lambda` [Parameterliste] : [Ausdruck]

Eine Lambda-Funktion ermöglicht Ihnen zudem, eine Funktion mit Parametern an einer Stelle zu übergeben, an der nur der Name einer Funktion übergeben werden darf.

5.6.9 Funktion als Parameter

Sie können einer Funktion nicht nur Werte übergeben, sondern auch den Namen einer benannten Funktion oder eine Lambda-Funktion. Eine solche Funktion, die als Parameter eingesetzt wird, wird *Callback* genannt. Damit wird die erstgenannte Funktion flexibler. Ein Beispiel:

```
def hochzwei(x):  
    return x * x  
  
def hochdrei(x):  
    return x * x * x  
  
def ausgabe(unten, oben, schritt, f):  
    for x in range(unten, oben, schritt):  
        print(x, ":", f(x), sep="", end=" ")  
    print()  
  
# Programm  
ausgabe(1, 5, 1, hochzwei)  
ausgabe(1, 5, 1, hochdrei)  
ausgabe(1, 5, 1, lambda x: x * x * x * x)
```

Listing 5.29 Datei »parameter_funktion.py«

Es wird diese Ausgabe erzeugt:

```
1:1 2:4 3:9 4:16
1:1 2:8 3:27 4:64
1:1 2:16 3:81 4:256
```

Zunächst werden die beiden benannten Funktionen `hochzwei()` und `hochdrei()` definiert. Sie liefern als Ergebnis einen einzelnen Wert zurück.

Die Funktion `ausgabe()` dient zur Ausgabe von mehreren Werten und den zugehörigen Funktionswerten. Sie erwartet insgesamt vier Parameter. Die ersten drei Parameter dienen zur Steuerung der `for`-Schleife.

Beim Aufruf der Funktion `ausgabe()` wird als vierter Parameter der Name einer benannten Funktion (ohne runde Klammern) oder eine Lambda-Funktion erwartet. Diese Funktion wird zur Berechnung des Funktionswerts verwendet.

5.7 Eingebaute Funktionen

Als Entwickler können Sie eine Reihe von eingebauten Funktionen ohne Einbindung eines Moduls verwenden.

Tabelle 5.1 gibt eine Übersicht über die eingebauten Funktionen, die in diesem Buch behandelt werden.

Name	Liefert ...	Beispiel in ...
<code>abs()</code>	Betrag einer Zahl	Abschnitt 4.1.10
<code>bin()</code>	binäre (duale) Zahl	Abschnitt 4.1.1
<code>bytes()</code>	Objekt des Datentyps <code>bytes</code>	Abschnitt 4.2.9
<code>chr()</code>	Zeichen zu Unicode-Zahl	Abschnitt 5.7.2
<code>eval()</code>	ausgeführten Python-Ausdruck	Abschnitt 5.1.5
<code>exec()</code>	(Ausführung einer Anweisung)	Abschnitt 5.1.5
<code>filter()</code>	Iterable, für das eine Funktion <code>True</code> ergibt	Abschnitt 5.3.3
<code>float()</code>	Zahl mit Nachkommastellen	Abschnitt 3.2.4
<code>frozenset()</code>	unveränderliches Set	Abschnitt 4.6.1
<code>hex()</code>	hexadezimale Zahl	Abschnitt 4.1.1
<code>input()</code>	Eingabe des Benutzers	Abschnitt 3.2.3

Name	Liefert ...	Beispiel in ...
int()	ganze Zahl	Abschnitt 3.2.4
len()	Anzahl der Elemente	Abschnitt 4.2.1
map()	Iterable mit Ergebnissen mehrerer Aufrufe	Abschnitt 5.3.2
max()	größtes Element	Abschnitt 5.7.1
min()	kleinstes Element	Abschnitt 5.7.1
oct()	oktale Zahl	Abschnitt 4.1.1
open()	Verweis auf geöffnete Datei	Abschnitt 8.2
ord()	Unicode-Zahl zu Zeichen	Abschnitt 5.7.2
print()	Ausgabe	Abschnitt 5.2.1
range()	Iterable über Bereich	Abschnitt 3.4.6
reversed()	Liste in umgekehrter Reihenfolge	Abschnitt 5.7.3
round()	gerundete Zahl	Abschnitt 4.1.5
set()	Set	Abschnitt 4.6
sorted()	sortierte Liste	Abschnitt 5.7.3

Name	Liefert ...	Beispiel in ...
<code>str()</code>	Zeichenkette	Abschnitt 4.2.1
<code>sum()</code>	Summe der Elemente	Abschnitt 5.7.1
<code>type()</code>	Typ eines Objekts	Abschnitt 4.1.3
<code>zip()</code>	Verbindung von Iterables	Abschnitt 5.3.1

Tabelle 5.1 Einige eingebaute Funktionen

5.7.1 Funktionen »max()«, »min()« und »sum()«

Die Funktionen `max()` und `min()` liefern den größten bzw. kleinsten Wert eines Iterables. Die Funktion `sum()` liefert die Summe der Elemente eines Iterables und wird nur mit einem Parameter aufgerufen.

Ein Beispiel zu den drei Funktionen:

```
print("Max. Wert eines Tupels:", max(3, 2, -7))
print("Min. Wert eines Sets:", min(set([3, 2, 3])))
print("Summe eines Tupels:", sum((3, 2, -7)))
```

Listing 5.30 Datei »max_min_sum.py«

Die Ausgabe lautet:

```
Max. Wert eines Tupels: 3
Min. Wert eines Sets: 2
Summe eines Tupels: -2
```

Die Funktion `max()` wird hier mit mehreren Parametern aufgerufen. Sie bilden ein Tupel von Werten, dessen größtes Element bestimmt wird. Die Funktion `min()` wird hier für ein Set aufgerufen. Der einzelne Parameter der Funktion `sum()` ist hier ein Tupel, das vor dem Funktionsaufruf mithilfe von runden Klammern gebildet werden muss.

5.7.2 Funktionen »chr()« und »ord()«

Die Funktion `chr()` liefert das zugehörige Zeichen zu einer Unicode-Zahl. Umgekehrt erhalten Sie mithilfe der Funktion `ord()` die Unicode-Zahl zu einem Zeichen. Ein Beispiel:

```
for i in range(48,58):
    print(chr(i), end="")
print()

for i in range(65,91):
    print(chr(i), end="")
print()

for i in range(97,123):
    print(chr(i), end="")
print()

for z in "abcde":
    print(ord(z), end=" ")
print()

for z in "abcde":
    print(chr(ord(z)+1), end="")
```

Listing 5.31 Datei »chr_ord.py«

Es wird diese Ausgabe erzeugt:

```
0123456789
ABCDEFGHIJKLMNPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
97 98 99 100 101
bcdef
```

An den Positionen 48 bis 57 stehen im Unicode die Ziffern 0 bis 9. An den Positionen 65 bis 90 bzw. 97 bis 122 stehen im Unicode die großen und die kleinen Buchstaben.

Im letzten Teil des Programms wird jedes Zeichen einer Zeichenkette in das codemäßig nachfolgende Zeichen umgewandelt. Dies wäre ein Beispiel für eine sehr einfache Verschlüsselung.

5.7.3 Funktionen »reversed()« und »sorted()«

Die Funktion `reversed()` liefert die Elemente einer Sequenz in umgekehrter Reihenfolge. Mithilfe der Funktion `sorted()` wird

eine sortierte Liste der Elemente einer Sequenz erstellt und geliefert.

Es folgt ein Beispiel, in dem die beiden Funktionen auf ein Tupel und die Werte-View eines Dictionarys angewendet werden:

```
t = 4, 12, 6, -2
print(t)
print(sorted(t))
for i in reversed(t):
    print(i, end=" ")
print()

dc = {"Peter":31, "Julia":28, "Werner":35}
print(dc)
va = dc.values()
print(va)
print(sorted(va))
for i in reversed(va):
    print(i, end=" ")
```

Listing 5.32 Datei »reversed_sorted.py«

Die Ausgabe lautet:

```
(4, 12, 6, -2)
[-2, 4, 6, 12]
-2 6 12 4
{'Peter': 31, 'Julia': 28, 'Werner': 35}
dict_values([31, 28, 35])
[28, 31, 35]
35 28 31
```

Die Funktion `reversed()` liefert eine Sequenz, die die Elemente in umgekehrter Reihenfolge enthält. Sie können mit einer `for`-Schleife ausgegeben werden.

Die Funktion `sorted()` liefert eine Liste, die die Elemente der Sequenz in sortierter Reihenfolge enthält. Bei Zahlen ist die Sortierung aufsteigend nach Wert, bei Zeichen aufsteigend nach der Codenummer.

5.8 Weitere mathematische Module

Bei den mathematisch orientierten Modulen aus diesem Abschnitt handelt es sich um spezielle Themen. Sie können zunächst übergangen und bei Bedarf nachgeschlagen werden.

5.8.1 Funktionsgraphen zeichnen

Mithilfe der Bibliothek `matplotlib` können Sie mathematische Darstellungen erstellen, zum Beispiel den Graphen einer Funktion zeichnen.

Zunächst muss das Modul `matplotlib` mithilfe des Paketverwaltungsprogramms `pip` installiert werden, siehe [Abschnitt A.1](#), »Paketverwaltungsprogramm ›pip‹«. Führen Sie die Installation mit dem folgenden Aufruf durch:

```
pip install matplotlib
```

Es folgt ein erstes Programm, mit dessen Hilfe einige der zahlreichen Möglichkeiten der Bibliothek gezeigt werden. Es werden die beiden mathematischen Funktionen $y_1(x) = x^2$ und $y_2(x) = 0.5x^2$ im Bereich von $x = -5$ bis $x = 5$ gezeichnet, siehe [Abbildung 5.13](#):

```
import matplotlib.pyplot as plt

x = []
y1 = []
y2 = []
i = -5
while i < 5.05:
    x.append(i)
    y1.append(i * i)
    y2.append(0.5 * i * i)
    i += 0.1

plt.plot(x, y1, label="y1(x) = x * x")
plt.plot(x, y2, label="y2(x) = 0.5 * x * x")
plt.title("Funktionen")
plt.legend()
plt.xlabel("x")
```

```
plt.ylabel("y(x)")  
plt.axis([-5, 5, 0, 25])  
plt.grid(axis="both")  
plt.axhline(9, color="red")  
plt.axvline(3, color="green")  
plt.show()
```

Listing 5.33 Datei »matplotlib_graph.py«

Zunächst wird das Modul `pyplot` aus dem Modul `matplotlib` importiert. Für die Aufrufe im Programm wird mithilfe des Schlüsselworts `as` der Alias `plt` erstellt. Darüber ist das Modul in verkürzter Schreibweise erreichbar.

Es werden drei leere Listen erstellt, die in einer Schleife gefüllt werden: Die Liste `x` mit den x-Werten von -5 bis 5 im Abstand 0.1 und die beiden Listen `y1` und `y2` mit den jeweiligen Funktionswerten.

Bei jedem Aufruf der Funktion `plot()` wird ein einzelner Graph für die Werte aus zwei Listen erzeugt. Hier wird die Methode zweimal aufgerufen, einmal für die Werte aus den beiden Listen `x` und `y1` und einmal für die Werte aus den beiden Listen `x` und `y2`. Der optionale Parameter `label` kann zusammen mit der Funktion `legend()` zur Gestaltung der Legende verwendet werden.

Die Funktion `show()` zeichnet die erzeugten Graphen in einem Anwendungsfenster auf dem Bildschirm.

Es können weitere Funktionen zur Gestaltung der Zeichnung genutzt werden:

- `title()` zum Setzen eines Titels oberhalb der Zeichnung
- `legend()` zur Darstellung einer Legende, zusammen mit dem benannten Parameter `label` der Funktion `plot()`
- `xlabel()` und `ylabel()` zum Beschriften der beiden Achsen
- `axis()` zum Festlegen der Grenzwerte für die beiden Achsen mithilfe einer Liste aus vier Werten
- `grid()` zur Darstellung eines Gitternetzes parallel zu den Achsen; mögliche Werte für den benannten Parameter `axis`

sind »x«, »y« oder der Standardwert »both«

- `axhline()` zur Darstellung einer horizontalen Linie an einem bestimmten y-Wert in einer bestimmten Farbe
- `axvline()` zur Darstellung einer vertikalen Linie an einem bestimmten x-Wert in einer bestimmten Farbe

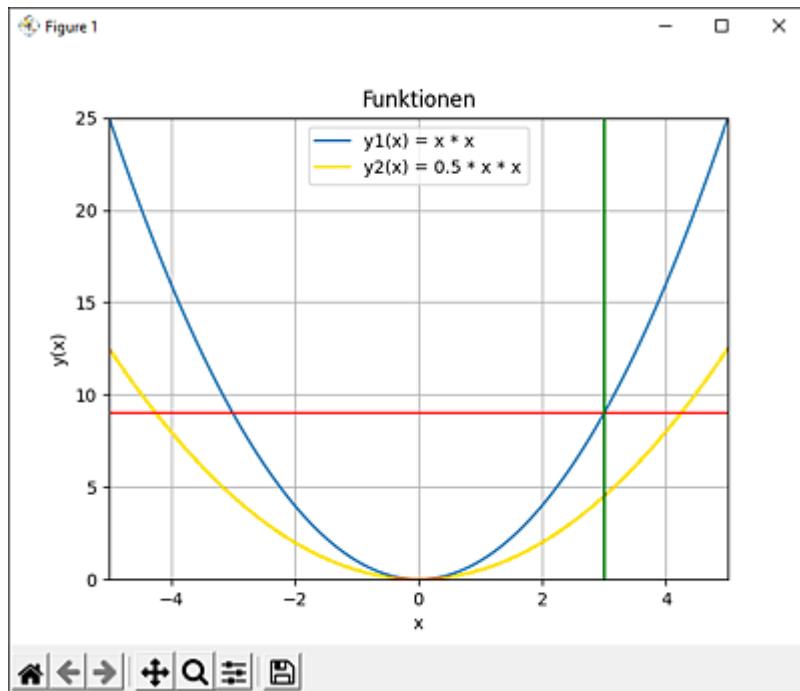


Abbildung 5.13 Funktionen $y_1(x) = x^2$ und $y_2(x) = 0.5x^2$

5.8.2 Mehrere Teilzeichnungen

Mit einem weiteren Programm wird gezeigt, wie mithilfe der Bibliothek `matplotlib` verschiedene Graphen in mehreren Teilzeichnungen dargestellt werden können. Hier handelt es sich um die beiden Funktionen $y_1(x) = \sin(x)$ und $y_2(x) = \cos(x)$ im Bereich von 0 bis 360 Grad, siehe [Abbildung 5.14](#):

```
import matplotlib.pyplot as plt
import math

x = []
y1 = []
y2 = []
i = 0
for i in range(0, 361):
    x.append(i)
    y1.append(math.sin(i))
    y2.append(math.cos(i))
```

```

bm = math.radians(i)
y1.append(math.sin(bm))
y2.append(math.cos(bm))

figur, (py1, py2) = plt.subplots(2, 1)

py1.plot(x, y1)
py1.set_title("Sinus")
py1.set_xlabel("x")
py1.set_ylabel("sin(x)")
py1.axis([0, 360, -1, 1])
py1.grid(axis="both")

py2.plot(x, y2)
py2.set_title("Kosinus")
py2.set_xlabel("x")
py2.set_ylabel("cos(x)")
py2.axis([0, 360, -1, 1])
py2.grid(axis="both")

plt.show()

```

Listing 5.34 Datei »matplotlib_subplot.py«

Es werden insgesamt drei leere Listen erstellt, die in einer Schleife gefüllt werden: Die Liste `x` mit den x-Werten von 0 bis 360 im Abstand 1 und die beiden Listen `y1` und `y2` mit den zugehörigen Funktionswerten, die mithilfe der Funktionen `sin()` und `cos()` ermittelt werden. Die Gradwerte werden zuvor mithilfe der Funktion `radians()` in Bogenmaßwerte umgerechnet.

Die Funktion `subplots()` dient zur Unterteilung einer Zeichnung in mehrere Teilzeichnungen. Diese werden mithilfe der Parameter in einem Raster aus Zeilen und Spalten angeordnet. Hier sind es zwei Zeilen und eine Spalte. Es gibt also zwei Teilzeichnungen. Zurückgeliefert wird eine Referenz auf die gesamte Zeichnung und eine passende Anzahl (hier: zwei) von Referenzen auf die Teilzeichnungen.

Für die Teilzeichnungen werden jeweils einige Funktionen aufgerufen:

- `set_title()` zum Setzen eines Titels oberhalb der Teilzeichnung
- `set_xlabel()` und `set_ylabel()` zum Beschriften der beiden Achsen

- `axis()` zum Festlegen der Grenzwerte für die beiden Achsen
- `grid()` zur Erstellung eines Gitternetzes parallel zu den Achsen

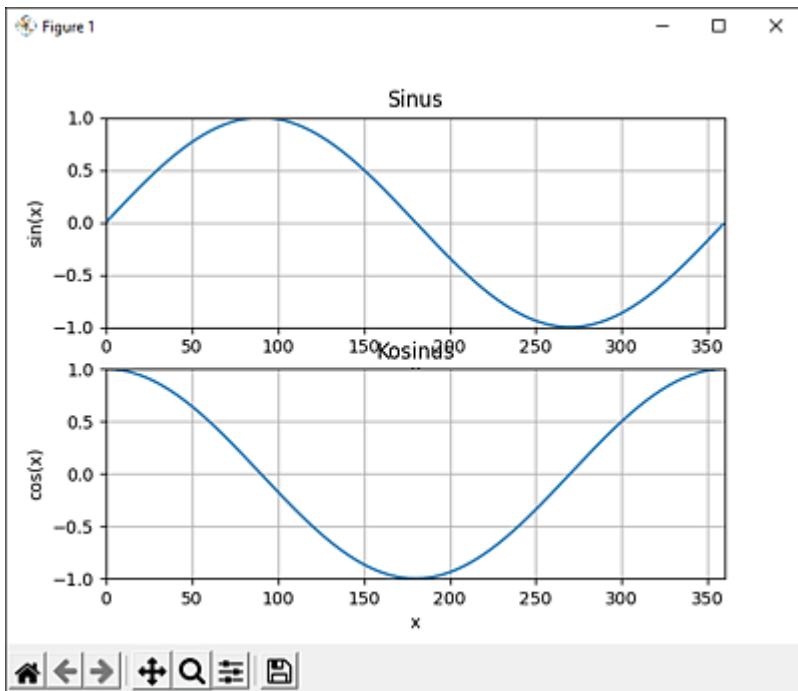


Abbildung 5.14 Funktionen $y_1(x) = \sin(x)$ und $y_2(x) = \cos(x)$

5.8.3 Eindimensionale Arrays und Vektoren

Die Bibliothek `numpy` bietet numerische Funktionen, die zum Beispiel für mathematische Näherungslösungen benötigt werden. Sie erleichtert zudem die Arbeit mit großen, mehrdimensionalen Datenstrukturen, wie sie bei der Rechnung mit Vektoren und Matrizen zum Einsatz kommen.

Nach der Installation des Moduls `matplotlib` ist das Modul `numpy` bereits installiert, da es dort intern benötigt wird. Ansonsten könnten Sie es mit dem Paketverwaltungsprogramm `pip` (siehe [Abschnitt A.1](#)) wie folgt installieren:

```
pip install numpy
```

In einem ersten Programm wird mit eindimensionalen Datenstrukturen gearbeitet, um einige Aufgaben aus der *Vektorrechnung* zu lösen:

```

import numpy as np

a = np.array([ 5,  8, -4] )
print("Eindimensionaler Array = Vektor:", a)

b = a * 2
print("Rechenoperation mit Skalar:", b)

c = np.linspace(0, 30, 4)
print("Array mit äquidistanten Werten:", c)

d = np.sin(np.radians(c))
print("Funktion auf Array anwenden:", d)

e = np.array([ 2,  8,  3] )
f = np.array([ 3,  7, -4] )
g = e + f
print("Vektoraddition:", g)

h = e @ f
print("Skalarprodukt:", h)

i = np.cross(e, f)
print("Kreuzprodukt:", i)

```

Listing 5.35 Datei »numpy_eindim.py«

Das Modul `numpy` wird importiert und gemäß Konvention über den Alias `np` erreichbar gemacht. Mithilfe der Funktion `array()` wird aus einer Liste ein eindimensionaler Array erstellt und ausgegeben. Der Array besitzt drei Elemente und kann daher auch als ein Vektor im dreidimensionalen Raum betrachtet werden.

Rechenoperationen mit Skalaren werden für den gesamten Array ausgeführt. Wie bei der Vektorrechnung werden diese Operationen Element für Element angewendet. Die Vektorrechnung selbst ist nicht Thema dieses Einsteigerbuchs zu Python. Sie finden sie über:

<https://de.wikipedia.org/wiki/Vektor#Rechenoperationen>

Die Funktion `linspace()` erstellt einen Array, der äquidistante Werte enthält, wie sie zum Beispiel für die x-Achse einer Zeichnung mit `matplotlib` benötigt werden. Mit den beiden ersten Parametern werden der erste und der letzte Wert des

Arrays festgelegt, mit dem dritten Parameter die Anzahl der Werte im Array.

Das Modul `numpy` bietet Winkelfunktionen, die effektiver und genauer arbeiten als die entsprechenden Funktionen aus dem Modul `math`. Diese Funktionen aus dem Modul `numpy` werden ebenfalls für den gesamten Array ausgeführt, Element für Element.

Die folgenden Berechnungen mit Vektoren werden gemäß den Regeln der Vektorrechnung für Vektoren derselben Größe ausgeführt:

- *Vektoraddition*
- *Skalarprodukt*, mithilfe des Operators `@`
- *Kreuzprodukt*, mithilfe der Funktion `cross()`

Die Ausgabe des Programms:

```
Eindimensionaler Array = Vektor: [ 5 8 -4]
Rechenoperation mit Skalar: [10 16 -8]
Array mit äquidistanten Werten: [ 0. 10. 20. 30.]
Funktion auf Array anwenden: [0. 0.17364818 0.34202014 0.5]
Vektoraddition: [ 5 15 -1]
Skalarprodukt: 50
Kreuzprodukt: [-53 17 -10]
```

5.8.4 Mehrdimensionale Arrays und Matrizen

In diesem Abschnitt geht es um mehrdimensionale Arrays und Matrizen, die mithilfe der Bibliothek `numpy` erstellt und bearbeitet werden:

- Arrays können eine beliebige Anzahl von Dimensionen besitzen. Als Beispiel wird ein dreidimensionaler Array erzeugt.
- Matrizen können nur zweidimensional sein. Mit ihnen werden hier einige Aufgaben aus der *Matrizenrechnung* gelöst.

Zunächst das Programm:

```

import numpy as np

a = np.array([[ [ 2,  8],[ 3,  7]],[[ 5,  4],[ 9,  1]]])
b = a * 2
print("Operator und Skalar auf mehrdimensionalen Array anwenden:")
print(b)
print("Form des Arrays:", b.shape)
print()

c = np.matrix([[ 5,  2,  3],[ 8,  4,  5]])
print("Matrix c:")
print(c)
print("Form der Matrix:", c.shape)
print()

d = c * 2
print("Rechenoperation mit Skalar:")
print(d)
print()

e = np.matrix([[ 2, -4,  7],[-1,  9,  3]])
print("Matrix e:")
print(e)
print("Form der Matrix:", e.shape)
print()

f = c + e
print("Matrizenaddition c + e:")
print(f)
print()

g = np.matrix([[ 2,  3],[ 4,  5],[ 2,  6]])
print("Matrix g:")
print(g)
print("Form der Matrix:", g.shape)
print()

h = c * g
print("Matrizenmultiplikation c * g:")
print(h)

```

Listing 5.36 Datei »numpy_mehrdim.py«

Als Erstes wird mithilfe der Funktion `array()` ein dreidimensionaler Array aus einer dreidimensionalen Liste erstellt und ausgegeben. Auf dem Bildschirm können nur zwei Dimensionen anschaulich dargestellt werden. Zur Verdeutlichung der nächsthöheren Dimension werden daher automatisch Leerzeilen eingefügt.

Mit dem gesamten dreidimensionalen Array wird eine Rechenoperation mit einem Skalar ausgeführt, wiederum

Element für Element.

Die Eigenschaft `shape` liefert ein Tupel mit der Größe eines Arrays in den einzelnen Dimensionen. Der vorliegende Array hat $2 \times 2 \times 2$ Elemente.

Es folgt die Erstellung der Matrix `c` mit zwei Zeilen und drei Spalten mithilfe der Funktion `matrix()`. Sie besitzt ebenso die Eigenschaft `shape`.

Rechenoperationen mit Skalaren werden für die gesamte Matrix ausgeführt. Wie bei der Matrizenrechnung werden auch diese Operationen Element für Element angewendet. Die Matrizenrechnung selbst ist nicht Thema dieses Einsteigerbuchs zu Python. Sie finden sie über [https://de.wikipedia.org/wiki/Matrix_\(Mathematik\)#Addition_und_Multiplikation](https://de.wikipedia.org/wiki/Matrix_(Mathematik)#Addition_und_Multiplikation).

Es wird die Matrix `e` erzeugt. Sie hat mit zwei Zeilen und drei Spalten dieselbe Form wie die Matrix `c`. Daher können diese beiden Matrizen gemäß den Regeln der *Matrizenaddition* addiert werden.

Anschließend wird die Matrix `g` mit drei Zeilen und zwei Spalten erstellt. Damit hat sie die passende Form, um mit der Matrix `c` gemäß den Regeln der *Matrizenmultiplikation* multipliziert werden zu können.

Die Ausgabe des Programms:

```
Operator und Skalar auf mehrdimensionalen Array anwenden:  
[[[ 4 16]  
 [ 6 14]]  
  
 [[10  8]  
 [18  2]]]  
Form des Arrays: (2, 2, 2)  
  
Matrix c:  
[[5 2 3]  
 [8 4 5]]  
Form der Matrix: (2, 3)  
  
Rechenoperation mit Skalar:
```

```

[[10  4  6]
 [16  8 10]]

Matrix e:
[[ 2 -4  7]
 [-1  9  3]]
Form der Matrix: (2, 3)

Matrizenaddition c + e:
[[ 7 -2 10]
 [ 7 13  8]]

Matrix g:
[[2 3]
 [4 5]
 [2 6]]
Form der Matrix: (3, 2)

Matrizenmultiplikation c * g:
[[24 43]
 [42 74]]

```

5.8.5 Signalverarbeitung

Die Bibliothek `scipy` bietet zahlreiche Funktionen für wissenschaftliche Berechnungen. Sie basiert auf der Bibliothek `numpy` und enthält unter anderem das Modul `io` zum Lesen und Schreiben von Werten aus Dateien. Darin befindet sich wiederum das Modul `wavfile` mit den Funktionen `read()` und `write()` zum Lesen und Schreiben von digitalen Signalen aus WAV-Dateien.

Sie installieren das Modul `scipy` mit dem Paketverwaltungsprogramm `pip` (siehe [Abschnitt A.1](#)) wie folgt:

```
pip install scipy
```

Im nachfolgenden Programm werden die digitalen Audiosignale aus einer WAV-Datei gelesen und, nach linkem und rechtem Kanal getrennt, in einer Zeichnung dargestellt, siehe [Abbildung 5.15](#). Zusätzlich werden einige Informationen zu der WAV-Datei ausgegeben:

```

import scipy.io.wavfile as wf
import matplotlib.pyplot as plt
import numpy as np

abtastrate, werte = wf.read("tada.wav")
print(f"Abtastrate: {abtastrate} Werte / sec.")

```

```

print("Anzahl Werte:", werte.shape[ 0 ] )
length = werte.shape[ 0 ] / abtastrate
print(f"Länge: {round(length,2)} sec.")

zeit = np.linspace(0, length, werte.shape[ 0 ] )
plt.plot(zeit, werte[ :, 0 ], label="Links")
plt.plot(zeit, werte[ :, 1 ], label="Rechts")
plt.legend()
plt.xlabel("sec.")
plt.ylabel("Amplitude")
plt.show()

```

Listing 5.37 Datei »scipy_wav.py«

Die zusätzliche Ausgabe sieht wie folgt aus:

```

Abtastrate: 44100 Werte / sec.
Anzahl Werte: 71296
Länge: 1.62 sec.

```

Die Funktion `read()` liefert ein Tupel mit zwei Elementen:

- Das erste Element enthält die Abtastrate. Sie entspricht der Frequenz, mit der das ursprünglich analoge Audiosignal digital abgetastet wurde. Sie wird in der Einheit Hz oder Werte pro Sekunde gemessen. Eine übliche Abtastrate ist 44.1 KHz, also 44100 Werte pro Sekunde.
- Das zweite Element enthält einen zweidimensionalen `numpy`-Array mit den gespeicherten Werten aus der WAV-Datei für die Amplituden der beiden Kanäle.

Die Anzahl der gespeicherten Werte lässt sich mithilfe der Eigenschaft `shape` ermitteln. Aus der Abtastrate und der Anzahl der Werte lässt sich die zeitliche Länge des Audiosignals in Sekunden berechnen.

Mithilfe der Funktion `linspace()` wird eine Liste für die x-Achse der Zeichnung erstellt. Auf der x-Achse wird die zeitliche Länge des Audiosignals aufgetragen. Für jeden Kanal des Audiosignals wird die Funktion `plot()` aufgerufen. Zusätzlich werden ein Titel, eine Legende und die Achsenbeschriftungen erstellt.

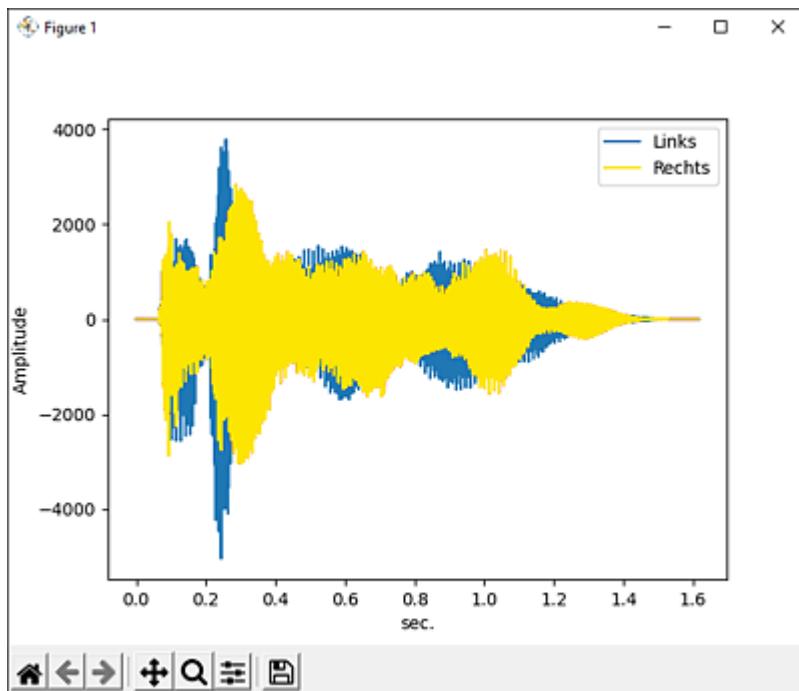


Abbildung 5.15 Darstellung von digitalen Audiosignalen

5.8.6 Statistikfunktionen

Die Statistik dient dazu, große Mengen von Werten zu analysieren und bestimmte repräsentative Informationen über diese Werte zu ermitteln. Dabei kann es sich zum Beispiel um Werte von Umfragen oder anderen Stichproben handeln. Python bietet seit Version 3.4 im Modul `statistics` einige hilfreiche Funktionen für Iterables aus diesen Werten. Mit Python 3.6 und Python 3.8 sind noch einige Funktionen hinzugekommen.

Ein Beispielprogramm:

```
import statistics as sta

pa = [ 5, 2, 4, 17]
print("Liste:", pa)
print("Arithmetischer Mittelwert:", sta.mean(pa))
print("Geometrischer Mittelwert:", sta.geometric_mean(pa))
print("Harmonischer Mittelwert:", sta.harmonic_mean(pa))
print("Median:", sta.median(pa))
print("Unterer Median:", sta.median_low(pa))
print("Oberer Median:", sta.median_high(pa))
print()

pb = [ 5, 2, 4, 17, 3]
print("Liste:", pb)
```

```

print("Median:", sta.median(pb))
print("Unterer Median:", sta.median_low(pb))
print("Oberer Median:", sta.median_high(pb))
print()

pc = [ 3, 5, 5, 12, 17, 17]
print("Modus:", sta.mode(pc))
print("Multimodus:", sta.multimode(pc))
print()

print("Arithm. Mittelwert aus Tupel:", sta.mean((5, 2, 4, 17)))
pe = {'D': 5, 'NL': 2, 'CH': 4, 'F': 17}
print("Arithm. Mittelwert aus Dictionary:", sta.mean(pe.values()))

```

Listing 5.38 Datei »statistik.py«

Es wird die folgende Ausgabe erzeugt:

```

Liste: [5, 2, 4, 17]
Arithmetisches Mittelwert: 7
Geometrischer Mittelwert: 5.1065457621381
Harmonischer Mittelwert: 3.9650145772594754
Median: 4.5
Unterer Median: 4
Oberer Median: 5

Liste: [5, 2, 4, 17, 3]
Median: 4
Unterer Median: 4
Oberer Median: 4

Modus: 5
Multimodus: [5, 17]

Arithm. Mittelwert aus Tupel: 7
Arithm. Mittelwert aus Dictionary: 7

```

Zunächst wird das Modul `statistics` importiert. Für die Aufrufe im Programm wird es mithilfe des Schlüsselworts `as` kurz als `sta` erreichbar.

Die Funktion `mean()` liefert den arithmetischen Mittelwert, also die Summe der Werte, geteilt durch ihre Anzahl. Hier entspricht das: $(5 + 2 + 4 + 17) / 4$.

Seit Python 3.8 gibt es die Funktion `geometric_mean()`. Sie liefert den geometrischen Mittelwert. Er entspricht der n-ten Wurzel des Produkts von n Werten. Hier entspricht das der vierten Wurzel aus $(5 \times 2 \times 4 \times 17)$.

Seit Python 3.6 gibt es die Funktion `harmonic_mean()`. Sie liefert den harmonischen Mittelwert. Er entspricht der Anzahl der Werte, geteilt durch die Summe ihrer Kehrwerte. Hier entspricht das: $4 / (1/5 + 1/2 + 1/4 + 1/17)$.

Weitere Anwendungsbeispiele für die verschiedenen Mittelwerte finden Sie unter <https://de.wikipedia.org/wiki/Mittelwert>.

Die Funktion `median()` liefert den Median. Dieser wird auch Zentralwert genannt, da er im Zentrum der Zahlenmenge steht: Es gibt genauso viele Werte, die größer sind als der Median, wie Werte, die kleiner sind als der Median. Bei einer ungeraden Menge von Werten handelt es sich beim Median um das Element in der Mitte der Zahlenmenge. Bei einer geraden Menge von Werten handelt es sich um den arithmetischen Mittelwert der beiden Elementen in der Mitte der Zahlenmenge.

Die Funktionen `median_low()` und `median_high()` liefern den unteren bzw. oberen Median. Dabei handelt es sich in jedem Fall um Elemente aus der Zahlenmenge. Bei einer ungeraden Menge von Werten ist es das Element in der Mitte der Zahlenmenge. Bei einer geraden Menge von Werten handelt es sich um die beiden Elemente in der Mitte der Zahlenmenge.

Die Funktion `mode()` liefert denjenigen Wert, der in der Zahlenmenge am häufigsten vorkommt. Besitzen mehrere Werte die größte Häufigkeit, wird derjenige Wert genannt, der als erster vorkommt. Seit Python 3.8 gibt es die Funktion `multimode()`. Sie liefert eine Liste mit allen Werten, die die größte Häufigkeit besitzen.

Die untersuchte Zahlenmenge kann auch aus einem Tupel oder, mithilfe der Methode `values()`, aus der Werte-View eines Dictionarys stammen.

5.9 Eigene Module

Es wurde bereits mit Funktionen aus Standardmodulen wie zum Beispiel `random`, `math`, `fractions`, `copy` und `statistics` oder zusätzlich installierten Modulen wie `matplotlib`, `numpy` und `scipy` gearbeitet.

Darüber hinaus können Sie eigene Module definieren und importieren. Das ist nützlich, wenn Sie feststellen, dass bestimmte Funktionen von verschiedenen Programmen benötigt werden. Die Erstellung und Nutzung von eigenen Modulen ist in Python sehr einfach und wird in diesem Abschnitt erläutert.

5.9.1 Eigene Module erzeugen

Zur Erzeugung eines Moduls speichern Sie die gewünschte Funktion einfach in einer eigenen Datei. Der Name der Datei ist zugleich der Name des Moduls. Die Funktion erstellen Sie wie gewohnt:

```
def quadrat(x):
    return x * x
```

Listing 5.39 Datei »modul_neu.py«

Anschließend können Sie die Funktion nach einem Import in jedem Programm nutzen. Dazu gibt es verschiedene Möglichkeiten, die ich im Folgenden beschreibe.

5.9.2 Standard-Import eines Moduls

Sie importieren das Modul wie gewohnt:

```
import modul_neu
print("Quadrat:", modul_neu.quadrat(3))
```

Listing 5.40 Datei »modul_verwenden.py«

Alle Funktionen des Moduls `modul_neu`, also der Datei `modul_neu.py`, werden mit der Anweisung `import` zugänglich gemacht. Die Funktion `quadrat()` wird wie folgt aufgerufen: [Modulname].[Funktionsname].

5.9.3 Import eines Moduls mit Umbenennung

Hat ein Standardmodul oder ein eigenes Modul einen langen, unhandlichen Namen, können Sie ihm mithilfe von `as` einen Aliasnamen geben:

```
import modul_neu as mneu
print("Quadrat:", mneu.quadrat(3))
```

Listing 5.41 Datei »modul_as.py«

Sie können hier auf das Modul `modul_neu` mithilfe von `mneu` zugreifen.

5.9.4 Import von Funktionen

Mithilfe des Schlüsselworts `from` können Sie einzelne Funktion importieren:

```
from modul_neu import quadrat
print("Quadrat:", quadrat(3))
```

Listing 5.42 Datei »modul_from.py«

Die Funktion `quadrat()` wird mithilfe der Anweisung `from` aus dem Modul `modul_neu` importiert. Sie können sie anschließend ohne den Modulnamen aufrufen, so als ob sie in der gleichen Datei definiert wäre. Die anderen Funktionen des Moduls stehen nicht zur Verfügung.

Im Unterschied dazu können Sie auch alle Funktionen aus dem Modul `modul_neu` auf einmal importieren. Die Anweisung lautet dann:

```
from modul_neu import *
```

Der Import mithilfe des Schlüsselworts `from` wird nur der Vollständigkeit halber erwähnt. Diese Möglichkeit wird nicht mehr empfohlen und sollte auch nur außerhalb von Funktionen genutzt werden. Sie erweist sich außerdem als ungünstig, falls Sie mehrere Funktionen mit gleichem Namen aus unterschiedlichen Modulen importieren möchten.

Hinweis

In den genannten Beispielen wird davon ausgegangen, dass sich die Module im selben Verzeichnis wie die Programme befinden, die sie nutzen.

Übung »u_modul«

Schreiben Sie das Programm aus Übung »u_rueckgabewert« um: Die Funktion `steuer()` soll in die Datei `u_modul_finanz.py` ausgelagert werden. Das Hauptprogramm in Datei `u_modul.py` soll die Funktion aus dieser Datei importieren und nutzen.

5.10 Parameter der Kommandozeile

Ein Python-Programm kann bekanntlich von der Kommandozeile des Betriebssystems aus aufgerufen werden. In [Abschnitt 2.3.2](#), »Ausführen unter Windows«, und in [Abschnitt 2.3.3](#), »Ausführen unter Ubuntu Linux und unter macOS«, wird beschrieben, wie Sie zur Kommandozeile bzw. zum Terminal gelangen.

Der Aufruf ähnelt dem Aufruf einer Funktion mit Parametern. Die einzelnen Parameter werden durch Leerzeichen voneinander getrennt. Sie stehen innerhalb des Programms in der Liste `argv` aus dem Modul `sys` zur Verfügung. Die Anzahl der Parameter lässt sich mithilfe der eingebauten Funktion `len()` ermitteln.

Das nachfolgende Programm kann mit einer beliebigen Anzahl von Parametern aufgerufen werden. Im Programm wird die Summe derjenigen Parameter ermittelt, die gültige Zahlenwerte enthalten:

```
import sys
print("Parameter:", sys.argv)
print("Anzahl:", len(sys.argv))

summe = 0
for i in sys.argv[1:]:
    try:
        summe += float(i)
    except:
        print(f" Fehler bei Parameter {i}")
print("Summe:", summe)
```

Listing 5.43 Datei »kommando.py«

Nach dem folgenden Aufruf von der Kommandozeile aus:

```
python kommando.py 12 abc -3.5
```

ergibt sich die Ausgabe:

```
Parameter: ['kommando.py', '12', 'abc', '-3.5']
Anzahl: 4
 Fehler bei Parameter abc
Summe: 8.5
```

Zunächst wird die Liste der Parameter zusammen mit ihrer Anzahl ausgegeben. Das erste Element der Liste `sys.argv` ist der Name des Programms. Die weiteren Elemente sind die Nutz-Parameter. Sie können mithilfe des Slice `sys.argv[1:]` und einer `for`-Schleife durchlaufen werden. Kann einer der Nutz-Parameter nicht umgewandelt werden, erfolgt eine Fehlermeldung.

5.11 Programm »Bruchtraining«

Das Programm in diesem Abschnitt enthält ein Trainingsprogramm zur Bruchrechnung mit drei Schwierigkeitsgraden. Sowohl der Zähler als auch der Nenner der vorkommenden Brüche können positiv oder negativ sein.

Es handelt sich nur um ein größeres Beispiel und kann zunächst übergangen werden. Es verdeutlicht das Zusammenspiel vieler bekannter Elemente, wie zum Beispiel mehrfache Verzweigungen, Schleifen, Ausnahmebehandlung, Tupel und Funktionen mit mehreren Rückgabewerten.

5.11.1 Der Ablauf des Programms

Zur Verdeutlichung wird zunächst der Ablauf des Programms mithilfe von einigen Beispieleingaben erläutert:

```
Ihre Wahl: 1=Leicht, 2=Mittel, 3=Schwer, 0=Ende
1
Ganze Zahl berechnen: -63 / 9
Ergebnis: -7
Ihre Wahl: 1=Leicht, 2=Mittel, 3=Schwer, 0=Ende
2
Bruch kürzen: -30 / -70
Ergebnis: 3 / 7
Ihre Wahl: 1=Leicht, 2=Mittel, 3=Schwer, 0=Ende
3
Ergebnis-Bruch berechnen: -21/14 / -15/25
Ergebnis: 5 / 2
Ihre Wahl: 1=Leicht, 2=Mittel, 3=Schwer, 0=Ende
0
```

Nach dem Start des Programms erscheint ein Menü, mit dem eine leichte Aufgabe (Eingabe: 1), eine mittelschwere Aufgabe (Eingabe: 2) oder eine schwere Aufgabe (Eingabe: 3) ausgewählt werden kann. Nach dem Bearbeiten einer Aufgabe erscheint wiederum das Menü. Hier kann entweder eine weitere Aufgabe ausgewählt oder das Programm beendet werden (Eingabe: 0).

Nach der Auswahl einer leichten Aufgabe wird ein Bruch angezeigt, der eine ganze Zahl ergibt. Der Bruchstrich wird durch einen Divisionsstrich zwischen Zähler und Nenner dargestellt. Der Benutzer rechnet, im Kopf oder auf dem Papier, die ganze Zahl aus, die das Ergebnis der Aufgabe darstellt. Nach dem Betätigen der Taste erscheint das richtige Ergebnis, mit dem der Benutzer sein eigenes Ergebnis vergleicht. Stellt der Benutzer fest, dass er zum Beispiel 20 leichte Aufgaben richtig lösen konnte, kann er als Nächstes eine mittelschwere Aufgabe auswählen.

Nach der Auswahl einer mittelschweren Aufgabe wird ein Bruch angezeigt, den der Benutzer kürzen soll, und zwar auf den kleinstmöglichen Bruch. Der Benutzer rechnet, im Kopf oder auf dem Papier, den gekürzten Bruch aus. Nach dem Betätigen der Taste erscheint wiederum das richtige Ergebnis, mit dem der Benutzer sein eigenes Ergebnis vergleicht. Stellt der Benutzer fest, dass er zum Beispiel 20 mittelschwere Aufgaben richtig lösen konnte, kann er als Nächstes eine schwere Aufgabe auswählen.

Nach der Auswahl einer schweren Aufgabe werden zwei Brüche angezeigt, die der Benutzer entweder miteinander addieren, subtrahieren, dividieren oder multiplizieren soll. Das Ergebnis soll er anschließend kürzen. Der Benutzer rechnet auch hier, im Kopf oder auf dem Papier, den gekürzten Bruch aus. Nach dem Betätigen der Taste erscheint auch hier das richtige Ergebnis, mit dem der Benutzer sein eigenes Ergebnis vergleicht.

5.11.2 Hauptprogramm

Das Programm wird in einzelnen Teilen erläutert. Zunächst folgt das Hauptprogramm, das am Ende der Datei steht:

```
# Programm
while True:
    while True:
        print("Ihre Wahl: 1=Leicht, 2=Mittel, 3=Schwer, 0=Ende")
        try:
```

```

wahl = int(input())
if wahl < 0 or wahl > 3:
    raise
else:
    break
except:
    print("Bitte nur 0, 1, 2 oder 3 eingeben")

if wahl == 0:
    break
elif wahl == 1:
    leicht()
elif wahl == 2:
    mittel()
else:
    schwer()

```

Listing 5.44 Datei »bruchtraining.py«, Hauptprogramm

Das Benutzermenü und die Eingabe sind in eine äußere Endlosschleife eingebettet, die nur mit der Eingabe von 0 verlassen werden kann.

Die Anzeige des Menüs ist in eine innere Endlosschleife eingebettet. Nimmt der Benutzer eine Eingabe vor, die nicht in eine ganze Zahl umgewandelt werden kann, tritt eine Ausnahme auf. Gibt der Benutzer eine Zahl ein, die keinem Wert im Menü entspricht, wird mithilfe des Schlüsselworts `raise` eine eigene Ausnahme erzeugt. Nach einer Ausnahme erscheint eine Fehlermeldung, und es wird wiederum das Menü angezeigt

Gibt der Benutzer eine gültige Zahl ein, wird eine der drei Funktionen `leicht()`, `mittel()` oder `schwer()` aufgerufen, oder es wird die äußere Endlosschleife verlassen.

5.11.3 Eine leichte Aufgabe

Es folgt die Funktion `leicht()` zur Erstellung einer leichten Aufgabe. Das gesamte Programm benötigt zudem einen initialisierten Zufallsgenerator:

```

import random
random.seed
...
def leicht():
    faktor = random.randint(-10, 10)

```

```

n = 0
while n == 0:
    n = random.randint(-10, 10)
z = faktor * n
print(f"Ganze Zahl berechnen: {z}/{n}")
input()
print("Ergebnis:", faktor)

```

Listing 5.45 Datei »bruchtraining.py«, Funktion »leicht()«

Der Zufallsgenerator liefert ganze Zahlen für einen Faktor und für den Nenner des Bruchs. Der Zähler des Bruchs entspricht dem Nenner, multipliziert mit dem Faktor. Ein Beispiel: Der Faktor 3 und der Nenner 5 ergeben den Zähler 15. Für den Benutzer wird also die Aufgabe $15 / 5$ ermittelt und ausgegeben. Als Ergebnis soll er den Faktor 3 berechnen.

Aufgrund des Bereichs von -10 bis 10 kann der Zufallsgenerator den Wert 0 liefern. Mithilfe der `while`-Schleife wird dafür gesorgt, dass ein neuer Nenner ermittelt wird, da eine Division durch 0 mathematisch nicht erlaubt ist.

5.11.4 Eine mittelschwere Aufgabe

Es folgt die Funktion `mittel()` zur Erstellung einer mittelschweren Aufgabe. Sie benötigt das Tupel `prob` und die Funktionen `produkt()` und `ausgabe()`:

```

...
prob = 2, 2, 2, 2, 3, 3, 3, 5, 5, 7
...
def produkt(anzahl):
    wert = 1
    for i in range(anzahl):
        wert *= random.choice(prob)
    if random.randint(0,1) == 0:
        return wert
    else:
        return -wert

def ausgabe(z, n):
    ggT = math.gcd(z, n)
    z = int(z / ggT)
    n = int(n / ggT)
    if n < 0:
        z *= -1
        n *= -1
    print("Ergebnis:", z if n == 1 else f"{z}/{n}")

```

```

...
def mittel():
    z = produkt(3)
    n = produkt(3)
    print(f"Bruch kürzen: {z}/{n}")
    input()
    ausgabe(z, n)

```

Listing 5.46 Datei »bruchtraining.py«, Funktion »mittel()«

Die Variablen `z` und `n` enthalten den Zähler und den Nenner des gekürzten Bruchs. Mit diesem Bruch vergleicht der Benutzer das Ergebnis seiner eigenen Berechnung.

Der Zähler und der Nenner eines Bruchs werden jeweils mithilfe der Funktion `produkt()` erzeugt. Sie liefert einen Wert, der mithilfe des Zufallsgenerators erzeugt wird. Die `for`-Schleife wird dreimal durchlaufen. Bei jedem Durchlauf liefert die Funktion `choice()` aus dem Modul `random` einen zufälligen Wert aus dem Tupel `prob`. Es enthält in unterschiedlicher Wahrscheinlichkeit (englisch: *probability*) die Werte 2, 3, 5 und 7. Damit wird dafür gesorgt, dass die Aufgaben eher kleinere als größere Zahlen enthalten.

Die drei Werte werden miteinander multipliziert. Anschließend wird, wiederum mithilfe des Zufallsgenerators, das Vorzeichen des ermittelten Produkts gesetzt. Liefert der Zufallsgenerator eine 0, bleibt das Produkt positiv. Liefert er eine 1, wird das Produkt mit -1 multipliziert.

Zur Anzeige des Ergebnisses wird die Funktion `ausgabe()` aufgerufen. Darin werden Zähler und Nenner zunächst gekürzt, indem sie jeweils durch ihren größten gemeinsamen Teiler dividiert werden.

Ist das Ergebnis ein negativer Bruch, soll das Vorzeichen im Zähler stehen. Sind sowohl Zähler als auch Nenner negativ, soll der Bruch in den entsprechenden positiven Bruch umgewandelt werden. Ist also der Nenner negativ, werden Zähler und Nenner mit -1 multipliziert. Zwei Beispiele: Aus $4 / -3$ wird $-4 / 3$, aus $-4 / -3$ wird $4 / 3$.

Hat der Nenner den Wert 1, wird nur der Zähler ausgegeben.

5.11.5 Eine schwere Aufgabe

Es folgt die Funktion `schwer()` zum Erstellen einer schweren Aufgabe. Sie benötigt zusätzlich das Tupel `ops`:

```
...
ops = '+', '- ', '*' , '/'

...
def schwer():
    az = produkt(2)
    an = produkt(2)
    bz = produkt(2)
    bn = produkt(2)
    op = random.choice(ops)
    print(f"Ergebnis-Bruch berechnen: {az}/{an} {op} {bz}/{bn}")

    if op == "+":
        z = az * bn + an * bz
        n = an * bn
    elif op == "-":
        z = az * bn - an * bz
        n = an * bn
    elif op == "*":
        z = az * bz
        n = an * bn
    else:
        z = az * bn
        n = an * bz

    input()
    ausgabe(z, n)
```

Listing 5.47 Datei »bruchtraining.py«, Funktion »schwer()«

Das Tupel `ops` enthält als Elemente vier Zeichen, und zwar die Operatoren für die vier verschiedenen Berechnungen.

Der Zähler und Nenner der beiden Brüche, nennen wir sie A und B, werden mithilfe der Funktion `produkt()` erzeugt. Der Bruch A hat den Zähler `az` und den Nenner `an`, der Bruch B den Zähler `bz` und den Nenner `bn`. Die vier Werte sind jeweils das Produkt aus zwei zufälligen Zahlen.

Der Operator für die Aufgabe wird ebenfalls per Zufallsgenerator erzeugt. Anschließend wird die Aufgabe in der Form A <Operator> B angezeigt.

Mithilfe einer mehrfachen Verzweigung wird das Ergebnis in Abhängigkeit vom Operator gemäß den Regeln der Bruchrechnung ermittelt. Auch hier enthalten die Variablen z und n den Zähler und den Nenner des Ergebnisses. Zur Anzeige des Ergebnisses wird die Funktion `ausgabe()` aufgerufen.

6 Objektorientierte Programmierung

Dieses Kapitel bietet Ihnen eine Einführung in die objektorientierte Programmierung (OOP). Es erläutert die Möglichkeiten zur Erzeugung einer Klassenhierarchie, mit der Sie große Softwareprojekte anhand von Klassen, Objekten, Eigenschaften, Methoden und des Prinzips der Vererbung bearbeiten können.

6.1 Was ist OOP?

Die *objektorientierte Programmierung* (OOP) bietet zusätzliche Möglichkeiten zum verbesserten Aufbau und zur vereinfachten Wartung und Erweiterung von Programmen. Diese Vorteile erschließen sich besonders in großen Programmierprojekten.

Bei der objektorientierten Programmierung werden Klassen erzeugt, in denen die Eigenschaften und Methoden von Objekten festgelegt werden. Methoden sind Funktionen, die nur auf Objekte der betreffenden Klasse angewendet werden können. Die Eigenschaften und die Methoden bilden zusammen die *Member* einer Klasse.

Sie können verschiedene Objekte dieser Klassen erzeugen, den Eigenschaften unterschiedliche Werte zuweisen und die Methoden auf die Objekte anwenden. Die Definitionen aus der Klasse und die zugewiesenen Werte begleiten die Objekte über ihren gesamten *Lebensweg* während der Dauer des Programms.

Ein Beispiel: Es wird die eigene Klasse `Fahrzeug` erschaffen, in der die Eigenschaften und die Methoden von Fahrzeugen bestimmt werden. Ein Fahrzeug hat unter anderem die Eigenschaften Bezeichnung, Geschwindigkeit und Fahrtrichtung. Außerdem kann man ein Fahrzeug beschleunigen und lenken. Innerhalb des Programms können viele unterschiedliche Fahrzeuge erschaffen und eingesetzt werden.

Klassen können ihre Eigenschaften und Methoden außerdem vererben. Eine solche Klasse fungiert als *Basisklasse*, ihre Erben nennt man *abgeleitete Klassen*. Damit kann die Definition von ähnlichen Objekten, die über eine Reihe von gemeinsamen Eigenschaften und Methoden verfügen, vereinfacht werden.

Ein Beispiel: Es werden die Klassen `PKW` und `LKW` geschaffen. Beide Klassen sind von der Basisklasse `Fahrzeug` abgeleitet und erben alle Eigenschaften und Methoden dieser Klasse. Zusätzlich verfügen sie über eigene Eigenschaften und Methoden, die bei der jeweiligen Klasse besonders wichtig sind. Ein `PKW` hat etwa eine bestimmte Anzahl von `Insassen`, und man kann `einstiegen` und `aussteigen`. Ein `LKW` hat eine `Ladung`, man kann ihn `beladen` und `entladen`.

Hinweis

Die in diesem Kapitel dargestellten Programme sind ein Kompromiss, denn die Vorteile der objektorientierten Programmierung (OOP) kommen erst bei größeren Programmierprojekten zum Tragen.

Bei einem kleineren Projekt ist die Frage berechtigt, warum für dieses einfache Ergebnis ein komplexes Programm geschrieben werden sollte. Anhand der hier vorgestellten Programme können Sie sich aber die Prinzipien der OOP erschließen, ohne den Überblick zu verlieren.

6.2 Klassen, Objekte und eigene Methoden

Als Beispiel wird die eigene Klasse `Fahrzeug` definiert. Zunächst verfügt ein Objekt dieser Klasse nur über die Eigenschaft `geschwindigkeit` und die Methoden `beschleunigen()` und `ausgabe()`. Die Methode `ausgabe()` informiert den Anwender über den aktuellen Zustand des jeweiligen Fahrzeug-Objekts.

Die Definition der eigenen Klasse sieht wie folgt aus:

```
class Fahrzeug:  
    geschwindigkeit = 0  
    def beschleunigen(self, wert):  
        self.geschwindigkeit += wert  
    def ausgabe(self):  
        print("Geschwindigkeit:", self.geschwindigkeit)
```

Listing 6.1 Datei »oop_klasse.py«, Klassendefinition

Die Definition der eigenen Klasse wird eingeleitet vom Schlüsselwort `class`, gefolgt vom Namen der Klasse und einem Doppelpunkt. Der Name einer eigenen Klasse sollte gemäß Konvention mit einem großen Buchstaben beginnen. Die weiteren Zeilen der Definition werden eingerückt. Die Eigenschaft `geschwindigkeit` wird definiert und auf den Wert `0` gesetzt.

Methoden werden wie Funktionen mithilfe des Schlüsselworts `def` definiert. Sie haben mindestens einen Parameter, nämlich eine Referenz auf das Objekt selbst. Gemäß Konvention wird diese Referenz `self` genannt – Sie können aber auch einen anderen Namen wählen.

Die Methode `beschleunigen()` hat insgesamt zwei Parameter: Der erste Parameter ist das Objekt selbst. Der zweite Parameter ist der Wert für die Änderung der Geschwindigkeit. Innerhalb der Methode wird dieser Wert genutzt, um den Wert der Eigenschaft des Objekts zu ändern. Die Methode `ausgabe()` hat nur einen

Parameter: das Objekt selbst. Sie dient zur Ausgabe der Eigenschaft(en) des Objekts.

Bisher enthielt das Programm nur eine Klassendefinition, es führte noch nichts aus. Das Hauptprogramm hat folgendes Aussehen:

```
volvo = Fahrzeug()  
opel = Fahrzeug()  
  
volvo.ausgabe()  
volvo.beschleunigen(20)  
volvo.ausgabe()  
  
opel.ausgabe()
```

Listing 6.2 Datei »oop_klasse.py«, Hauptprogramm

Das Programm erzeugt die Ausgabe:

```
Geschwindigkeit: 0  
Geschwindigkeit: 20  
Geschwindigkeit: 0
```

Im Hauptprogramm werden zunächst zwei Objekte der Klasse `Fahrzeug` erzeugt, hier mit den Namen `volvo` und `opel`. Diesen Vorgang nennt man auch: Instanzen einer Klasse erzeugen.

Beim anschließenden Aufruf der Methoden ist zu beachten, dass das Objekt selbst nicht als Parameter angegeben wird. Eine Methode bekommt also beim Aufruf immer einen Parameter weniger übermittelt, als in der Definition angegeben ist. Dies liegt daran, dass die Methode *für* ein bestimmtes Objekt aufgerufen wird. Innerhalb der Methode ist daher bekannt, um welches Objekt es sich handelt, meist mithilfe von `self`.

Die Geschwindigkeit des Objekts `volvo` wird ausgegeben, einmal vor und einmal nach der Beschleunigung. Die Geschwindigkeit des Objekts `opel` wird nur einmal ausgegeben. Zu Beginn, also nach ihrer Erzeugung, haben die Objekte laut der Klassendefinition die Geschwindigkeit 0.

Hinweis

Standardmäßig gibt es in objektorientierten Sprachen die Möglichkeit, Eigenschaften innerhalb einer Klasse zu kapseln. Damit soll eine unmittelbare Änderung einer Eigenschaft (Beispiel: `volvo.geschwindigkeit = 100`) durch den Benutzer der Klasse verhindert werden. Eine Änderung soll nur durch die definierten Methoden möglich sein (Beispiel: `volvo.beschleunigen(20)`).

In Python können die Eigenschaften einer Klasse nicht gekapselt werden. Sie könnten allerdings einer Konvention folgen und den Namen der Eigenschaft mit einem `_` (Unterstrich) beginnen lassen (Beispiel: `_geschwindigkeit = 0` und `self._geschwindigkeit += wert`). Damit dokumentieren Sie, dass diese Eigenschaft geschützt sein soll.

6.3 Besondere Member

Es gibt einige besondere Member, die häufig im Zusammenhang mit einer Klasse definiert bzw. genutzt werden können. Sie sind leicht erkennbar, da ihr Name mit einem doppelten Unterstrich beginnt und endet:

- Sie können eine besondere Methode mit dem festgelegten Namen `__init__()` als Konstruktormethode definieren, die ein Objekt zu Beginn seiner *Lebensdauer* mit Anfangswerten initialisiert. Dabei kann mit optionalen (siehe [Abschnitt 5.6.3, »Optionale Parameter«](#)) und benannten Parametern (siehe [Abschnitt 5.6.2, »Benannte Parameter«](#)) gearbeitet werden, damit Objekte ihre Anfangswerte auf unterschiedliche Art und Weise erhalten können.
- Sie können eine besondere Methode mit dem festgelegten Namen `__str__()` definieren, die die Eigenschaften eines Objekts ausgibt.
- Die Eigenschaft `__dict__` stellt ein Dictionary mit den Namen und Werten der Eigenschaften bereit.
- Sie können eine besondere Methode mit dem festgelegten Namen `__del__()` als Destruktormethode definieren, die am Ende der *Lebensdauer* eines Objekts Aktionen auslöst, zum Beispiel das Schließen einer offenen Datei.

Die Klasse `Fahrzeug` wird wie folgt verändert: Ein Fahrzeug erhält neben der Eigenschaft `geschwindigkeit` die Eigenschaft `bezeichnung`, eine Konstruktormethode, eine Ausgabemethode und eine Destruktormethode.

Das Programm sieht in seiner veränderten Form wie folgt aus:

```
class Fahrzeug:  
    def __init__(self, bez="leer", ge=0):  
        self.bezeichnung = bez
```

```

        self.geschwindigkeit = ge
    def __str__(self):
        return f"{self.bezeichnung} {self.geschwindigkeit} km/h"
    def __del__(self):
        print(f"Entfernt: {self}")
    def beschleunigen(self, wert):
        self.geschwindigkeit += wert

opel = Fahrzeug("Opel Admiral", 40)
renault = Fahrzeug("Renault Alpine")
fiat = Fahrzeug(ge=60)
mercedes = Fahrzeug()

print(opel)
print(renault)
print(fiat)
print(mercedes)

opel.beschleunigen(20)
print(opel.__dict__)
del opel
# print(opel)

```

Listing 6.3 Datei »oop_besondere.py«

Die Ausgabe des Programms lautet:

```

Opel Admiral 40 km/h
Renault Alpine 0 km/h
(leer) 60 km/h
(leer) 0 km/h
{'bezeichnung': 'Opel Admiral', 'geschwindigkeit': 60}
Entfernt: Opel Admiral 60 km/h

```

Die Konstruktormethode besitzt zwei Parameter, die dazu genutzt werden, die beiden Eigenschaften zu erzeugen und ihnen Anfangswerte zuzuweisen. Beide Parameter sind optional. Fehlt der zweite Parameter, wird sein Vorgabewert genutzt. Ist kein Wert für den ersten Parameter verfügbar, kann der zweite Parameter als benannter Parameter übergeben werden. Fehlen beide Parameter, werden beide Vorgabewerte verwendet.

Die Ausgabemethode liefert eine Zeichenkette mit den Werten der Eigenschaften eines Objekts.

Die Eigenschaft `__dict__` können Sie zum Beispiel für den Datenaustausch mithilfe von JSON nutzen, siehe [Abschnitt 8.6, »Datenaustausch mit JSON«](#).

In der hier definierten Destruktormethode wird nur eine Information ausgegeben. Sie wird mithilfe des Schlüsselworts `del` aufgerufen. Anschließend existiert das Objekt nicht mehr.

Beachten Sie (hier in der Destruktormethode) den Aufruf der Ausgabemethode mithilfe von `self`.

Hinweis

Sollten Sie bereits mit anderen Programmiersprachen gearbeitet haben: Methoden können, wie auch Funktionen, in Python nicht überladen werden. Definieren Sie eine Methode mehrfach, gegebenenfalls mit unterschiedlichen Parametern, so gilt nur die jeweils letzte Definition. Methoden können allerdings in abgeleiteten Klassen überschrieben werden, siehe [Abschnitt 6.6, »Vererbung«](#).

6.4 Operatormethoden

Operatormethoden werden für Objekte mithilfe von Operatoren aufgerufen. Sie sind für die eingebauten Datentypen bereits vordefiniert. Für Ihre eigenen Datentypen, also Ihre Klassen, können Sie sie selbst definieren.

Nachfolgend werden drei dieser Methoden zum Vergleich zweier Objekte bzw. zur Subtraktion zweier Objekte der Klasse `Fahrzeug` definiert und genutzt:

```
class Fahrzeug:
    def __init__(self, bez, ge):
        self.bezeichnung = bez
        self.geschwindigkeit = ge
    def __gt__(self, other):
        return self.geschwindigkeit > other.geschwindigkeit
    def __eq__(self, other):
        return self.geschwindigkeit == other.geschwindigkeit
    def __sub__(self, other):
        return self.geschwindigkeit - other.geschwindigkeit

opel = Fahrzeug("Opel Admiral", 60)
volvo = Fahrzeug("Volvo Amazon", 45)

if opel > volvo:
    print("Opel ist schneller")
elif opel == volvo:
    print("Beide sind gleich schnell")
else:
    print("Volvo ist schneller")

differenz = opel - volvo
print(f"Geschwindigkeitsdifferenz: {differenz} km/h")
```

Listing 6.4 Datei »oop_operator.py«

Die Ausgabe des Programms:

```
Opel ist schneller
Geschwindigkeitsdifferenz: 15 km/h
```

Der Ausdruck `opel > volvo` ruft für das Objekt `opel` die Operatormethode `__gt__()` auf und übergibt den Parameter `volvo`. Ebenso verhält es sich bei der Methode `__eq__()` und dem Ausdruck `opel == volvo`.

- Die Methode `__gt__()` liefert `True`, falls das Objekt `self` größer als das Objekt `other` ist. Für diese Klasse wird definiert: Größer bedeutet höhere Geschwindigkeit.
- Die Methode `__eq__()` liefert `True`, falls die beiden Objekte gleich groß sind, hier also: gleich schnell.

Die Methode `__sub__()` wird für das Objekt `opel` beim Einsatz des Operators - aufgerufen. Sie liefert die Differenz zweier Objekte zurück. Hier bezieht sich das auf die Differenz der Geschwindigkeiten der beiden Objekte.

Neben den Vergleichsoperatoren `>` (*greater than*) und `==` (*equal*) können Sie weitere nutzen:

- `>=` führt zu `__ge__()`, *greater equal*
- `<` führt zu `__lt__()`, *lower than*
- `<=` führt zu `__le__()`, *lower equal*
- `!=` führt zu `__ne__()`, *not equal*

Neben dem Rechenoperator `-` (minus) können Sie weitere nutzen, unter anderem:

- `+` führt zu `__add__()`
- `*` führt zu `__mul__()`
- `/` führt zu `__truediv__()`
- `//` führt zu `__floordiv__()`
- `%` führt zu `__mod__()`
- `**` führt zu `__pow__()`

6.5 Referenz, Identität und Kopie

Wie Sie bereits in [Abschnitt 4.8](#), »Referenz, Identität und Kopie«, erfahren haben, ist der Name eines Objekts lediglich eine Referenz auf das Objekt. Auch bei Objekten von eigenen Klassen erzeugt die Zuweisung dieser Referenz an einen anderen Namen lediglich eine zweite Referenz auf dasselbe Objekt.

Echte Kopien von Objekten benutzerdefinierter Klassen können Sie durch Erzeugung eines neuen Objekts erstellen, dem Sie die Werte eines anderen Objekts derselben Klasse zuweisen.

Für die Kopie von umfangreichen Objekten können Sie sich ebenfalls der Funktion `deepcopy()` aus dem Modul `copy` bedienen, siehe auch [Abschnitt 4.8.3](#), »Objekte kopieren«. Im folgenden Programm sehen Sie dazu Beispiele:

```
import copy

class Fahrzeug:
    def __init__(self, bez, ge):
        self.bezeichnung = bez
        self.geschwindigkeit = ge
    def beschleunigen(self, wert):
        self.geschwindigkeit += wert
    def __str__(self):
        return f"{self.bezeichnung} {self.geschwindigkeit} km/h"

opel = Fahrzeug("Opel Admiral", 40)
print("Original:", opel)
print()

opel2 = Fahrzeug(opel.bezeichnung, opel.geschwindigkeit)
opel2.beschleunigen(30)
print("Kopie:", opel2)
print("Identität", opel is opel2)
print()

opel3 = copy.deepcopy(opel)
opel3.beschleunigen(35)
print("Kopie:", opel3)
print("Identität:", opel is opel3)
print()

opel4 = opel
opel4.beschleunigen(40)
```

```
print("Referenz:", opel4)
print("Identität:", opel is opel4)
```

Listing 6.5 Datei »oop_kopieren.py«

Die Ausgabe des Programms:

```
Original: Opel Admiral 40 km/h
```

```
Kopie: Opel Admiral 70 km/h
Identität False
```

```
Kopie: Opel Admiral 75 km/h
Identität: False
```

```
Referenz: Opel Admiral 80 km/h
Identität: True
```

Das Objekt `opel` ist das Original. Das Objekt `opel2` ist eine Kopie. Es handelt sich um ein neues Objekt, das mithilfe der Daten des Originalobjekts erzeugt wird. Das Objekt `opel3` ist ebenfalls eine Kopie. Es wird mithilfe der Funktion `copy.deepcopy()` erzeugt.

Bei `opel4` handelt es sich um eine zusätzliche Referenz auf das Originalobjekt `opel`. Die Änderung einer Eigenschaft über diese Referenz ändert das Originalobjekt, wie die Ausgabe zeigt.

Die Vergleiche auf Identität mithilfe des Operators `is` zeigen, dass nur die Objekte `opel` und `opel4` identisch sind.

6.6 Vererbung

Eine Klasse kann ihre Eigenschaften und Methoden an eine andere Klasse vererben. Dieser Mechanismus wird häufig angewendet. Damit erzeugt man eine Hierarchie von Klassen, die die Darstellung von Objekten ermöglicht, die ähnliche Merkmale aufweisen.

Im folgenden Beispiel wird eine Klasse `PKW` auf Basis der bereits existierenden Klasse `Fahrzeug` definiert. Bei der Klasse `PKW` kommen noch einige Eigenschaften und Methoden hinzu. Diese Klasse ist spezialisiert, bezogen auf die allgemeine Klasse `Fahrzeug`.

Von der Klasse `PKW` aus gesehen ist die Klasse `Fahrzeug` eine Basisklasse. Von der Klasse `Fahrzeug` aus gesehen ist die Klasse `PKW` eine abgeleitete Klasse. Zunächst die Definition der Basisklasse `Fahrzeug`. Sie enthält drei Methoden und zwei Eigenschaften:

```
class Fahrzeug:
    def __init__(self, bez, ge):
        self.bezeichnung = bez
        self.geschwindigkeit = ge
    def beschleunigen(self, wert):
        self.geschwindigkeit += wert
    def __str__(self):
        return f"{self.bezeichnung} {self.geschwindigkeit} km/h"
```

Listing 6.6 Datei »oop_vererbung.py«, Basisklasse »Fahrzeug«

Es folgt die Definition der abgeleiteten Klasse `PKW`:

```
class PKW(Fahrzeug):
    def __init__(self, bez, ge, ins):
        super().__init__(bez, ge)
        self.insassen = ins
    def __str__(self):
        return f"{super().__str__()} {self.insassen} Insassen"
    def einsteigen(self, anzahl):
        self.insassen += anzahl
    def aussteigen(self, anzahl):
        self.insassen -= anzahl
```

Listing 6.7 Datei »oop_vererbung.py«, abgeleitete Klasse »PKW«

Bei der Definition der abgeleiteten Klasse folgt nach ihrem eigenen Namen der Name der Basisklasse in runden Klammern.

Die Klasse `PKW` erbt von der Klasse `Fahrzeug` und enthält fünf Methoden und drei Eigenschaften:

- eine eigene Konstruktormethode und eine eigene Ausgabemethode, die jeweils die gleichnamige Methode der Basisklasse überschreiben
- die eigenen Methoden `einstigen()` und `aussteigen()`
- die von der Klasse `Fahrzeug` geerbte Methode `beschleunigen()`
- die eigene Eigenschaft `insassen`
- die geerbten Eigenschaften `bezeichnung` und `geschwindigkeit`

Eine Methode einer Basisklasse kann in einer abgeleiteten Klasse überschrieben werden. Zum expliziten Aufruf einer Methode der übergeordneten Klasse wird die Referenz `super()` benötigt.

Es folgt das Hauptprogramm, in dem ein Objekt der Klasse `PKW` erzeugt, verändert und ausgegeben wird:

```
fiat = PKW("Fiat Marea", 50, 0)
fiat.einstigen(3)
fiat.aussteigen(1)
fiat.beschleunigen(10)
print(fiat)
```

Listing 6.8 Datei »oop_vererbung.py«, Hauptprogramm

Die Ausgabe des Programms:

```
Fiat Marea 60 km/h 2 Insassen
```

Allgemein gilt: Eigenschaften und Methoden werden zunächst in der Klasse des Objekts gesucht. Sind sie dort nicht vorhanden, wird die Suche in der zugehörigen Basisklasse fortgesetzt.

Es wird das Objekt `fiat` erzeugt, dabei wird der Konstruktor aufgerufen. Er wird in der abgeleiteten Klasse gefunden. Er gibt diejenigen Eigenschaften, die nicht in der abgeleiteten Klasse definiert wurden, über einen Aufruf des Konstruktors der

übergeordneten Klasse weiter. Die Methode `super()` liefert dazu eine Referenz auf das aktuelle Objekt in der übergeordneten Klasse. Diese abgestufte Vorgehensweise ist nicht zwingend vorgeschrieben, aber empfehlenswert. Anschließend wird die verbliebene Eigenschaft der abgeleiteten Klasse initialisiert.

Es werden die Methoden `einstiegen()` und `aussteigen()` aufgerufen. Diese werden in der abgeleiteten Klasse gefunden und verändern die Eigenschaft `insassen`.

Anschließend wird die Methode `beschleunigen()` aufgerufen. Diese wird in der abgeleiteten Klasse nicht gefunden, daher wird in der Basisklasse weitergesucht. Dort wird sie gefunden und dient zur Veränderung der Eigenschaft `geschwindigkeit`.

Es wird die Ausgabemethode aufgerufen. Diese wird in der abgeleiteten Klasse gefunden. Sie ruft zunächst mithilfe von `super()` die gleichnamige Methode der übergeordneten Klasse auf, damit deren Eigenschaften ausgegeben werden können. Das Ergebnis dieses Aufrufs wird mit der Ausgabe der Eigenschaften der abgeleiteten Klasse zusammengesetzt. Dies führt zur Ausgabe aller Daten des Objekts.

6.7 Mehrfachvererbung

Eine abgeleitete Klasse kann wiederum Basisklasse für eine weitere Klasse sein. Dadurch ergibt sich eine Vererbung über mehrere Ebenen.

Bei der Mehrfachvererbung kann eine Klasse außerdem von mehr als einer Klasse erben. Sie übernimmt in diesem Fall die Eigenschaften und Methoden aller Basisklassen.

Im folgenden Beispiel werden drei Klassen definiert:

1. die Klasse `Datum` mit den Eigenschaften `tag`, `monat` und `jahr`
2. die Klasse `Uhrzeit` mit den Eigenschaften `stunde`, `minute` und `sekunde`
3. die Klasse `Zeit`, die von diesen beiden Klassen erbt

Anschließend wird jeweils ein Objekt erzeugt und ausgegeben:

```
class Datum:  
    def __init__(self, d, m, y):  
        self.tag = d  
        self.monat = m  
        self.jahr = y  
    def __str__(self):  
        return f"{self.tag:02d}.{self.monat:02d}.{self.jahr:d}"  
  
class Uhrzeit:  
    def __init__(self, h, m, s):  
        self.stunde = h  
        self.minute = m  
        self.sekunde = s  
    def __str__(self):  
        return f"{self.stunde:02d}:{self.minute:02d}:{self.sekunde:02d}"  
  
class Zeit(Datum, Uhrzeit):  
    def __init__(self, d, mo, y, h, mi, s):  
        Datum.__init__(self, d, mo, y)  
        Uhrzeit.__init__(self, h, mi, s)  
    def __str__(self):  
        return f"{Datum.__str__(self)} {Uhrzeit.__str__(self)}"  
  
d = Datum(3, 1, 2022)  
print(d)  
u = Uhrzeit(16, 5, 20)  
print(u)
```

```
z = Zeit(9, 5, 2022, 9, 35, 8)
print(z)
```

Listing 6.9 Datei »oop_mehrfach.py«

Die Ausgabe des Programms:

```
03.01.2022
16:05:20
09.05.2022 09:35:08
```

Die beiden Klassen, von denen die Klasse `Zeit` erbt, werden in den runden Klammern nach dem Namen der Klasse notiert, durch Komma getrennt.

In der Klasse `Zeit` kann der Aufruf der Konstruktoren bzw. Ausgabemethoden der übergeordneten Klassen nicht mehr mithilfe von `super()` erfolgen, da diese Referenz hier mehrdeutig ist.

Der Name der übergeordneten Klasse kann allerdings auch explizit angegeben werden. Damit wird die Zuordnung eindeutig. Es muss zusätzlich die Referenz `self` auf das aktuelle Objekt als erster Parameter notiert werden.

Hinweis

Nicht alle objektorientierten Programmiersprachen bieten das Konzept der Mehrfachvererbung an, da es den Entwurf von Klassenbibliotheken komplex und unübersichtlich machen kann.

6.8 Datenklassen

Seit Python 3.7 haben Sie die Möglichkeit, zusammengehörige Werte auf einfache Weise mithilfe von Datenklassen gemeinsam zu speichern.

Dabei handelt es sich um vereinfachte Klassen. Einige der besonderen Methoden sind für diese Klassen bereits vordefiniert, wie zum Beispiel eine Konstruktormethode, eine Ausgabemethode und eine Methode für den Operator `==`.

Im nachfolgenden Beispiel wird eine Datenklasse für zweidimensionale Vektoren definiert und genutzt:

```
import dataclasses, math
@dataclasses.dataclass
class Vektor:
    x:float = 0.0
    y:float = 0.0

    def betrag(self):
        return math.sqrt(self.x * self.x + self.y * self.y)

va = Vektor(3.0, 4.0)
print(va)
print("Betrag:", va.betrag())

vb = Vektor(3.0, 4.0)
print(vb)

if va == vb:
    print("Vektoren sind gleich")

vc = Vektor(1.8)
print(vc)

vd = Vektor(y=9.1)
print(vd)
```

Listing 6.10 Datei »oop_datenklasse.py«

Die Klasse `Vektor` wird mithilfe des Dekorators `@dataclass` aus dem Modul `dataclasses` als Datenklasse gekennzeichnet. Ein Objekt der Klasse `Vektor` besitzt die beiden Eigenschaften `x` und `y`. Sie stehen für die beiden Komponenten eines zweidimensionalen Vektors.

Die Eigenschaften müssen mithilfe von Typhinweisen (siehe [Abschnitt 3.7.5](#), »Typhinweise«) und Vorgabewerten angegeben werden. Der intern vordefinierte Konstruktor nutzt die beiden Eigenschaften als optionale Parameter.

Die Methode `betrag()` berechnet den Betrag eines zweidimensionalen Vektors nach der folgenden mathematischen Regel: Der Betrag entspricht der Quadratwurzel der Summe der Quadrate der Komponenten.

Zwei Objekte der Klasse `Vektor` werden mit Werten für beide Eigenschaften initialisiert. Die Methode für den Operator `==` ist ebenfalls vordefiniert und vergleicht die Werte aller Eigenschaften.

Das Objekt `vc` erhält den Wert für die Eigenschaft `y` über den Vorgabewert. Das trifft bei Objekt `vd` für die Eigenschaft `x` zu. Der Wert für die Eigenschaft `y` wird hier über einen benannten Parameter übergeben.

Die Ausgabe des Programms:

```
Vektor(x=3.0, y=4.0)
Betrag: 5.0
Vektor(x=3.0, y=4.0)
Vektoren sind gleich
Vektor(x=1.8, y=0.0)
Vektor(x=0.0, y=9.1)
```

Eine Ausgabemethode ist bei Datenklassen ebenfalls bereits vordefiniert. Sie führt zu einer Ausgabe des Namens der Klasse, gefolgt von den Namen der Eigenschaften des Objekts mit ihren jeweiligen Werten.

6.9 Enumerationen

Enumerationen sind Aufzählungen von Konstanten. Der Programmcode wird durch die Nutzung der Elemente einer Enumeration besser lesbar. Python bietet seit der Version 3.4 innerhalb des Moduls `enum` unter anderem die Klasse `IntEnum` zur Erstellung einer Enumeration, deren Elemente als Konstanten für ganze Zahlen stehen.

Ein Beispiel:

```
import enum
class Farbe(enum.IntEnum):
    rot = 5
    gelb = 2
    blau = 4

x = 2
if x == Farbe.gelb:
    print("Das ist gelb")
print(Farbe.gelb)
print(Farbe.gelb * 10)
```

Listing 6.11 Datei »enumeration.py«

Es wird die folgende Ausgabe erzeugt:

```
Das ist gelb
Farbe.gelb
20
```

Nach Import des Moduls `enum` wird die Klasse `Farbe` erzeugt, die von der Klasse `IntEnum` erbt. Die Elemente einer Enumeration müssen unmittelbar mit Werten versehen werden. Die Werte müssen nicht in aufsteigender Reihenfolge sein. Ein bestimmter Wert könnte durch mehrere Konstanten repräsentiert werden, dies wäre aber nicht sinnvoll.

Der Vergleich eines Werts mit anderen Werten wird durch die Elemente der Enumeration erleichtert. Dies verbessert die Lesbarkeit des Programmcodes.

Sie können den repräsentierten Wert auch in Berechnungen einbauen.

6.10 Spiel, objektorientierte Version

In diesem Abschnitt stelle ich Ihnen eine objektorientierte Version des Kopfrechenspiels vor. In der Anwendung gibt es ein Objekt der Klasse `Spiel`. Während der Lebensdauer dieses Objekts werden mehrere Objekte der Klasse `Aufgabe` erzeugt und genutzt.

Zunächst die Importanweisung und das kurze Hauptprogramm:

```
import random
...
s = Spiel()
s.spielen()
print(s)
```

Listing 6.12 Datei »spiel_oop.py«, Hauptprogramm

Das Modul `random` wird für den Zufallsgenerator benötigt. Im Hauptprogramm wird das Objekt `s` der Klasse `Spiel` erzeugt. Damit wird ein Spiel initialisiert. Für dieses Objekt wird zum Start des Vorgangs die Methode `spielen()` ausgeführt. Am Ende werden die Spielergebnisse ausgegeben.

Es folgt die Definition der Klasse `Spiel`:

```
class Spiel:
    def __init__(self):
        random.seed()
        self.richtig = 0
        self.anzahl = -1
        while self.anzahl<1 or self.anzahl>10:
            try:
                print("Wieviele Aufgaben (1 bis 10):")
                self.anzahl = int(input())
            except:
                continue

    def spielen(self):
        for i in range(self.anzahl):
            a = Aufgabe(i+1, self.anzahl)
            print(a)
            self.richtig += a.beantworten()

    def __str__(self):
        return f"Richtig: {self.richtig} von {self.anzahl}"
```

Listing 6.13 Datei »spiel_oop.py«, Klasse »Spiel«

Im Konstruktor der Klasse `spiel` wird der Zufallsgenerator initialisiert. Der Zähler für die richtig gelösten Aufgaben wird zunächst auf `0` gesetzt. Es wird die Anzahl der zu lösenden Aufgaben ermittelt. Dabei werden zwei Eigenschaften der Klasse `spiel` erstellt: `richtig` und `anzahl`.

In der Methode `spielen()` wird die gewünschte Anzahl von Aufgaben erzeugt. Jede Aufgabe ist ein Objekt der Klasse `Aufgabe`. Die Aufgabe wird ausgegeben, also der Benutzerin gestellt. Anschließend wird die Methode `beantworten()` aufgerufen, also die Eingabe der Benutzerin verarbeitet. Rückgabewert der Methode `beantworten()` ist `1` oder `0`. Entsprechend wird der Zähler von `richtig` verändert.

In der Ausgabemethode wird das Ergebnis des Spiels veröffentlicht.

Es folgt die Definition der Klasse `Aufgabe`:

```
class Aufgabe:
    def __init__(self, i, anzahl):
        self.nr = i
        self.anzahl = anzahl

    def __str__(self):
        a = random.randint(10, 30)
        b = random.randint(10, 30)
        self.ergebnis = a + b
        return f"Aufgabe {self.nr} von {self.anzahl}: {a} + {b}"

    def beantworten(self):
        try:
            if self.ergebnis == int(input()):
                print(f"{self.nr}: *** Richtig ***")
                return 1
            else:
                raise
        except:
            print(f"{self.nr}: *** Falsch ***")
            return 0
```

Listing 6.14 Datei »spiel_oop.py«, Klasse »Aufgabe«

Der Konstruktor der Klasse `Aufgabe` wird mit der individuellen Nummer der Aufgabe und der Gesamtanzahl der Aufgaben aufgerufen. Dabei werden zwei Eigenschaften der Klasse `Aufgabe` gesetzt: `nr` und `anzahl`.

In der Ausgabemethode wird die Aufgabe erstellt und veröffentlicht. Das richtige Ergebnis der Aufgabe ist eine Eigenschaft der Klasse `Aufgabe`.

In der Methode `beantworten()` wird die Eingabe der Benutzerin mit dem richtigen Ergebnis der Aufgabe verglichen. Eine falsche oder ungültige Eingabe erzeugt eine Ausnahme; dies führt zur Rückgabe einer `0`. Eine richtige Eingabe führt zur Rückgabe einer `1`.

7 Verschiedene Module

In diesem Kapitel erläutere ich Programmiertechniken zur Arbeit mit Datums- und Zeitangaben, Warteschlangen, Collections, parallel ablaufenden Programmteilen (Multithreading) sowie regulären Ausdrücken.

7.1 Datum und Uhrzeit

Das Modul `time` enthält Funktionen zur Speicherung, Bearbeitung und Ausgabe von Zeitangaben für Datum und Uhrzeit. Auf vielen Betriebssystemen gilt der 1. Januar 1970 00:00 Uhr als Nullpunkt für die Verarbeitung von Zeitangaben. Sie werden in Sekunden ab diesem Zeitpunkt gerechnet.

7.1.1 Funktionen

Die wichtigsten Funktionen sind `time()`, `localtime()`, `strftime()` und `mktime()`. Im folgenden Programm werden ihre Möglichkeiten verdeutlicht:

```
import time
print("time()", time.time())

lt = time.localtime()
print("localtime()", f"{lt[2]:02d}.{lt[1]:02d}.{lt[0]} "
      f"{lt[3]:02d}:{lt[4]:02d}:{lt[5]:02d}")

s = 1_650_000_000
lt = time.localtime(s)
print("localtime()", f"{lt[2]:02d}.{lt[1]:02d}.{lt[0]} "
      f"{lt[3]:02d}:{lt[4]:02d}:{lt[5]:02d}")

tu = 2022, 1, 15, 12, 35, 20, 0, 0, 0

print("strftime()", time.strftime("%d.%m.%Y %H:%M:%S"))
print("strftime()", time.strftime("%d.%m.%Y %H:%M:%S", tu))
```

```
print("strftime():", time.strftime("%d.%m.%Y %H:%M:%S", lt))

print("mktime():", time.mktime(tu))
print("mktime():", time.mktime(lt))
```

Listing 7.1 Datei »zeit.py«

Die Funktion `time()` liefert die Anzahl der Sekunden, die seit dem Nullpunkt verstrichen sind, bezogen auf die aktuelle Systemzeit des Rechners.

Die Funktion `localtime()` hat einen optionalen Parameter und liefert eine Zeitangabe als Variable des Typs `struct_time`:

- Wird sie ohne Parameter aufgerufen, wird eine Zeitangabe mit der aktuellen Systemzeit des Rechners ermittelt.
- Wird sie mit einer Zahl als Parameter aufgerufen, wird diese als Anzahl der Sekunden zum Nullpunkt hinzugefügt. Aus diesem Zeitpunkt wird eine Zeitangabe ermittelt.

Eine Variable des Typs `struct_time` enthält insgesamt neun Elemente einer Zeitangabe, auf die einzeln zugegriffen werden kann. Die Elemente 0 bis 5 enthalten nacheinander die Werte für Jahr, Monat, Tag, Stunde, Minute und Sekunde. Element 6 enthält die Nummer des Wochentags, von Montag = 0 bis Sonntag = 6. Die laufende Nummer des Tages im Jahr wird von Element 7 bereitgestellt. Element 8 enthält den Status der Sommerzeit: 1 = Sommerzeit ist aktiv, 0 = Sommerzeit ist nicht aktiv.

Die Ausgabe des ersten Programmteils:

```
time(): 1637311991.960514
localtime(): 19.11.2021 09:53:12
localtime(): 15.04.2022 07:20:00
```

Die Funktion `strftime()` dient zur Ausgabe einer formatierten Zeitangabe. Sie benötigt als ersten Parameter eine Zeichenkette mit Angaben zur Formatierung und besitzt einen zweiten, optionalen Parameter:

- Wird sie mit nur einem Parameter aufgerufen, enthält die formatierte Zeitangabe die aktuelle Systemzeit des Rechners.

- Bei dem zweiten Parameter muss es sich um eine Variable des Typs `struct_time` oder um ein Tupel handeln, das denselben Aufbau wie eine solche Variable hat. Aus den Elementen ergibt sich die formatierte Zeitangabe. Die Angaben für Wochentag, Tag des Jahres und Sommerzeit sind häufig zunächst unbekannt. Wird hier einfach jeweils eine `0` gesetzt, werden die richtigen Werte automatisch ermittelt.
- Es gibt unter anderem folgende Formatierungsangaben: `%d` = Tag, `%m` = Monat, `%Y` = Jahr, `%H` = Stunde, `%M` = Minute, `%S` = Sekunde. Alle Ausgaben erfolgen zweistellig und gegebenenfalls mit führender Null. Eine Ausnahme bildet die vierstellige Ausgabe für das Jahr.

Die Funktion `mkttime()` liefert zu einer Zeitangabe die Anzahl der Sekunden seit dem Nullpunkt. Sie benötigt einen Parameter, bei dem es sich wiederum um ein Tupel aus neun Elementen oder eine Variable des Typs `struct_time` handeln muss. Die Anzahl der Sekunden kann zum Rechnen mit Zeitangaben genutzt werden.

Die Ausgabe des zweiten Programmteils:

```
strftime(): 19.11.2021 09:53:12
strftime(): 15.01.2022 12:35:20
strftime(): 15.04.2022 07:20:00
mkttime(): 1642246520.0
mkttime(): 1650000000.0
```

7.1.2 Rechnen mit Zeitangaben

Zur Berechnung eines Zeitraums ermitteln Sie die Differenz zwischen zwei einzelnen Zeitangaben in Sekunden. Daraus lässt sich auch die Differenz in Minuten, Stunden und Tagen ermitteln.

In dem nun folgenden Programm wird die Differenz zwischen dem 15. Februar 2022, 23:55:00 Uhr, und dem 16. Februar 2022, 00:05:15 Uhr, berechnet.

```
import time

z_start = 2022, 2, 15, 23, 55, 0, 0, 0, 0
```

```

print("Start:", time.strftime("%d.%m.%Y %H:%M:%S", z_start))
mk_start = time.mktime(z_start)

z_ende = 2022, 2, 16, 0, 5, 15, 0, 0, 0
print("Ende: ", time.strftime("%d.%m.%Y %H:%M:%S", z_ende))
mk_ende = time.mktime(z_ende)
print()

print("Differenz:")
diff_sek = mk_ende - mk_start
diff_min = diff_sek/60
diff_std = diff_min/60
diff_tag = diff_std/24

print(diff_sek, "Sekunden")
print(diff_min, "Minuten")
print(diff_std, "Stunden")
print(diff_tag, "Tage")

```

Listing 7.2 Datei »zeit_rechnen.py«

Folgende Ausgabe wird erzeugt:

```

Start: 15.02.2022 23:55:00
Ende: 16.02.2022 00:05:15

Differenz:
615.0 Sekunden
10.25 Minuten
0.1708333333333334 Stunden
0.00711805555555555 Tage

```

In der Variablen `diff_sek` wird die Differenz zwischen den beiden Zeitangaben `mk_start` und `mk_ende` in Sekunden berechnet. Zur Ermittlung der Minuten wird der Wert für die Sekunden durch 60 geteilt. Zur Ermittlung der Stunden wird der Wert für die Minuten wiederum durch 60 geteilt. Zur Ermittlung der Tage wird zuletzt der Wert für die Stunden durch 24 geteilt.

7.1.3 Programm anhalten

Die Funktion `sleep()` aus dem Modul `time` ermöglicht das Anhalten eines Programms für einen bestimmten Zeitraum.

Im folgenden Beispielprogramm wird innerhalb einer Schleife mehrmals die aktuelle Zeit ausgegeben. Nach jeder Ausgabe wird das Programm für zwei Sekunden mithilfe der Funktion `sleep()`

angehalten. Am Ende wird die Zeitdifferenz zwischen Start und Ende berechnet.

```
import time

z_start = time.time()
print("Start:", z_start)
for i in range(5):
    time.sleep(2)
    print(time.time())
z_ende = time.time()
print("Ende:", z_ende)

differenz = z_ende - z_start
print("Differenz:", differenz)
```

Listing 7.3 Datei »zeit_anhalten.py«

Es wird die folgende Ausgabe erzeugt:

```
Start: 1637324746.7443616
1637324748.7600112
1637324750.7755709
1637324752.791132
1637324754.8067403
1637324756.8223028
Ende: 1637324756.8223028
Differenz: 10.077941179275513
```

Wie die Ausgabe der Differenz zeigt, lässt sich die Funktion `sleep()` nur für eine zeitliche Steuerung von begrenzter Genauigkeit einsetzen, und zwar aus den folgenden Gründen:

- zum einen benötigen die einzelnen Programmschritte eine eigene, wenn auch kurze Laufzeit, abhängig von der Geschwindigkeit und der Auslastung des Rechners
- zum anderen unterscheiden sich die Zeitabstände geringfügig

Dennoch können Sie die Funktion `sleep()` für viele Anwendungen sinnvoll einsetzen, unter anderem im Bereich von grafischen Animationen, Simulationen oder der Spieleprogrammierung.

7.1.4 Spiel, Version mit Zeitmessung

Datums- und Zeitangaben ermöglichen Erweiterungen des Kopfrechenspiels. Wir können eine Zeitmessung einbauen, die feststellt, wie lange ein Benutzer zur Lösung einer oder aller Aufgaben benötigt hat. Wird eine Bestleistung erzielt, zum Beispiel eine hohe Anzahl von gelösten Aufgaben im ersten Versuch oder die Lösung der Aufgaben innerhalb besonders kurzer Zeit, können wir das mit Datum und Uhrzeit festhalten.

In dieser Version des Kopfrechenspiels werden fünf Additionsaufgaben mit Zahlen aus dem Bereich von 10 bis 30 gestellt. Der Spieler hat pro Aufgabe nur einen Versuch. Ungültige Eingaben oder falsche Ergebnisse werden einfach mit dem Text `Falsch` kommentiert und bewertet. Am Ende wird die Gesamtzeit angegeben, die der Spieler benötigt hat.

```
import random, time

random.seed()
richtig = 0
z_start = time.time()

for aufgabe in range(5):
    a = random.randint(10, 30)
    b = random.randint(10, 30)
    c = a + b
    print(f"Aufgabe {aufgabe+1} von 5: {a} + {b}")
    try:
        zahl = int(input("Bitte Lösungsvorschlag eingeben: "))
        if zahl == c:
            print("Richtig")
            richtig += 1
        else:
            raise
    except:
        print("Falsch")

differenz = time.time() - startzeit
print(f"Richtig: {richtig} von 5 in {differenz:.2f} Sekunden")
print("Ergebnis erzielt:", time.strftime("%d.%m.%Y %H:%M:%S"))
```

Listing 7.4 Datei »spiel_zeit.py«

Nachfolgend eine mögliche Ausgabe. Es werden auch einige ungültige oder falsche Ergebnisse eingegeben:

```
Aufgabe 1 von 5: 20 + 19
Bitte Lösungsvorschlag eingeben: 39
Richtig
Aufgabe 2 von 5: 14 + 13
```

```
Bitte Lösungsvorschlag eingeben: abc
Falsch
Aufgabe 3 von 5: 14 + 20
Bitte Lösungsvorschlag eingeben: 34
Richtig
Aufgabe 4 von 5: 19 + 12
Bitte Lösungsvorschlag eingeben: 0
Falsch
Aufgabe 5 von 5: 20 + 24
Bitte Lösungsvorschlag eingeben: 44
Richtig
Richtig: 3 von 5 in 11.53 Sekunden
Ergebnis erzielt: 19.11.2021 13:31:44
```

Am Anfang und am Ende wird die Zeit genommen und anschließend die Differenz berechnet. Die richtigen Ergebnisse werden mitgezählt. Im Fall einer ungültigen Eingabe oder eines falschen Ergebnisses wird eine Ausnahme erzeugt.

7.1.5 Spiel, objektorientierte Version mit Zeitmessung

Die objektorientierte Version des Kopfrechenspiels mit Zeitmessung basiert auf der objektorientierten Version ohne Zeitmessung, siehe Datei *spiel_oop.py* in [Abschnitt 6.10](#), »Spiel, objektorientierte Version«. Hier stelle ich nur die Erweiterungen vor. Zunächst die Importanweisung und das kurze Hauptprogramm:

```
import random, time

# Hauptprogramm
s = Spiel()
s.messen(True)
s.spielen()
s.messen(False)
print(s)
```

Listing 7.5 Datei »spiel_zeit_oop.py«, Hauptprogramm

Im Hauptprogramm wird zusätzlich das Modul `time` für die Zeitmessung benötigt. Vor und nach dem Spielvorgang wird die Methode `messen()` zur Ermittlung der Spieldauer aufgerufen. Der Parameter des Typs `bool` gibt an, ob es sich um den Start des Spiels handelt.

Die Definition der Klasse `Aufgabe` wird nicht verändert. Es folgt die Definition der Klasse `spiel`. Darin wird die Methode `messen()` hinzugefügt und die Ausgabemethode verändert:

```
class Spiel:  
    ...  
    def messen(self, start):  
        if start:  
            self.z_start = time.time()  
        else:  
            self.zeit = time.time() - self.z_start  
  
    def __str__(self):  
        datum = time.strftime("%d.%m.%Y")  
        uhrzeit = time.strftime("%H:%M:%S")  
        return f"Richtig: {self.richtig} von {self.anzahl} "\\  
               f"in {self.zeit:.2f} Sekunden\nnam {datum} um {uhrzeit}"
```

Listing 7.6 Datei »spiel_zeit_oop.py«, Klasse »Spiel«, Ausschnitt

In der Methode `messen()` wird die Startzeit ermittelt, falls der Parameterwert `True` übermittelt wird. Andernfalls werden die Endzeit und die Spieldauer ermittelt. Die Spieldauer ist eine Eigenschaft der Klasse `spiel`. In der Ausgabemethode wird das Ergebnis des Spiels zusammen mit Spieldauer, Datum und Uhrzeit veröffentlicht.

7.2 Warteschlangen

Eine Warteschlange (englisch: *queue*) im allgemeinen Sinn entsteht immer dann, wenn ein System innerhalb eines Zeitraums mehr Anforderungen erhält, als es verarbeiten kann. Dabei kann es sich um eine Warteschlange von Personen an einer Kinokasse oder um eine Warteschlange von gesendeten Daten in einem Netzwerk handeln.

Ziel ist es, eine Warteschlange möglichst schnell zu leeren. Abhängig von der Problemstellung kann dies auf unterschiedliche Weise geschehen:

- In einer *FIFO-Queue* wird diejenige Anforderung, die als Erste eingetroffen ist, auch als Erste behandelt. FIFO steht als Abkürzung für *First In First Out*. Anforderungen können nur am Ende der Warteschlange eintreffen und nur am Anfang der Warteschlange behandelt werden.
- In einer *LIFO-Queue* wird diejenige Anforderung, die als Letzte eingetroffen ist, als Erste behandelt. LIFO steht für *Last In First Out*. Anforderungen können nur am Ende der Warteschlange eintreffen und auch nur dort behandelt werden.
- Eine *Double-Ended Queue* besitzt keinen Anfang, aber dafür zwei Enden. Anforderungen können an beiden Enden der Warteschlange eintreffen und auch an beiden Enden behandelt werden.

Die Module `queue` und `collections` bieten einige spezialisierte Datentypen zur Bearbeitung von Warteschlangen. Sie ähneln den eingebauten Datentypen (wie Dictionary, Liste, Set oder Tupel), haben aber zusätzliche Fähigkeiten und bieten einen sehr schnellen Zugriff.

7.2.1 Klasse »SimpleQueue«

Seit Python 3.7 gibt es im Modul `queue` die Klasse `SimpleQueue` zur Bearbeitung einer FIFO-Queue. Nachfolgend ein Beispiel:

```
import queue
x = queue.SimpleQueue()
x.put(5)
x.put(19)
x.put(-2)
x.put(12)
print("Anzahl Elemente:", x.qsize())

while not x.empty():
    print(x.get())
print("Anzahl Elemente:", x.qsize())
```

Listing 7.7 Datei »queue_fifo.py«

Es wird das Objekt `x` der Klasse `SimpleQueue()` erzeugt. Die Methode `qsize()` liefert die Anzahl der Elemente. Die Methode `put()` fügt einzelne Elemente am Ende der Warteschlange hinzu. Die Methode `empty()` liefert die Information, ob die Warteschlange leer ist. Die Methode `get()` entfernt ein Element vom Anfang der Warteschlange und liefert gleichzeitig seinen Wert.

Die Ausgabe des Programms:

```
Anzahl Elemente: 4
5
19
-2
12
Anzahl Elemente: 0
```

Der Wert 5 wurde der Warteschlange als Erster hinzugefügt. Er wird auch als erster Wert wieder entnommen (FIFO).

7.2.2 Klasse »LifoQueue«

Zur Bearbeitung einer LIFO-Queue bietet das Modul `queue` die Klasse `LifoQueue`. Nachfolgend ein Beispiel:

```
import queue
x = queue.LifoQueue()
x.put(5)
x.put(19)
```

```

x.put(-2)
x.put(12)
print("Anzahl Elemente:", x.qsize())

while not x.empty():
    print(x.get())
print("Anzahl Elemente:", x.qsize())

```

Listing 7.8 Datei »queue_lifo.py«

Die Methode `get()` entfernt diesmal jeweils ein Element vom Ende der Warteschlange. Daher erfolgt die Ausgabe der Elemente in umgekehrter Reihenfolge:

```

Anzahl Elemente: 4
12
-2
19
5
Anzahl Elemente: 0

```

7.2.3 Klasse »PriorityQueue«

Die Klasse `PriorityQueue` aus dem Modul `queue` bietet eine Besonderheit: Die Elemente werden in einer sortierten Reihenfolge geliefert. Nachfolgend ein Beispiel:

```

import queue
x = queue.PriorityQueue()
x.put(5)
x.put(19)
x.put(-2)
x.put(12)
print("Anzahl Elemente:", x.qsize())

while not x.empty():
    print(x.get())
print("Anzahl Elemente:", x.qsize())

```

Listing 7.9 Datei »queue_sortiert.py«

Der Name der Klasse ist wiederum geändert. Die Methode `get()` entfernt die Elemente aus der Warteschlange gemäß einer aufsteigenden Sortierung. Die Ausgabe sieht daher wie folgt aus:

```

Anzahl Elemente: 4
-2
5
12
19
Anzahl Elemente: 0

```

7.2.4 Klasse »deque«

Zur Bearbeitung einer Double-Ended Queue gibt es im Modul `collections` die Klasse `deque` (gesprochen: *deck*). Sie bietet weitaus mehr Möglichkeiten als die zuvor behandelten Queue-Klassen. Es folgt ein erstes Beispiel, in dem einige Operationen an einer Double-Ended Queue vorgenommen werden:

```
import collections

d = collections.deque([ 8, 18, 28] )
print("Erstellt:", d)
print("Kopie:", d.copy())

print("Elemente:")
for x in d:
    print(x)
for i in range(len(d)):
    print(f"{i}: {d[ i ]}")

print("Deque verdoppelt:", d * 2)
print("Andere deque addiert:", d + collections.deque([ 9, 19, 29] ))

d.clear()
print("Nach Leerung:", d)
```

Listing 7.10 Datei »deque_operation.py«

Die Ausgabe lautet:

```
Erstellt: deque([8, 18, 28])
Kopie: deque([8, 18, 28])
Elemente:
8
18
28
0: 8
1: 18
2: 28
Deque verdoppelt: deque([8, 18, 28, 8, 18, 28])
Andere deque addiert: deque([8, 18, 28, 9, 19, 29])
Nach Leerung: deque([])
```

Die Funktion `deque()` liefert ein neues Objekt des Datentyps `deque`. Als Parameter wird ein Iterable benötigt. Seit Python 3.5 können Sie mithilfe der Methode `copy()` eine Kopie erstellen. Auf die einzelnen Elemente können Sie mithilfe einer `for`-Schleife zugreifen, mit oder ohne Index.

Ebenfalls seit Python 3.5 können Sie mithilfe des Operators * eine *Double-Ended Queue* vervielfachen und mithilfe des Operators + andere *Double-Ended Queues* addieren. Die Methode `clear()` löscht alle Elemente.

Es folgt ein weiteres Beispiel, in dem eine Double-Ended Queue auf verschiedene Arten verändert wird:

```
import collections
d = collections.deque([ 8, 18, 28 ])
print("Erstellt:", d)

d.appendleft(5)
print("Element angefügt:", d)
d.append(25)
print("Element angefügt:", d)
d.insert(2, 11)
print("Element eingefügt:", d)
d.extendleft([ 7, 9 ])
print("Elemente angefügt:", d)
d.extend([ 17, 19 ])
print("Elemente angefügt:", d)

x = 5
if x in d:
    print(f"Position von {x}: {d.index(5)}")

for i in range(5):
    li = d.popleft()
    print("Entferntes Element, links:", li)
re = d.pop()
print("Entferntes Element, rechts:", re)
print("Danach:", d)

d.rotate()
print("Nach Rotation um +1:", d)
d.rotate(-1)
print("Nach Rotation um -1:", d)
```

Listing 7.11 Datei »deque_aendern.py«

Die Ausgabe lautet:

```
Erstellt: deque([8, 18, 28])
Element angefügt: deque([5, 8, 18, 28])
Element angefügt: deque([5, 8, 18, 28, 25])
Element eingefügt: deque([5, 8, 11, 18, 28, 25])
Elemente angefügt: deque([9, 7, 5, 8, 11, 18, 28, 25])
Elemente angefügt: deque([9, 7, 5, 8, 11, 18, 28, 25, 17, 19])
Position von 5: 2
Entferntes Element, links: 9
Entferntes Element, links: 7
Entferntes Element, links: 5
Entferntes Element, links: 8
```

```
Entferntes Element, links: 11
Entferntes Element, rechts: 19
Danach: deque([18, 28, 25, 17])
Nach Rotation um +1: deque([17, 18, 28, 25])
Nach Rotation um -1: deque([18, 28, 25, 17])
```

Mithilfe der Methoden `appendleft()` und `append()` fügen Sie einzelne Elemente am linken oder rechten Ende hinzu.

Die Methode `insert()` kann seit Python 3.5 zum Einfügen eines Elements an der genannten Position genutzt werden. Gibt es die Position nicht, wird das Element am rechten Ende angefügt.

Die Methoden `extendleft()` und `extend()` dienen zum Erweitern um mehrere Elemente, die aus einem Iterable stammen, am linken bzw. rechten Ende. Die Elemente werden nacheinander in der Reihenfolge des Iterables angefügt. Daher ändert sich die Reihenfolge für die angefügten Elemente bei der Methode `extendleft()`.

Mithilfe der Methode `index()` können Sie seit Python 3.5 nach der Position eines bestimmten Werts suchen. Gibt es kein Element mit diesem Wert, wird eine Ausnahme erzeugt.

Die Methoden `popleft()` und `pop()` entfernen ein Element am linken Ende oder am rechten Ende. Das entfernte Element wird jeweils als Rückgabewert geliefert.

Mithilfe der Methode `rotate()` lassen Sie die Elemente zirkular rotieren. Geben Sie keinen Parameter an, wird um ein Element nach rechts rotiert. Geben Sie negative Werte an, wird nach links rotiert. Ein Element, das bei der Rotation über den Rand geschoben wird, wird am anderen Ende wieder angefügt.

7.3 Multithreading

Der Begriff *Multithreading* bezeichnet die Eigenschaft eines Programms, mehrere Teile, die sogenannten *Threads*, parallel bearbeiten zu lassen. Die Programmiersprache Python bietet die entsprechenden Routinen.

7.3.1 Wozu dient Multithreading?

Es gibt eine Reihe von Problemstellungen, bei denen sich Multithreading als nützlich erweist, zum Beispiel bei Simulationen von realen Prozessen, bei denen eine Aktion eine weitere Aktion anstößt, danach aber selbst auch weiterläuft. Die angestoßene Aktion kann wiederum eine dritte Aktion anstoßen usw. Alle Aktionen greifen auf dieselben Daten zu und verändern sie gegebenenfalls. Sie enden zu verschiedenen Zeitpunkten und mit unterschiedlichen Ergebnissen.

Es kann sich auch um GUI-Anwendungen handeln, bei denen rechenintensive Programmteile im Hintergrund laufen (*ausgelagert werden*) oder parallel die Ergebnisse von Messwertauswertungen oder anderen Echtzeitprozessen dargestellt werden.

7.3.2 Erzeugung eines Threads

Das Modul `threading` bietet eine einfache Möglichkeit zum Multithreading.

Im nachfolgenden Programm wird ein neuer paralleler Thread aus dem Hauptprogramm gestartet. Anschließend muss das Hauptprogramm zehn Sekunden warten und wird danach beendet. Beginn und Ende werden angezeigt. Im parallelen Thread läuft während der Wartezeit die Funktion

`thread_ablauf()`, in der insgesamt fünfmal die aktuelle Zeit angezeigt wird. Nach jeder Anzeige wird der Thread für 1,5 Sekunden angehalten. Beginn und Ende des Threads werden ebenfalls angezeigt.

```
import time, threading

def thread_ablauf():
    print("Start Thread")
    for i in range(5):
        print(i, time.time())
        time.sleep(1.5)
    print("Ende Thread")

print("Start Hauptprogramm")
t = threading.Thread(target=thread_ablauf)
t.start()
time.sleep(10)
print("Ende Hauptprogramm")
```

Listing 7.12 Datei »thread_erzeugen.py«

Es wird eine Ausgabe wie nachfolgend erzeugt:

```
Start Hauptprogramm
Start Thread
0 1637331510.7583477
1 1637331512.2896414
2 1637331513.867718
3 1637331515.4145403
4 1637331516.961366
Ende Thread
Ende Hauptprogramm
```

Zur Erzeugung eines parallelen Threads wird ein Objekt der Klasse `Thread` erstellt. Der Konstruktor erwartet mindestens den benannten Parameter `target`, der den Namen einer Callback-Funktion übermittelt, die beim Start des Threads durchlaufen wird.

Ein Thread wird durch den Aufruf der Funktion `thread_ablauf()` für das Thread-Objekt gestartet. Bei den vorliegenden Zeitangaben endet das Hauptprogramm erst, nachdem der parallel laufende Thread beendet ist.

7.3.3 Identifizierung eines Threads

Zur Unterscheidung der Auswirkungen unterschiedlicher Threads hat jeder Thread eine eindeutige Identifikationsnummer (ID), die Sie mithilfe der Funktion `get_ident()` ermitteln können. Das Hauptprogramm ist ebenfalls ein Thread – auch dieser *Haupt-Thread* verfügt über eine ID.

Das vorherige Programm wird erweitert. Aus dem Haupt-Thread werden, zeitlich versetzt, insgesamt zwei Threads mit der Funktion `thread_ablauf()` gestartet. Für jeden Thread, auch für den Haupt-Thread, wird die ID ermittelt und bei jeder Ausgabe zusätzlich angezeigt.

```
import time, threading

def thread_ablauf():
    id = threading.get_ident()
    print("Start Thread", id)
    for i in range(5):
        print(i, "Thread", id)
        time.sleep(1.5)
    print("Ende Thread", id)
    return

id = threading.get_ident()
print("Start Hauptprogramm", id)

t1 = threading.Thread(target=thread_ablauf)
t1.start()
time.sleep(0.5)
t2 = threading.Thread(target=thread_ablauf)
t2.start()

time.sleep(10)
print("Ende Hauptprogramm", id)
```

Listing 7.13 Datei »thread_ident.py«

Das Programm erzeugt zum Beispiel die Ausgabe:

```
Start Hauptprogramm 1416
Start Thread 1644
0 Thread 1644
Start Thread 5824
0 Thread 5824
1 Thread 1644
1 Thread 5824
2 Thread 1644
2 Thread 5824
3 Thread 1644
3 Thread 5824
4 Thread 1644
4 Thread 5824
```

```
Ende Thread 1644
Ende Thread 5824
Ende Hauptprogramm 1416
```

Der Haupt-Thread hat hier die ID 1416, der erste Unter-Thread die ID 1644, der zweite Unter-Thread die ID 5824. Den parallelen Ablauf der Threads können Sie anhand der IDs beobachten.

7.3.4 Gemeinsame Daten und Objekte

Alle Threads haben Zugriff auf alle globalen Daten und Objekte des übergeordneten Threads. Im folgenden Beispiel wird gezeigt, wie eine Variable vom Haupt-Thread und zwei weiteren Threads gemeinsam genutzt und verändert wird.

```
import time, threading

def thread_ablauf():
    global counter
    id = threading.get_ident()
    for i in range(5):
        counter += 1
        print(i, id, counter)
        time.sleep(1.5)
    return

id = threading.get_ident()
counter = 0
print(id, counter)

t1 = threading.Thread(target=thread_ablauf)
t1.start()
time.sleep(0.5)
t2 = threading.Thread(target=thread_ablauf)
t2.start()
time.sleep(10)

counter += 1
print(id, counter)
```

Listing 7.14 Datei »thread_gemeinsam.py«

Die Ausgabe lautet zum Beispiel:

```
8816 0
0 6868 1
0 9072 2
1 6868 3
1 9072 4
2 6868 5
2 9072 6
```

```
3 6868 7
3 9072 8
4 6868 9
4 9072 10
8816 11
```

Die Variable `counter` ist global im Haupt-Thread (hier: ID 8816) und wird dort mit `0` vorbesetzt. Sie wird mithilfe des Schlüsselworts `global` (siehe auch [Abschnitt 5.6.6](#), »Namensräume«) in der Thread-Funktion bekannt gemacht, um 1 erhöht und angezeigt. In beiden Unter-Threads (hier: ID 6868 und ID 9072) ist die Variable bekannt und kann verändert werden. Am Ende wird sie auch im Haupt-Thread um 1 erhöht und angezeigt.

7.3.5 Threads und Exceptions

Tritt eine unbehandelte Ausnahme auf, betrifft das nur den Thread, in dem sie auftritt. Weder der aufrufende Haupt-Thread noch die anderen Threads werden dadurch gestört.

Das vorherige Programm wird erweitert. Hat die Variable `counter` den Wert 5, wird die unbehandelte Ausnahme `ZeroDivisionError` künstlich hervorgerufen. Der Thread, in dem das geschieht, wird unmittelbar beendet. Die anderen Threads – auch der Haupt-Thread – laufen unbeeinflusst zu Ende. Es folgt das geänderte Programm:

```
import time, threading

def thread_ablauf():
    global counter
    id = threading.get_ident()
    for i in range(5):
        counter += 1
        print(i, id, counter)
        if counter == 5:
            erg = 1/0
            time.sleep(1.5)
    return

id = threading.get_ident()
counter = 0
print(id, counter)

t1 = threading.Thread(target=thread_ablauf)
```

```
t1.start()
time.sleep(0.5)
t2 = threading.Thread(target=thread_ablauf)
t2.start()
time.sleep(10)

counter += 1
print(id, counter)
```

Listing 7.15 Datei »thread_ausnahme.py«

Es wird zum Beispiel die folgende Ausgabe erzeugt:

```
1608 0
0 10156 1
0 3064 2
1 10156 3
1 3064 4
2 10156 5
Exception in thread Thread-1 (thread_ablauf):
Traceback (most recent call last):
...
ZeroDivisionError: division by zero
2 3064 6
3 3064 7
4 3064 8
1608 9
```

Der Thread mit der ID 10156 endet aufgrund der unbehandelten Ausnahme. Der Thread mit der ID 3064 läuft regulär zu Ende, ebenso der Haupt-Thread mit der ID 1608.

7.4 Reguläre Ausdrücke

Reguläre Ausdrücke erleichtern das Suchen und Ersetzen von bestimmten Texten oder Teiltexten. Sie kommen häufig bei der Kontrolle und Auswertung von Benutzereingaben oder bei der Suche nach Dateien zum Einsatz. Wird festgestellt, dass ein Benutzer eine falsche oder unvollständige Eingabe gemacht hat, kann dies zu einer Hilfestellung und einer erneuten Eingabeaufforderung führen.

In Python ermöglicht das Modul `re` das Arbeiten mit regulären Ausdrücken. Zwei Programme zeigen einen Teil der umfangreichen Möglichkeiten.

7.4.1 Suchen von Teiltexten

Zum Suchen von Teiltexten wird im folgenden Programm die Funktion `.findall()` eingesetzt. Sie findet alle Teiltexte, die zum regulären Ausdruck passen, und liefert eine Liste dieser Teiltexte zur weiteren Auswertung zurück. Zur besseren Übersicht ist das Programm unterteilt. Außerdem sind die einzelnen Blöcke des Programms und die zugehörigen Ausgaben nummeriert.

Es folgt Teil 1 des Programms:

```
import re

tx = "Haus und Maus und Laus"
print(tx)
print("1:", re.findall("Maus",tx))
print("2:", re.findall("[ HM] aus",tx))
print("3:", re.findall("[ L-M] aus",tx))
print("4:", re.findall("[ ^L-M] aus",tx))
print("5:", re.findall(".aus",tx))
print("6:", re.findall("^aus",tx))
print("7:", re.findall("aus$",tx))
print()
...
...
```

Listing 7.16 Datei »regexp_suchen.py«, Teil 1

Die Ausgabe des ersten Teils des Programms inklusive der Nummern:

```
Haus und Maus und Laus
1: ['Maus']
2: ['Haus', 'Maus']
3: ['Maus', 'Laus']
4: ['Haus']
5: ['Haus', 'Maus', 'Laus']
6: ['Haus']
7: ['Laus']
```

Die ersten sieben regulären Ausdrücke beziehen sich auf den Text Haus und Maus und Laus.

1. Im ersten Beispiel wird genau die Zeichenfolge `Maus` gesucht. Alle Vorkommen dieser Zeichenfolge werden geliefert.
2. Es werden die Teiltexte gesucht, die mit einem `H` oder einem `M` beginnen und mit `aus` enden. In den rechteckigen Klammern steht: Suche `H` oder `M`.
3. Es werden die Teiltexte gesucht, die mit einem der Zeichen von `L` bis `M` beginnen und mit `aus` enden. In den rechteckigen Klammern wird mithilfe des Bindestrichs ein Bereich angegeben.
4. Es werden die Teiltexte gesucht, die *nicht* mit einem der Zeichen von `L` bis `M` beginnen, aber mit `aus` enden. Das Zeichen `^` bedeutet im Zusammenhang mit einem Bereich eine logische Verneinung.
5. Es werden die Teiltexte gesucht, die mit einem beliebigen Zeichen beginnen und mit `aus` enden. Das Zeichen `.` bedeutet: beliebiges Zeichen.
6. Wie zuvor, aber nur gültig für Teiltexte, die *zu Beginn* des untersuchten Texts stehen. Das Zeichen `^` bedeutet: zu Beginn.
7. Wie zuvor, aber nur gültig für Teiltexte, die *am Ende* des untersuchten Texts stehen. Das Zeichen `$` bedeutet: am Ende.

Es folgt der zweite Teil des Programms:

```
...
tx = "0172-445633"
print(tx)
print("8:", re.findall("[ 0-2]", tx))
print("9:", re.findall("[ ^0-2]", tx))
print("10:", re.findall("[ 047-]", tx))
print()
...

```

Listing 7.17 Datei »regexp_suchen.py«, Teil 2

Die Ausgabe des zweiten Teils des Programms:

```
0172-445633
8: ['0', '1', '2']
9: ['7', '-', '4', '4', '5', '6', '3', '3']
10: ['0', '7', '-', '4', '4']
```

Die nächsten drei Ausdrücke beziehen sich auf den Text 0172-445633.

1. Es wird als Teilstext eine der Ziffern von 0 bis 2 gesucht. Der Bereich umfasst diesmal Ziffern.
2. Es wird als Teilstext ein Zeichen gesucht, das *nicht* im Ziffernbereich von 0 bis 2 liegt. Gefunden werden alle Ziffern ab 3 sowie alle Nichtziffern.
3. Es wird als Teilstext eines der Zeichen aus der angegebenen Menge von Zeichen gesucht. Gefunden werden alle genannten Zeichen oder Ziffern.

Es folgt der letzte Teil des Programms:

```
...
tx = "aa und aba und abba und abbba und aca"
print(tx)
print("11:", re.findall("ab*a", tx))
print("12:", re.findall("ab+a", tx))
print("13:", re.findall("ab?a", tx))
print("14:", re.findall("ab{2,3}a", tx))
print()

tx = "axyz und a123a456a789"
print(tx)
print("15:", re.findall("a.*a", tx))
print("16:", re.findall("a.*?a", tx))
```

Listing 7.18 Datei »regexp_suchen.py«, Teil 3

Die Ausgabe des letzten Teils des Programms:

```
aa und aba und abba und abbba und aca
11: ['aa', 'aba', 'abba', 'abbba']
12: ['aba', 'abba', 'abbba']
13: ['aa', 'aba']
14: ['abba', 'abbba']

axyz und a123a456a789
15: ['axyz und a123a456a']
16: ['axyz und a', 'a456a']
```

Die nächsten vier regulären Ausdrücke beziehen sich auf den Text aa und aba und abba und abbba und aca. An diesem Text wird das Verhalten bei der Wiederholung von Zeichen verdeutlicht.

1. Gefunden werden alle Teiltexte, die wie folgt aussehen: ein `a`, eine beliebige Anzahl (auch 0 möglich) des Zeichens `b`, wiederum ein `a`. Das Zeichen `*` (Stern) bedeutet: beliebige Wiederholungszahl.
2. Gefunden werden alle Teiltexte, die wie folgt aussehen: ein `a`, mindestens ein `b`, wiederum ein `a`. Das Zeichen `+` (plus) bedeutet: beliebige Anzahl von Wiederholungen, mindestens eine.
3. Gefunden werden alle Teiltexte, die wie folgt aussehen: ein `a`, kein oder ein `b`, wiederum ein `a`. Das Fragezeichen `?` bedeutet in diesem Zusammenhang: keine oder eine Wiederholung.
4. Gefunden werden alle Teiltexte, die wie folgt aussehen: ein `a`, zweimal oder dreimal das Zeichen `b`, wiederum ein `a`. Die geschweiften Klammern bieten die Möglichkeit, die gewünschte Anzahl von Wiederholungen anzugeben.

Die letzten zwei regulären Ausdrücke beziehen sich auf den Text axyz und a123a456a789. An diesem Text wird das Verhalten bei der Wiederholung von beliebigen Zeichen verdeutlicht.

1. Gefunden werden alle Teiltexte, die wie folgt aussehen: ein `a`, anschließend so viele beliebige Zeichen wie möglich, wiederum ein `a`. Enthält der durchsuchte Text höchstens ein

`a`, wird keine Zeichenfolge gefunden. Enthält der Text mindestens zwei `a`, wird genau eine Zeichenfolge gefunden.

2. Gefunden werden alle Teiltexte, die wie folgt aussehen: ein `a`, anschließend so wenig beliebige Zeichen wie möglich, wiederum ein `a`. Zu jeweils zwei `a` wird eine Zeichenfolge gefunden.

7.4.2 Ersetzen von Teiltexten

Die Funktion `sub()` ersetzt alle Teiltexte, die zum regulären Ausdruck passen, durch einen anderen Text. Zum leichteren Verständnis werden in diesem Programm die gleichen Texte und regulären Ausdrücke eingesetzt wie im vorherigen Programm. Alle Teiltexte, die gemäß dem Ausdruck gefunden werden, werden durch `x` ersetzt.

```
import re

tx = "Haus und Maus und Laus"
print(tx)
print("1:", re.sub("Maus", "x", tx))
print("2:", re.sub("[ H|M] aus", "x", tx))
print("3:", re.sub("[ L-M] aus", "x", tx))
print("4:", re.sub("[ ^L-M] aus", "x", tx))
print("5:", re.sub(".aus", "x", tx))
print("6:", re.sub("^ .aus", "x", tx))
print("7:", re.sub(".aus$", "x", tx))
print()

tx = "0172-445633"
print(tx)
print("8:", re.sub("[ 0-2]", "x", tx))
print("9:", re.sub("[ ^0-2]", "x", tx))
print("10:", re.sub("[ 047-]", "x", tx))
print()

tx = "aa und aba und abba und abbba und aca"
print(tx)
print("11:", re.sub("ab*a", "x", tx))
print("12:", re.sub("ab+a", "x", tx))
print("13:", re.sub("ab?a", "x", tx))
print("14:", re.sub("ab{2,3}a", "x", tx))
print()

tx = "axyz und a123a456a789"
print(tx)
```

```
print("15:", re.sub("a.*a","x",tx))
print("16:", re.sub("a.*?a","x",tx))
```

Listing 7.19 Datei »regexp_ersetzen.py«

Die Ausgabe dieses Programms sieht wie folgt aus:

```
Haus und Maus und Laus
1: Haus und x und Laus
2: x und x und Laus
3: Haus und x und x
4: x und Maus und Laus
5: x und x und x
6: x und Maus und Laus
7: Haus und Maus und x

0172-445633
8: xx7x-445633
9: 01x2xxxxxx
10: x1x2xxx5633

aa und aba und abba und abbba und aca
11: x und x und x und x und aca
12: aa und x und x und x und aca
13: x und x und abba und abbba und aca
14: aa und aba und x und x und aca

axyz und a123a456a789
15: x789
16: x123x789
```

7.5 Audioausgabe

Das Modul `winsound` bietet unter Windows die Möglichkeit zur Ausgabe von Systemtönen und WAV-Dateien. Ein kleines Beispiel:

```
import winsound, time
for i in range(600, 1500, 200):
    winsound.Beep(i, 500)

time.sleep(3)
winsound.PlaySound("SystemQuestion", winsound.SND_ALIAS)
winsound.PlaySound("GAKkord.wav", winsound.SND_FILENAME)
print("Ende")
```

Listing 7.20 Datei »sound.py«

Die Funktion `Beep()` sendet ein Tonsignal. Der erste Parameter steht für die Frequenz, der zweite Parameter für die Dauer des Tonsignals.

Mithilfe der Funktion `PlaySound()` können Windows-Systemtöne oder WAV-Dateien abgespielt werden. Dabei steht der erste Parameter für den Namen des Systemtons oder den Namen der WAV-Datei. Beim zweiten Parameter handelt es sich um eine Konstante. Im Fall eines Systemtons muss `SND_ALIAS` angegeben werden. Bei einer WAV-Datei ist es `SND_FILENAME`.

In [Abschnitt 8.10](#), »Beispielprojekt Morsezeichen«, sehen Sie ein weiteres Beispiel für die Nutzung des Moduls `winsound`. Dabei wird ein Text in Morsecode umgesetzt, der wiederum als Tonsignal per Lautsprecher ausgegeben wird.

Unterschiede in Ubuntu Linux und macOS

Das Modul `winsound` steht nicht zur Verfügung.

8 Dateien

Die dauerhafte Speicherung von Daten kann in einfachen Dateien oder in Datenbanken erfolgen. In diesem Kapitel lernen Sie verschiedene Techniken zur Speicherung von Daten in Dateien kennen.

Abschließend stelle ich Ihnen das bereits bekannte Kopfrechenspiel in einer Version vor, die das dauerhafte Abspeichern des Namens und der benötigten Zeit in einer Datei ermöglicht.

8.1 Dateitypen

Bei der Ein- und Ausgabe von Daten in Dateien sollten Sie wissen, welcher Dateityp vorliegt und welche Zugriffsart Sie verwenden können. Wir unterscheiden zwischen folgenden Zugriffsarten:

- **Sequenzieller Zugriff:** Diese Möglichkeit wird bei einer Datei bevorzugt, deren einzelne Zeilen unterschiedlich lang sind und jeweils mit einem Zeilenumbruch beendet werden. Der Inhalt der Datei kann mit einem einfachen Editor bearbeitet werden. Die Zeilen werden rein sequenziell gelesen und geschrieben. Der direkte Zugriff auf eine bestimmte Zeile ist nicht möglich, da die Länge der Vorgängerzeilen nicht bekannt ist.
- **Wahlfreier Zugriff:** Diese Möglichkeit haben Sie bei einer Datei, die Datensätze mit fester und bekannter Länge enthält. Zeilenumbrüche können, müssen aber nicht existieren. Neben der Länge sollte die Struktur eines Datensatzes bekannt sein oder innerhalb der Datei an einer vereinbarten Stelle stehen.

Die Zeilen können direkt gelesen und verändert werden, da man den Ort jedes Datensatzes berechnen kann.

- Binärer Zugriff: Diese Zugriffsmöglichkeit steht für alle Dateitypen zur Verfügung. Sie arbeiten mit den reinen Bytefolgen; diese können Sie mithilfe eines darauf angepassten Python-Programms lesen oder verändern. Allerdings kann das zur Folge haben, dass die Dateien nicht mehr mit den zugehörigen Anwendungsprogrammen gelesen werden können. Überschreiben Sie bspw. in einer Oracle-Datenbank die Stelle, an der die Anzahl der Datensätze einer bestimmten Tabelle steht, kann das dazu führen, dass diese Tabelle oder auch alle Tabellen zerstört werden.

Ohne Kenntnis der Struktur einer Datei ist es nicht möglich, sie richtig zu bearbeiten. Neben den genannten Typen gibt es auch Mischformen.

8.2 Öffnen und Schließen einer Datei

Eine Datei, die bearbeitet werden soll, muss vorher geöffnet werden. Dies geschieht mithilfe der eingebauten Funktion `open()`. Dabei werden der Name der Datei (eventuell mit Pfadangabe) und der Öffnungsmodus angegeben. Rückgabewert ist ein Dateiobjekt, das für den weiteren Zugriff auf die Datei verwendet wird.

In den Programmen dieses Abschnitts wird davon ausgegangen, dass sich die zu öffnende Datei im gleichen Verzeichnis wie das Python-Programm befindet. Andernfalls müssen Sie den absoluten oder relativen Pfad zur Datei gemäß [Tabelle 8.1](#) angeben. Eine Datei kann in verschiedenen Modi geöffnet werden, siehe [Tabelle 8.2](#).

Beschreibung	Pfad
zur Datei <code>ein.txt</code> im Unterverzeichnis <code>unt</code>	<code>unt/ein.txt</code>
zur Datei <code>ein.txt</code> im übergeordneten Verzeichnis	<code>../ein.txt</code>
zur Datei <code>ein.txt</code> im parallelen Verzeichnis <code>neb</code>	<code>../neb/ein.txt</code>
zur Datei <code>ein.txt</code> im Verzeichnis <code>C:\Temp</code>	<code>C:/Temp/ein.txt</code>

Tabelle 8.1 Relative und absolute Pfadangaben

Hinweis

Bei Python unter Windows sind sowohl der Forwardslash / als auch der Backslash \ zum Verzeichniswechsel erlaubt. Häufig wird ein Dateizugriff im Zusammenhang mit Internetserver-

Programmierung genutzt. Im Internet herrschen Unix-Server vor, daher nutze ich hier die Unix-freundliche Variante mit dem Forwardslash `/`.

Modus	Erläuterung
<code>r</code>	zum Lesen (Standardwert, kann auch weggelassen werden)
<code>w</code>	zum Schreiben
<code>a</code>	zum Anhängen am Ende der Datei
<code>r+</code>	zum Lesen und Schreiben; aktuelle Position am Anfang
<code>w+</code>	zum Lesen und Schreiben; Datei wird geleert
<code>a+</code>	zum Lesen und Schreiben; aktuelle Position am Ende
<code>b</code>	zum Öffnen einer Datei im Binärmodus, die gelesen oder geschrieben werden soll. Es handelt sich um eine zusätzliche Angabe, siehe auch Abschnitt 8.5 , »Serialisierung mit ›pickle‹«.

Tabelle 8.2 Öffnungsmodi

In jedem Dateiobjekt ist die aktuelle Zugriffsposition gespeichert. Dies ist die Position, an der aktuell gelesen oder geschrieben wird. Sie verändert sich mit jedem Lese- oder Schreibvorgang. Außerdem kann sie mit der Methode `seek()` verändert werden, ohne lesen und schreiben zu müssen.

Nach der Bearbeitung muss eine Datei mit der Methode `close()` geschlossen werden. Wird sie nicht geschlossen, ist sie eventuell

für weitere Zugriffe gesperrt.

8.3 Textdateien

Textdateien können sequenziell beschrieben oder gelesen werden.

8.3.1 Schreiben einer Textdatei

Zum Schreiben in eine Datei muss diese zunächst mit der Funktion `open()` und dem Öffnungsmodus "`w`" geöffnet werden. Wenn die Datei noch nicht existiert, wird sie in diesem Moment erzeugt. Achtung: Wenn sie bereits existiert, wird sie ohne Vorwarnung überschrieben!

Beim Öffnen einer Datei im Schreibmodus kann ein Laufzeitfehler auftreten (zum Beispiel beim Schreiben auf ein schreibgeschütztes Medium oder ein nicht vorhandenes Laufwerk). Daher sollten Sie eine Ausnahmebehandlung durchführen, die das Programm gegebenenfalls mithilfe der Funktion `exit()` aus dem Modul `sys` beendet.

War das Öffnen der Datei erfolgreich, können Sie mit der Methode `write()` einzelne Strings und mit der Methode `writelines()` ein Iterable mit Zeichenketten in die Datei schreiben. Das Zeichen `\n` (*new line*) für das Zeilenende müssen Sie jeweils hinzufügen.

```
import sys

try:
    d = open("schreiben.txt", "w")
except:
    print("Datei nicht geöffnet")
    sys.exit(0)

d.write("Die erste Zeile\n")
for i in range(2,11,2):
    d.write(f"{i} ")
d.write("\n")

li = [ "Hamburg", "Berlin", "München"]
```

```

d.writelines(li)
d.write("\n")
for i in li:
    d.write(f"{i}\n")

dc = {"Peter":31, "Julia":28, "Werner":35}
d.writelines(dc)
d.write("\n")
for k,v in dc.items():
    d.write(f"{k}:{v}\n")

d.close()

```

Listing 8.1 Datei »schreiben.py«

Die Datei *schreiben.txt* hat anschließend den folgenden Inhalt:

```

Die erste Zeile
2 4 6 8 10
HamburgBerlinMünchen
Hamburg
Berlin
München
PeterJuliaWerner
Peter:31
Julia:28
Werner:35

```

Listing 8.2 Datei »schreiben.txt«

Zunächst wird eine einzelne Textzeile mit der Methode `write()` in die Datei geschrieben. Anschließend werden die Zahlen der Zahlenfolge 2, 4, 6, 8, 10 mithilfe eines formatierten String-Literals in eine Zeile der Datei geschrieben. Nach beiden Zeilen wird zusätzlich ein Zeilenende ausgegeben.

Danach wird eine Liste von Zeichenketten zweimal in die Datei geschrieben:

1. Als Ganzes mithilfe der Methode `writelines()`, dabei erscheinen die Elemente in einer Zeile, ohne Trennung zwischen den Elementen.
2. Element für Element mithilfe einer `for`-Schleife, der Methode `write()` und mit jeweils einem Zeilenende

Als Letztes wird ein Dictionary, bei dem die Schlüssel Zeichenketten sind, zweimal in die Datei geschrieben:

1. Als Ganzes mithilfe der Methode `writelines()`, dabei erscheinen nur die Schlüssel in einer Zeile, wiederum ohne Trennung.
2. Element für Element mithilfe einer `for`-Schleife über die Items-View des Dictionarys, der Methode `write()` und mit jeweils einem Zeilenende

Hinweis

Verwenden Sie beim Öffnen der Datei den Öffnungsmodus „`a`“ statt „`w`“ (`d = open("schreiben.txt", "a")`), werden die Ausgaben an den bisherigen Dateiinhalt angehängt. Die Datei wird also immer größer.

Unterschiede unter Ubuntu Linux und macOS

Das Erzeugen von Dateien mithilfe eines Python-Programms kann misslingen, falls nicht Sie der Besitzer der Datei mit dem Python-Programm sind, sondern gegebenenfalls der Benutzer »root«.

8.3.2 Lesen einer Textdatei

Die Zeilen einer Textdatei können Sie unter anderem mithilfe der folgenden Methoden lesen:

- `readlines()` liest alle Zeilen in eine Liste von Strings
- `readline()` liest eine Zeile in einen String
- `read()` liest alle Zeilen in einen String

Nachfolgend wird der Inhalt der Datei `lesen.txt` zum besseren Vergleich mit den drei genannten Methoden gelesen. Die Datei enthält unter anderem einige Zahlen. Die Summe dieser Zahlen wird ermittelt und ausgegeben:

```
abc  
-4
```

Listing 8.3 Datei »lesen.txt«

Es folgt das Programm:

```
import sys

def oeffnen():
    global d
    try:
        d = open("lesen.txt")
    except:
        print("Datei nicht geöffnet")
        sys.exit(0)

def wert(zeile):
    try:
        return float(zeile)
    except:
        return 0.0

oeffnen()
li = d.readlines()
summe = 0
for zeile in li:
    summe += wert(zeile)
print("Summe:", summe)
d.close()

oeffnen()
summe = 0
zeile = d.readline()
while zeile:
    summe += wert(zeile)
    zeile = d.readline()
print("Summe:", summe)
d.close()

oeffnen()
tx = d.read()
li = tx.split("\n")
summe = 0
for zeile in li:
    summe += wert(zeile)
print("Summe:", summe)
d.close()
```

Listing 8.4 Datei »lesen.py«

In der Funktion `oeffnen()` wird die Datei zum Lesen geöffnet. Die globale Variable `d` entspricht einer Referenz auf das Dateiobjekt, das zum Zugriff auf die Inhalte benötigt wird. Gelingt das Öffnen

nicht, weil zum Beispiel die genannte Datei nicht im gewünschten Verzeichnis existiert, wird das Programm beendet.

In der Funktion `wert()` wird der Zahlenwert einer Zeichenkette ermittelt und zurückgeliefert. Ist die Umwandlung nicht erfolgreich, wird der Wert 0.0 zurückgeliefert, der die Summierung nicht beeinflusst.

Bei der Nutzung der drei Methoden wird die Datei jeweils zu Beginn geöffnet und am Ende wieder geschlossen.

Die Liste von Zeichenketten, die die Methode `readlines()` liefert, wird mithilfe einer `for`-Schleife durchlaufen. Der Zahlenwert jeder Zeichenkette wird ermittelt und der Summe hinzugefügt.

Die Methode `readline()` wird zunächst einzeln, anschließend in einer `while`-Schleife aufgerufen. Bei jedem Aufruf liefert die Methode eine Zeichenkette, deren Zahlenwert ermittelt und der Summe hinzugefügt wird. Wird dabei das Dateiende erreicht, ist die Zeichenkette leer. Das führt zum Beenden der `while`-Schleife. Anmerkung: Eine (optisch) leere Zeile enthält das Zeichen für Zeilenende, ist also für die Methode nicht leer.

Die Zeichenkette, die die Methode `read()` liefert, wird mithilfe der Methode `split()` und des Zeichens für Zeilenende in eine Liste von Zeichenketten zerlegt. Diese Liste wird mithilfe einer `for`-Schleife durchlaufen. Der Zahlenwert jeder Zeichenkette wird ermittelt und der Summe hinzugefügt.

Die Ausgabe des Programms lautet:

```
Summe: 4.5
Summe: 4.5
Summe: 4.5
```

8.3.3 CSV-Datei schreiben

CSV-Dateien (CSV = *Comma-separated Values*) sind Textdateien, in denen ein Datensatz pro Zeile steht. Die Daten des Datensatzes

werden durch festgelegte Zeichen voneinander getrennt. Sie können von vielen Programmen (zum Beispiel MS Excel oder LibreOffice Calc) gelesen werden.

Neben den Beispielen in diesem Abschnitt finden Sie in [Abschnitt 8.11](#), »Spiel, Version mit Highscore-Datei«, eine weitere Version des bereits bekannten Kopfrechenspiels. Darin werden die Daten der Spielerin (Name und benötigte Zeit) in einer CSV-Datei gespeichert, sodass Sie eine Highscore-Liste führen können.

Es soll eine Tabelle mit Personendaten, die aus drei Datensätzen besteht, in einer CSV-Datei gespeichert werden. Jeder Datensatz enthält Name, Vorname, Personalnummer, Gehalt und Geburtstag einer Person. Die Tabelle wird im Programm in einer zweidimensionalen Liste gespeichert:

```
import sys
try:
    d = open("daten.csv", "w")
except:
    print("Datei nicht geöffnet")
    sys.exit(0)

li = [[ "Maier", "Hans", 6714, 3500, "15.03.1962"] ,
       [ "Schmitz", "Peter", 81343, 3750, "12.04.1958"] ,
       [ "Mertens", "Julia", 2297, 3621.5, "30.12.1959"] ]
for ds in li:
    gh = str(ds[ 3 ]).replace('.', ',')
    d.write(f"{ds[ 0 ]};{ds[ 1 ]};{ds[ 2 ]};{gh};{ds[ 4 ]}\n")
d.close()
```

Listing 8.5 Datei »schreiben_csv.py«

Die Ausgabedatei *daten.csv* hat anschließend den folgenden Inhalt:

```
Maier; Hans; 6714; 3500; 15.03.1962
Schmitz; Peter; 81343; 3750; 12.04.1958
Mertens; Julia; 2297; 3621,5; 30.12.1959
```

Listing 8.6 Datei »daten.csv«

Nach dem Öffnen der Datei wird die mehrdimensionale Liste erstellt. Sie wird mithilfe einer for-Schleife durchlaufen. Bei jedem Durchlauf werden die Daten eines Datensatzes mithilfe der

Methode `write()` in die Datei geschrieben. Zwischen den Elementen wird jeweils ein Semikolon eingefügt.

Handelt es sich um eine Zahl mit Nachkommastellen, wird zuvor der Dezimalpunkt mithilfe der Methode `replace()` in ein Komma verwandelt. Damit können diese Zahlen problemlos von einer deutschen Version von MS Excel eingelesen werden.

Zur Gewährleistung eines erfolgreichen Exports muss also die Struktur der Daten bekannt sein. Die Daten der verschiedenen Datentypen müssen individuell verarbeitet werden. Ist MS Excel unter Windows installiert, genügt ein Doppelklick auf die Ausgabedatei `daten.csv`, um den Inhalt als Tabelle in MS Excel darzustellen, siehe [Abbildung 8.1](#).

	A	B	C	D	E	F
1	Maier	Hans	6714	3500	15.03.1962	
2	Schmitz	Peter	81343	3750	12.04.1958	
3	Mertens	Julia	2297	3621,5	30.12.1959	
4						

Abbildung 8.1 CSV-Datei in MS Excel

Unterschiede unter Ubuntu Linux und macOS

Unter Ubuntu Linux und macOS können Sie die Datei mithilfe eines Doppelklicks auf den Dateinamen in der Verzeichnisanzeige mit LibreOffice Calc öffnen. Es wird das Dialogfeld **TEXTIMPORT** geöffnet.

Achten Sie darauf, dass im Bereich **TRENNOPTIONEN** (englisch: **SEPARATOR OPTIONS**) nur die Optionen **GETRENNT** (englisch: **SEPARATED BY**) und **SEMIKOLON** (englisch: **SEMICOLON**) markiert sind. Würden Sie zusätzlich die Option **KOMMA** (englisch: **COMMA**) markieren, würden die Zahlen mit Nachkommastellen auf zwei Spalten aufgeteilt. Anschließend werden die Daten korrekt dargestellt wie in MS Excel unter Windows.

Gegebenenfalls müssen Sie in der Liste **ZEICHENSATZ** (englisch: **CHARACTER SET**) den Eintrag **SYSTEM** auswählen.

8.3.4 CSV-Datei lesen

Zum erfolgreichen Import einer CSV-Datei müssen die Struktur und die Datentypen der Daten bekannt sein. Für das nachfolgende Beispiel wird die CSV-Datei aus dem vorherigen Beispiel in MS Excel um einen Datensatz ergänzt und abgespeichert, siehe [Abbildung 8.2](#).

	A	B	C	D	E	F
1	Maier	Hans	6714	3500	15.03.1962	
2	Schmitz	Peter	81343	3750	12.04.1958	
3	Mertens	Julia	2297	3621,5	30.12.1959	
4	Weber	Jürgen	4711	2900	12.08.1976	
5						

Abbildung 8.2 CSV-Datei in MS Excel, ergänzt

Die Datei wird mit dem folgenden Programm gelesen. Die Daten der Datensätze werden einzeln ausgegeben.

```
import sys
try:
    d = open("daten.csv")
except:
    print("Datei nicht geöffnet")
    sys.exit(0)

tx = d.read()
d.close()
li = tx.split("\n")

for zeile in li:
    if zeile:
        ds = zeile.split("; ")
        gh = str(ds[3]).replace(",",".")
        print(f"{ds[0]} {ds[1]} {ds[2]} {gh} {ds[4]}")
```

Listing 8.7 Datei »lesen_csv.py«

Das Programm gibt alle vier Datensätze aus:

```
Maier Hans 6714 3500 15.03.1962
Schmitz Peter 81343 3750 12.04.1958
Mertens Julia 2297 3621.5 30.12.1959
Weber Jürgen 4711 2900 12.08.1976
```

Zunächst wird der gesamte Text der Datei mit der Methode `read()` in einer Zeichenkette gespeichert. Sie wird mithilfe der Methode `split()` anhand des Zeichens für Zeilenende in eine Liste von Zeichenketten zerlegt. Jede Zeichenkette enthält eine

Zeile aus der Datei. Die weitere Verarbeitung wird nur vorgenommen, falls die Zeile nicht leer ist.

Jede Zeile wird, wiederum mithilfe der Methode `split()`, anhand des Semikolons in eine Liste mit den einzelnen Daten des Datensatzes zerlegt.

Handelt es sich um eine Zahl mit Nachkommastellen, wird das Dezimalkomma mithilfe der Methode `replace()` in einen Punkt verwandelt. Zuletzt werden die Daten, durch Leerzeichen voneinander getrennt, ausgegeben.

8.4 Dateien mit festgelegter Struktur

Wird eine Datei *formatiert* beschrieben, ist damit ihre Struktur festgelegt. Es ist genau bekannt, an welcher Stelle welche Information steht. Somit können die gewünschten Informationen direkt aus der Datei gelesen werden, ohne die gesamte Datei Zeile für Zeile einlesen zu müssen.

Außerdem ist es möglich, bestimmte Informationen *punktuell* zu verändern, ohne den restlichen Inhalt der Datei zu beeinflussen. Beide Möglichkeiten werden in den folgenden Abschnitten vorgeführt.

8.4.1 Formatiertes Schreiben

Das formatierte Schreiben in eine Datei ähnelt einer formatierten Ausgabe auf dem Bildschirm. Es werden Datensätze mit gleicher Länge und gleichem Aufbau gebildet. Optional können Sie nach jedem Datensatz ein Zeilenende-Zeichen einfügen, um die Datei in einem Editor leichter lesbar zu machen. Sie können die Datei auch mithilfe des Editors verändern, solange Sie sich an die festgelegte Struktur halten.

Im nachfolgenden Programm wird die Tabelle der Personen aus Abschnitt 8.3.3, »CSV-Datei schreiben«, formatiert in einer Datei gespeichert:

```
import sys
try:
    d = open("formatiert.txt", "w")
except:
    print("Datei nicht geöffnet")
    sys.exit(0)

li = [[ "Maier", "Hans", 6714, 3500, "15.03.1962"] ,
      [ "Schmitz", "Peter", 81343, 3750, "12.04.1958"] ,
      [ "Mertens", "Julia", 2297, 3621.5, "30.12.1959"] ]
for ds in li:
    d.write(f"{ds[ 0 ] :12} {ds[ 1 ] :12} {ds[ 2 ] :6} {ds[ 3 ] :8.2f} {ds[ 4 ] :>11}\n")
d.close()
```

Listing 8.8 Datei »schreiben_formatiert.py«

Anschließend sieht die Datei *formatiert.txt* wie folgt aus:

Maier	Hans	6714 3500.00 15.03.1962
Schmitz	Peter	81343 3750.00 12.04.1958
Mertens	Julia	2297 3621.50 30.12.1959

Listing 8.9 Datei »formatiert.txt«

Die Datei wird zum Schreiben geöffnet. Die Datensätze werden mithilfe von formatierten String-Literalen ausgegeben, wie in [Abschnitt 5.2.2](#), »Formatierung von Zahlen mit Nachkommastellen«, beschrieben. Zusätzlich wird jeweils ein `\n` als Zeilenende angefügt.

Jeder Datensatz hat unter Windows dieselbe Größe von 51 Byte. Sie ergibt sich aus der Summe der einzelnen Formate und den zwei Byte für das Zeilenende: $12 + 12 + 6 + 8 + 11 + 2 = 51$. Die gesamte Datei hat eine Größe von $51 \times 3 = 153$ Byte.

Unterschiede unter Ubuntu Linux oder macOS

Das Zeichen für das Zeilenende hat eine Größe von 1 Byte. Daher hat jeder Datensatz 50 Byte und die Datei 150 Byte.

8.4.2 Lesen an beliebiger Stelle

Die Methode `seek()` ermöglicht die Änderung der aktuellen Lese- oder Schreibposition innerhalb der Datei. Sie kann mit einem oder zwei Parametern aufgerufen werden. Wird sie mit einem Parameter aufgerufen, handelt es sich dabei um den Abstand in Byte, gemessen vom Dateianfang. Der Dateianfang entspricht dem Wert 0.

Die gewünschte Position wird direkt erreicht, ohne Lesen der Informationen, die vor der Position stehen. Ausgehend von der erreichten Position kann anschließend gelesen oder geschrieben werden.

Die Methode `seek()` wird hier zusammen mit einer Variante der Methode `read()` eingeführt. Diese Methode gestattet auch das Einlesen einer bestimmten Anzahl von Bytes. Ein Beispiel:

```
import sys
try:
    d = open("formatiert.txt")
except:
    print("Datei nicht geöffnet")
    sys.exit(0)

dslaenge = 51
for i in range(3):
    d.seek(dselaenge*i)
    name = d.read(12).strip()
    d.seek(30 + dselaenge*i)
    gehalt = float(d.read(8))
    print(name, gehalt)

d.close()
```

Listing 8.10 Datei »lesen_beliebig.py«

Die Ausgabe lautet:

```
Maier 3500.0
Schmitz 3750.0
Mertens 3621.5
```

Die Datei wird zunächst zum Lesen geöffnet. Anschließend werden mithilfe der Methode `seek()` nacheinander Positionen in allen Datensätzen erreicht:

- An den Positionen $0 \times 51 = 0$, $1 \times 51 = 51$ und $2 \times 51 = 102$ werden die nächsten 12 Byte mit dem Namen der Person gelesen. Mithilfe der Methode `strip()` werden die Leerzeichen nach dem Namen entfernt.
- An den Positionen $(0 \times 51 + 30) = 30$, $(1 \times 51 + 30) = 81$ und $(2 \times 51 + 30) = 132$ werden die nächsten 8 Byte mit dem Gehalt der Person gelesen. Mithilfe der Funktion `float()` wird das Gelesene in eine Zahl umgewandelt.

Unterschiede unter Ubuntu Linux und macOS

Aufgrund der Größe von 1 Byte für das Zeilenende-Zeichen muss `dselaenge` den Wert 50 haben.

8.4.3 Schreiben an beliebiger Stelle

Sie können eine Datei im Modus `r+` öffnen. Anschließend können Sie sie sowohl lesen als auch ändern. Im nachfolgenden Programm werden die Gehälter der Personen gelesen, geändert und neu gespeichert.

```
import sys
try:
    d = open("formatiert.txt", "r+")
except:
    print("Datei nicht geöffnet")
    sys.exit(0)

dselaenge = 51
for i in range(3):
    d.seek(30 + dselaenge*i)
    gehalt = float(d.read(8))
    gehalt = round(gehalt * 1.05, 2)
    d.seek(30 + dselaenge*i)
    d.write(f"gehalt:{8.2f}")

d.close()
```

Listing 8.11 Datei »schreiben_beliebig.py«

Anschließend sieht die Datei *formatiert.txt* wie folgt aus:

Maier	Hans	6714	3675.00	15.03.1962
Schmitz	Peter	81343	3937.50	12.04.1958
Mertens	Julia	2297	3802.58	30.12.1959

Listing 8.12 Datei »formatiert.txt«

Das Gehalt jeder Person wird wie im vorherigen Programm gelesen. Anschließend wird es um 5 % erhöht. Der Wert wird auf zwei Stellen nach dem Komma gerundet. Mithilfe von `seek()` wird wieder die Position erreicht, an der das Gehalt steht. Dann wird das geänderte Gehalt formatiert in die Datei geschrieben.

Unterschiede unter Ubuntu Linux und macOS

Aufgrund der Größe von 1 Byte für das Zeilenende-Zeichen muss `dselaenge` wiederum den Wert 50 haben.

8.5 Serialisierung mit »pickle«

Das Modul `pickle` dient zur Serialisierung und Deserialisierung von Objekten mithilfe von Python. Das können Objekte der eingebauten Datentypen oder Objekte eigener Klassen sein.

Bei der Serialisierung werden die Objekte als Bytefolge in einer Datei gespeichert. Bei der Deserialisierung werden die Objekte wieder aus einer Datei geladen. Beim Speichern wird der Typ des Objekts hinzugefügt, sodass er beim Laden bekannt ist.

Neben den Beispielen in diesem Abschnitt finden Sie in [Abschnitt 8.11, »Spiel, Version mit Highscore-Datei«](#), eine weitere Version des bereits bekannten Kopfrechenspiels. Die Daten des Spielers (Name und Zeit) werden mithilfe der Serialisierung dauerhaft gespeichert, sodass eine Highscore-Liste geführt werden kann.

8.5.1 Objekte in Datei schreiben

Zum Serialisieren von Objekten dient die Funktion `dump()`. Im folgenden Beispiel wird ein Objekt, das Elemente der eingebauten Datentypen enthält, und ein Objekt einer eigenen Klasse in einer Datei gespeichert:

```
import pickle, sys
class Fahrzeug:
    def __init__(self, bez, ge):
        self.bezeichnung = bez
        self.geschwindigkeit = ge
    def __str__(self):
        return f'{self.bezeichnung} {self.geschwindigkeit} km/h'

try:
    d = open("pickle_datei.bin", "wb")
except:
    print("Datei nicht geöffnet")
    sys.exit(0)

tu = [ 4, "abc", 8], -12.5, {"a":1, "b":1}, set([ 8, 5, 2, 5])
pickle.dump(x, tu)
```

```
opel = Fahrzeug("Opel Admiral", 40)
pickle.dump(opel, d)

pickle.dump(3, d)
pickle.dump("Berlin", d)
pickle.dump("Hamburg", d)
pickle.dump("Dortmund", d)

d.close()
```

Listing 8.13 Datei »pickle_schreiben.py«

Das Programm beginnt mit der bekannten Definition der Klasse `Fahrzeug`. Die Datei wird im Modus `wb` geöffnet, also zum Schreiben im Binärmodus. Zusätzlich habe ich die Dateiendung `.bin` gewählt.

Es wird ein Tupel erstellt, das aus einer Liste, einer Zahl, einem Dictionary und einem Set besteht. Die Liste enthält unter anderem eine Zeichenkette. Das Tupel wird mithilfe der Funktion `dump()` binär in der Datei gespeichert. Der erste Parameter ist der Name des Objekts, der zweite Parameter ist das Dateiobjekt. Außerdem wird ein Objekt der Klasse `Fahrzeug` erzeugt und ebenfalls binär in der Datei gespeichert.

Möchten Sie eine bestimmte Anzahl von Objekten speichern, hier zum Beispiel drei Zeichenketten, so speichern Sie zunächst diese Anzahl binär in der Datei, anschließend die Objekte selbst. Auf diese Weise kann später die richtige Anzahl von Objekten wieder geladen werden (siehe auch nachfolgend den [Abschnitt 8.5.2](#)).

Hinweis

Ist die Datei bereits vorhanden, wird sie überschrieben. Zum Anhängen von Objekten können Sie die Datei im Modus `ab` öffnen. Die Datei kann nur mithilfe der genannten Methoden geschrieben oder gelesen werden.

8.5.2 Objekte aus Datei lesen

Objekte, die mit der Funktion `dump()` in einer Datei gespeichert wurden, können mit der Funktion `load()` wieder in der gleichen Reihenfolge aus der Datei übernommen werden. Handelt es sich um ein Objekt einer eigenen Klasse, muss deren Definition bekannt sein.

Nachfolgend werden die Objekte, die mithilfe des vorherigen Programms serialisiert wurden, wieder deserialisiert und damit übernommen.

```
import pickle, sys
class Fahrzeug:
    def __init__(self, bez, ge):
        self.bezeichnung = bez
        self.geschwindigkeit = ge
    def __str__(self):
        return f"{self.bezeichnung} {self.geschwindigkeit} km/h"

try:
    d = open("pickle_datei.bin", "rb")
except:
    print("Datei nicht geöffnet")
    sys.exit(0)

tu = pickle.load(d)
print(tu)

opel = pickle.load(d)
opel.geschwindigkeit += 20
print(opel)

anzahl = pickle.load(d)
for i in range(anzahl):
    print(pickle.load(d))

d.close()
```

Listing 8.14 Datei »pickle_leSEN.py«

Es wird diese Ausgabe erzeugt:

```
([4, 'abc', 8], -12.5, {'a': 1, 'b': 1}, {8, 2, 5})
Opel Admiral 60 km/h
Berlin
Hamburg
Dortmund
```

Das Programm beginnt mit der Definition der eigenen Klasse `Fahrzeug`. Die Datei wird im Modus `rb` geöffnet, also zum Lesen im Binärmodus.

Mithilfe der Funktion `load()` aus dem Modul `pickle` wird zunächst das Tupel geladen. Alle Elemente des Tupels werden mitsamt ihren Datentypen richtig erkannt. Es folgt das Laden des Objekts der Klasse `Fahrzeug`. An der Veränderung sehen Sie, dass auch dieses Objekt mitsamt seiner Klasse richtig erkannt wurde.

Anschließend wird die Anzahl der nachfolgenden Objekte als Zahl geladen. Damit ist die Anzahl der zu ladenden Objekte für die `for`-Schleife bekannt. Alle Objekte werden zur Kontrolle auf dem Bildschirm ausgegeben.

8.6 Datenaustausch mit JSON

Bei JSON (JavaScript Object Notation) handelt es sich um ein weit verbreitetes Format zum Austausch von Daten. Im Unterschied zur Serialisierung mit `pickle` werden JSON-Daten in lesbarer Form gespeichert und können mithilfe unterschiedlicher Programmiersprachen gelesen werden.

Ein einzelnes Objekt von einem der nachfolgenden Datentypen kann direkt als JSON-Objekt oder in einem JSON-Objekt gespeichert werden: Zahl, Zeichenkette, Liste, Tupel und Dictionary.

Folgende Besonderheiten sind zu beachten:

- Ein Tupel wird als Liste gespeichert, kann aber anschließend wieder aus dieser Liste erzeugt werden.
- Ein Set kann nicht direkt als JSON-Objekt oder in einem JSON-Objekt gespeichert werden, aber zum Beispiel mithilfe einer Liste.
- Ein Objekt eines eigenen Datentyps kann mithilfe der besonderen Eigenschaft `__dict__` als JSON-Objekt oder in einem JSON-Objekt gespeichert und anschließend wieder neu erzeugt werden.

8.6.1 JSON-Objekte in Datei schreiben

Im nachfolgenden Programm wird ein Tupel erstellt, das aus einer Liste, einer Zahl, einem Dictionary, einem Set und einem Objekt einer eigenen Klasse besteht. Die Liste enthält unter anderem eine Zeichenkette. Anschließend wird das Tupel als JSON-Objekt in einer Datei gespeichert:

```
import json, sys
```

```

class Fahrzeug:
    def __init__(self, bez, ge):
        self.bezeichnung = bez
        self.geschwindigkeit = ge
    def __str__(self):
        return f"{self.bezeichnung} {self.geschwindigkeit} km/h"

try:
    d = open("json_datei.json", "w")
except:
    print("Dateizugriff nicht erfolgreich")
    sys.exit(0)

s = set([8, 5, 2, 5])
ls = []
for element in s:
    ls.append(element)

opel = Fahrzeug("Opel Admiral", 40)

tu = [4, "abc", 8], -12.5, {"a": 1, "b": 1}, ls, opel.__dict__
json.dump(tu, d)
d.close()

```

Listing 8.15 Datei »json_schreiben.py«

Die Elemente des Sets werden vor der Aufnahme in das Tupel in einer Liste gespeichert. Im Tupel wird statt des Objekts der eigenen Klasse seine besondere Eigenschaft `__dict__` gespeichert. Die Funktion `dump()` aus dem Modul `json` schreibt das Tupel als einzelnes JSON-Objekt in eine Datei, die zuvor zum Schreiben geöffnet wurde. Als Letztes wird die Datei geschlossen. Sie hat das folgende lesbare Aussehen:

```
[[4, "abc", 8], -12.5, {"a": 1, "b": 1}, [8, 2, 5],
 {"bezeichnung": "Opel Admiral", "geschwindigkeit": 40}]
```

Listing 8.16 Datei »json_datei.json«

8.6.2 JSON-Objekte aus Datei lesen

Im nächsten Programm wird ein Tupel als einzelnes JSON-Objekt aus einer Datei gelesen und ausgegeben:

```

import json, sys

class Fahrzeug:
    def __init__(self, bez, ge):
        self.bezeichnung = bez
        self.geschwindigkeit = ge
    def __str__(self):

```

```

        return f"{self.bezeichnung} {self.geschwindigkeit} km/h"

try:
    d = open("json_datei.json", "r")
except:
    print("Dateizugriff nicht erfolgreich")
    sys.exit(0)

li = json.load(d)
d.close()

s = set()
for element in li[3]:
    s.add(element)

opel = Fahrzeug(li[4][ "bezeichnung"] , li[4][ "geschwindigkeit"] )

tu = li[0] , li[1] , li[2] , s, opel.__str__()
print(tu)

```

Listing 8.17 Datei »json_leSEN.py«

Die Funktion `load()` aus dem Modul `json` liest ein JSON-Objekt aus einer Datei, die zuvor zum Lesen geöffnet wurde. Als Nächstes wird die Datei geschlossen. Das JSON-Objekt enthält eine Liste, aus deren Elementen das ursprüngliche Tupel aus dem vorherigen Programm erstellt wird.

Das Set ist zunächst leer. Ihm werden die Elemente der Unterliste hinzugefügt, die aus dem ursprünglichen Set erzeugt wurde. Das Objekt der eigenen Klasse wird mithilfe des Konstruktors aus dem Dictionary erzeugt, das mithilfe der besonderen Eigenschaft `__dict__` erzeugt wurde.

Es folgt die Ausgabe des Tupels:

```
([4, 'abc', 8], -12.5, {'a': 1, 'b': 1}, {8, 2, 5}, 'Opel Admiral 40 km/h')
```

8.7 Bearbeitung mehrerer Dateien

Bisher wurde immer nur eine einzelne Datei bearbeitet, deren Name bekannt war. Stehen Sie vor der Aufgabe, mehrere Dateien aus einem Verzeichnis zu bearbeiten, deren Anzahl und Namen unbekannt sind, können Sie sich der Funktion `glob()` aus dem Modul `glob` und der Funktion `scandir()` aus dem Modul `os` bedienen. Sie liefern jeweils eine Liste, die einen Zugriff auf die einzelnen Dateien ermöglicht.

8.7.1 Funktion »glob.glob()«

Im folgenden Beispiel werden zunächst die Namen bestimmter Dateien aus dem aktuellen Verzeichnis mithilfe der Funktion `glob()` in einer Liste gespeichert. Diese Dateien werden der Reihe nach geöffnet und durchsucht. Wird in einer der Dateien ein bestimmter Suchtext gefunden, so wird ihr Name jeweils ausgegeben.

```
import glob
dateiliste = glob.glob("schr*.py")
for datei in dateiliste:
    try:
        d = open(datei)
    except:
        print("Datei nicht geöffnet")
        continue
    gesamtertext = d.read()
    d.close()
    if gesamtertext.find("Schmitz") != -1:
        print(datei)
```

Listing 8.18 Datei »datei_liste.py«

Die Ausgabe lautet:

```
schreiben_csv.py
schreiben_formatiert.py
```

Der Parameter der Funktion `glob()` ist eine Zeichenkette, die als Filter für die Dateisuche dient. Die Zeichenkette kann auch einen der Platzhalter `*` (für mehrere beliebige Zeichen) oder `?` (für ein

einzelnes beliebiges Zeichen) enthalten. Hier wird nach Dateien gesucht, deren Name mit `schr` beginnt und mit `.py` endet.

Jedes Element der erzeugten Liste entspricht einem Dateinamen. Jede dieser Dateien wird geöffnet. War das Öffnen nicht erfolgreich, wird mit der nächsten Datei fortgefahren. Der vollständige Inhalt der jeweiligen Datei wird mithilfe der Methode `read()` in eine Zeichenkette eingelesen. Anschließend wird die Datei wieder geschlossen.

Die Zeichenkette wird mithilfe der Methode `find()` durchsucht. Wird der Suchtext gefunden, so wird der Dateiname ausgegeben.

Seit Python 3.5 können Sie mit der Funktion `glob.glob()` und der Zeichenfolge `**` auch rekursiv einen gesamten Verzeichnisbaum durchsuchen. In den folgenden Vergleichsbeispielen wird eine Liste aller Dateien mit der Endung `.py` geliefert, deren Name mit `c` beginnt:

- `glob.glob("c*.py")` durchläuft nur das aktuelle Verzeichnis.
- `glob.glob("**/c*.py")` durchläuft nur direkte Unterverzeichnisse.
- `glob.glob("**/c*.py", recursive=True)` durchläuft das aktuelle Verzeichnis, direkte Unterverzeichnisse, deren Unterverzeichnisse usw., also den gesamten Verzeichnisbaum.

8.7.2 Funktion »os.scandir()«

Seit Python 3.5 gibt es mit der Funktion `scandir()` aus dem Modul `os` eine Alternative zum Durchlaufen eines Verzeichnisses. Das nachfolgende Programm liefert dasselbe Ergebnis wie oben:

```
import os
eintrag_iterator = os.scandir(".")
for eintrag in eintrag_iterator:
    if eintrag.name.startswith("schr") and eintrag.name.endswith(".py"):
        try:
            d = open(eintrag)
        except:
```

```
    print("Datei nicht geöffnet")
    continue

    gesamtertext = d.read()
    d.close()
    if gesamtertext.find("Schmitz") != -1:
        print(eintrag.name)
eintrag_iterator.close()
```

Listing 8.19 Datei »datei_liste_scandir.py«

Die Funktion `scandir()` erwartet den Namen eines Pfads als Parameter. Mit `.` geben Sie das aktuelle Verzeichnis an. Die Funktion liefert ein Iterator-Objekt des Typs `os.DirEntry`, mit dessen Hilfe alle Einträge des Verzeichnisses durchlaufen werden.

Die Eigenschaft `name` des `DirEntry`-Objekts enthält den Namen des Verzeichniseintrags. Beginnt dieser mit `schr` und endet er mit `.py`, wird die betreffende Datei wie oben weiterbearbeitet.

Seit Python 3.6 haben Sie die Möglichkeit, den Iterator mithilfe von `close()` zu schließen und die genutzten Ressourcen wieder freizugeben.

8.8 Informationen über Dateien

Die Funktion `stat()` aus dem Modul `os` liefert eine Reihe von Informationen über Dateien in Form eines Tupels.

Ein Beispiel:

```
import os, time
tu = os.stat("formatiert.txt")
print("Anzahl Byte:", tu[ 6 ])
print("Letzter Zugriff:",
      time.strftime("%d.%m.%Y %H:%M:%S", time.localtime(tu[ 7 ])))
print("Letzte Modifikation:",
      time.strftime("%d.%m.%Y %H:%M:%S", time.localtime(tu[ 8 ])))
```

Listing 8.20 Datei »datei_info.py«

Die Ausgabe lautet:

```
Anzahl Byte: 153
Letzter Zugriff: 21.11.2021 05:54:23
Letzte Modifikation: 20.11.2021 16:25:13
```

Element 6 des Tupels enthält die Größe der Datei in Byte, Element 7 den Zeitpunkt des letzten Zugriffs auf die Datei und Element 8 den Zeitpunkt der letzten Änderung der Datei. Die beiden Zeitangaben werden mithilfe der Funktionen `localtime()` umgewandelt und mithilfe der Funktion `strftime()` formatiert.

Unterschiede unter Ubuntu Linux und macOS

Die Datensätze haben nur eine Länge von 50 statt 51 Zeichen. Damit ergibt sich eine Dateigröße von 150 Byte.

8.9 Dateien und Verzeichnisse verwalten

Die Module `os` und `shutil` bieten weitere Funktionen zur Verwaltung von Dateien und Verzeichnissen. Im folgenden Beispiel werden Dateien kopiert, umbenannt und gelöscht.

```
import sys, shutil, os, glob
print(glob.glob("le*.txt"))

if not os.path.exists("lesen.txt"):
    print("Datei nicht vorhanden")
    sys.exit(0)

shutil.copy("lesen.txt","lesen_kopie.txt")
print(glob.glob("le*.txt"))

try:
    shutil.move("lesen_kopie.txt","lesen_neu.txt")
except:
    print("Fehler beim Umbenennen")
print(glob.glob("le*.txt"))

try:
    os.remove("lesen_neu.txt")
except:
    print("Fehler beim Entfernen")
print(glob.glob("le*.txt"))
```

Listing 8.21 Datei »datei_verwalten.py«

Unter Windows sieht die Ausgabe wie folgt aus:

```
['lesen.txt']
['lesen.txt', 'lesen_kopie.txt']
['lesen.txt', 'lesen_neu.txt']
['lesen.txt']
```

Die erste Zeile der Ausgabe liefert die ursprüngliche Liste derjenigen Dateien, die mit `le` beginnen und mit `.txt` enden.

Mithilfe der Funktion `exists()` aus dem Modul `os.path` wird geprüft, ob eine bestimmte Datei existiert, die kopiert werden soll. Existiert sie nicht, wird das Programm vorzeitig beendet.

Anschließend wird die Datei `lesen.txt` mithilfe der Funktion `copy()` aus dem Modul `shutil` kopiert. Die Kopie heißt

lesen_kopie.txt. Die zweite Zeile der Ausgabe liefert die geänderte Liste der Dateien.

Nun wird die Datei *lesen_kopie.txt* mithilfe der Funktion `move()` aus dem Modul `shutil` in *lesen_neu.txt* umbenannt. Die nächste Zeile der Ausgabe liefert wiederum die geänderte Liste der Dateien. Die Funktion `move()` kann auch zum Verschieben einer Datei in ein anderes Verzeichnis genutzt werden.

Als Letztes wird die Datei *lesen_kopie.txt* mithilfe der Funktion `remove()` aus dem Modul `os` wieder gelöscht. Die letzte Zeile der Ausgabe liefert die letzte Liste der Dateien.

8.10 Beispielprojekt Morsezeichen

In diesem Abschnitt sehen Sie ein Beispielprojekt, in dem auf Dateien, externe Module, Zeichenketten, Dictionarys, einige Funktionen und das Modul `winsound` zur Ausgabe von Tönen (siehe [Abschnitt 7.5](#), »Audioausgabe«) zugegriffen wird.

Ein Beispieltext wird in Morsecode umgesetzt. Er besteht aus einzelnen Morsezeichen und Pausen. Ein Morsezeichen steht für ein einzelnes Zeichen eines Texts und setzt sich aus einer Folge von kurzen und langen Signalen zusammen. Damit werden bereits seit langer Zeit Texte als eine Folge von Ton-, Funk- oder Lichtsignalen übermittelt.

8.10.1 Morsezeichen aus Datei lesen

Zunächst werden die verschiedenen Zeichen und das jeweils zugehörige Morsezeichen aus der Datei `morsen.txt` in ein Dictionary gelesen. In der Datei wird ein kurzes Signal durch einen Punkt dargestellt, ein langes Signal durch einen Strich. Die einzelnen Zeichen und das dazugehörige Morsezeichen sind jeweils durch ein Leerzeichen getrennt:

```
A .-
B -...
C -..
D -..
E .
```

Listing 8.22 Datei »morsen.txt«, Ausschnitt

Zum Lesen dient eine Funktion im nachfolgenden Modul `morsen`, die anderen Python-Programmen zur Verfügung gestellt werden kann:

```
import sys
def leseCode():
    try:
        d = open("morsen.txt")
    except:
```

```

        print("Datei nicht geöffnet")
        sys.exit(0)
    allezeilen = d.readlines()
    d.close()
    code = {}
    for zeile in allezeilen:
        worte = zeile.split()
        code[ worte[ 0] ] = worte[ 1]
    for i in range(97,123):
        code[ chr(i)] = code[ chr(i-32)]
    return code

```

Listing 8.23 Datei »morsen.py«

In der Funktion `leseCode()` werden zuerst alle Zeilen der Datei *morsen.txt* gelesen. Anschließend wird ein leeres Dictionary erzeugt.

Jede Zeile wird anhand des Leerzeichens in zwei Zeichenketten zerlegt. Die erste Zeichenkette enthält ein Unicode-Zeichen, die zweite Zeichenkette das zugehörige Morsezeichen. Das Unicode-Zeichen wird als Schlüssel für den Eintrag des Morsezeichens im Dictionary genutzt.

Beim Morsen wird nicht zwischen Groß- und Kleinschreibung unterschieden. Daher wird den Dictionary-Elementen für die Kleinbuchstaben das gleiche Morsezeichen zugewiesen wie den entsprechenden Elementen für die Großbuchstaben.

Als Ergebnis der Funktion wird das Dictionary zurückgeliefert.

8.10.2 Ausgabe auf dem Bildschirm

In diesem Abschnitt werden die Morsezeichen eines Beispieltexts auf dem Bildschirm ausgegeben. Die einzelnen Morsezeichen werden jeweils durch ein Leerzeichen getrennt. Es folgt das Programm:

```

import sys, morsen
def schreibeCode(text, code):
    for zeichen in text:
        try:
            print(code[ zeichen], end=" ")
        except KeyError:
            print(" ", end=" ")
    print()

```

```
code = morsen.leseCode()
schreibeCode("Hallo Welt", code)
```

Listing 8.24 Datei »morsen_bildschirm.py«

Die Ausgabe des Beispieltextes "Hallo Welt" in Morsezeichen:

```
.... .- .-. .-.. --- .-- . .-.. -
```

Im Hauptprogramm wird die Funktion `leseCode()` aus dem oben beschriebenen Modul `morsen` aufgerufen. Sie liefert ein Dictionary zurück. Anschließend wird die Funktion `schreibeCode()` mit einem Beispieltext und diesem Dictionary aufgerufen.

In der Funktion `schreibeCode()` wird ein übergebener Beispieltext in Morsezeichen codiert und ausgegeben. Gibt es zu einem Zeichen keine Entsprechung im Dictionary, tritt ein `KeyError` auf. In diesem Fall wird ein Leerzeichen ausgegeben.

8.10.3 Ausgabe mit Tonsignalen

In diesem Abschnitt werden die Morsezeichen eines Beispieltextes als Tonsignale ausgegeben. Bevor Sie das nachfolgende Programm starten, sollten Sie den Lautsprecher Ihres Windows-PCs einschalten:

```
import sys, morsen, time, winsound
def tonCode(text, code):
    signalDauer = {"." : 200, "-" : 600}
    signalPause = 0.2
    zeichenPause = 0.6
    wortPause = 1.4

    alleWorte = text.split()
    for w in range(len(alleWorte)):
        wort = alleWorte[w]
        for z in range(len(wort)):
            zeichen = wort[z]
            print(zeichen, end="")
            try:
                alleSignale = code[zeichen]
                for s in range(len(alleSignale)):
                    signal = alleSignale[s]
                    winsound.Beep(800, signalDauer[signal])
                    if s < len(alleSignale)-1:
                        time.sleep(signalPause)
            if z < len(wort)-1:
```

```

        time.sleep(zeichenPause)
    except KeyError:
        pass
    if w < len(alleWorte)-1:
        print(" ", end="")
        time.sleep(wortPause)

code = morsen.leseCode()
tonCode("Hallo Welt", code)

```

Listing 8.25 Datei »morsen_ton.py«

In der Funktion `tonCode()` wird ein übergebener Beispieltext in Morsezeichen codiert und als Folge von Tonsignalen ausgegeben. Dazu wird zunächst ein weiteres Dictionary angelegt. Darin wird die Signaldauer für ein kurzes Signal mit 200 Millisekunden festgelegt. Ein langes Signal muss dreimal so lang sein. Daher bekommt es eine Länge von 600 Millisekunden.

Nach jedem Signal folgt eine Signalpause in der Dauer eines kurzen Signals, nach jedem Zeichen eine Zeichenpause in der Dauer eines langen Signals und nach jedem Wort eine Wortpause in einer Dauer von sieben kurzen Signalen.

Der gesamte Text wird anhand der Leerzeichen in einzelne Wörter zerlegt. Jedes Wort wird in einzelne Zeichen zerlegt. Zu jedem Zeichen wird der zugehörige Eintrag im Dictionary der Morsezeichen gesucht. Jedes gefundene Morsezeichen wird in einzelne Signale zerlegt. Jedes Signal wird mithilfe der Funktion `Beep()` aus dem Modul `winsound` als Ton ausgegeben.

Gibt es zu einem Zeichen keine Entsprechung im Dictionary, tritt ein `KeyError` auf. In diesem Fall wird das Zeichen ignoriert.

Die Funktion `sleep()` aus dem Modul `time` sorgt für die Signal-, Zeichen- und Wortpausen.

Im Hauptprogramm wird die Funktion `leseCode()` aus dem Modul `morsen` aufgerufen. Sie liefert ein Dictionary zurück. Anschließend wird die Funktion `tonCode()` mit einem Beispieltext und diesem Dictionary aufgerufen.

Unterschiede in Ubuntu Linux und macOS

Das Modul `winsound` steht nicht zur Verfügung.

8.11 Spiel, Version mit Highscore-Datei

Es folgt eine weitere Version des bereits bekannten Kopfrechenspiels. Die Daten des Spielers (Name und Zeit) werden mithilfe der Serialisierung (siehe [Abschnitt 8.5](#), »Serialisierung mit ›pickle‹«) dauerhaft gespeichert, sodass eine Highscore-Liste geführt werden kann.

Es wird zunächst nach dem Namen des Spielers gefragt. Anschließend bekommt der Spieler fünf Additionsaufgaben mit Zahlen aus dem Bereich von 10 bis 30 gestellt. Pro Aufgabe hat er nur einen Versuch. Ungültige Eingaben oder falsche Ergebnisse werden mit dem Text `Falsch` kommentiert. Zuletzt wird die Gesamtzeit angegeben, die der Spieler für die Lösungsversuche benötigt hat.

Löst der Spieler alle fünf Aufgaben richtig, werden sein Name und die benötigte Zeit in eine Highscore-Liste aufgenommen, die in einer Datei abgespeichert wird.

8.11.1 Eingabebeispiel

In [Abbildung 8.3](#) sehen Sie einen typischen Durchlauf des Programms.

Der Benutzer schaut sich zuerst die Highscores an. Dort gibt es bereits einige Einträge. Anschließend spielt er eine Runde, löst alle fünf Aufgaben richtig, und sein Name erscheint ebenfalls im Highscore. Als Letztes verlässt er das Programm.

```

Bitte eingeben (0: Ende, 1: Highscores, 2: Spielen): 1
P. Name      Zeit
1. Ole        8.03 sec
2. Rudi       8.36 sec
3. Gerd       9.36 sec
4. Ben        12.42 sec
5. Tom        12.75 sec
Bitte eingeben (0: Ende, 1: Highscores, 2: Spielen): 2
Bitte geben Sie Ihren Namen ein (max. 10 Zeichen): Paul
Aufgabe 1 von 5: 20 + 30 : 50
*** Richtig ***
Aufgabe 2 von 5: 29 + 29 : 58
*** Richtig ***
Aufgabe 3 von 5: 23 + 21 : 44
*** Richtig ***
Aufgabe 4 von 5: 29 + 10 : 39
*** Richtig ***
Aufgabe 5 von 5: 27 + 30 : 57
*** Richtig ***
Ergebnis: 5 von 5 in 9.48 Sekunden, Highscore
P. Name      Zeit
1. Ole        8.03 sec
2. Rudi       8.36 sec
3. Gerd       9.36 sec
4. Paul       9.48 sec
5. Ben        12.42 sec
6. Tom        12.75 sec
Bitte eingeben (0: Ende, 1: Highscores, 2: Spielen): 0

```

Abbildung 8.3 Eingabebeispiel

8.11.2 Aufbau des Programms

Das Programm enthält ein Hauptprogramm und vier Funktionen. Im Hauptprogramm wird innerhalb einer Endlosschleife ein Hauptmenü aufgerufen. Darin kann sich der Spieler entweder die Highscores anzeigen lassen, spielen oder das Programm beenden.

Es folgen die vier Funktionen des Programms:

- `hs_leSEN()`: Lesen der Highscores aus der Datei in eine Liste
- `hs_anzeigen()`: Anzeigen der Highscore-Liste auf dem Bildschirm
- `hs_schreiben()`: Schreiben der Highscore-Liste in die Datei
- `spiel()`: Stellen der Aufgaben, Lösen der Aufgaben, Einfügen der ermittelten Zeit in die Highscore-Liste

8.11.3 Code des Programms

Im nachfolgenden Listing sehen Sie den Beginn des Programms mit der Funktion zum Lesen der Highscore-Daten aus der Datei:

```
import random, time, glob, pickle

def hs_leSEN():
    global hs_liste
    if not glob.glob("highscore.bin"):
        hs_liste = []
        return
    d = open("highscore.bin", "rb")
    hs_liste = pickle.load(d)
    d.close()
```

Listing 8.26 Datei »spiel_datei.py«, Teil 1 von 5

Es werden die Module `random` (zur Erzeugung der Zufallszahlen), `time` (zur Zeitmessung), `glob` (zur Prüfung der Datei) und `pickle` (zum Zugriff auf die Datei) benötigt.

Nach dem Einlesen aus der Datei stehen in der Liste `hs_liste` die Namen und Ergebniszeiten der Spielerinnen und Spieler. Da die Liste in dieser Funktion erstmalig benutzt, aber im gesamten Programm benötigt wird, wird sie hier mit `global` im globalen Namensraum bekannt gemacht.

Ist die Datei nicht vorhanden, wird eine leere Liste erzeugt. Ist die Datei vorhanden, wird der gesamte Inhalt mithilfe der Funktion `load()` aus dem Modul `pickle` in die Liste `hs_liste` gelesen.

Es folgt die Funktion zum Anzeigen des Highscores:

```
def hs_anzeigen():
    if not hs_liste:
        print("Keine Highscores vorhanden")
        return
    print(" P. Name           Zeit")
    for i in range(len(hs_liste)):
        print(f"{i+1:2d}. {hs_liste[ i][ 0]:10}"
              f" {hs_liste[ i][ 1]:5.2f} sec")
    if i >= 9:
        break
```

Listing 8.27 Datei »spiel_datei.py«, Teil 2 von 5

Ist die Highscore-Liste leer, wird eine entsprechende Meldung angezeigt. Ist die Liste nicht leer, werden nach einer Überschrift

maximal zehn Highscores mit den Informationen *Platzierung*, *Name*, *Zeit* formatiert ausgegeben.

Es folgt die Funktion zum Schreiben der Highscore-Liste in die Datei:

```
def hs_schreiben():
    d = open("highscore.bin", "wb")
    pickle.dump(hs_liste, d)
    d.close()
```

Listing 8.28 Datei »spiel_datei.py«, Teil 3 von 5

Die gesamte Liste wird mithilfe der Funktion `dump()` aus dem Modul `pickle` in die Datei geschrieben.

Es folgt die Funktion zum Spielen:

```
def spiel():
    name = input("Bitte geben Sie Ihren Namen ein (max. 10 Zeichen): ")
    name = name[0:10]
    richtig = 0
    startzeit = time.time()

    for aufgabe in range(5):
        a = random.randint(10, 30)
        b = random.randint(10, 30)
        c = a + b

        try:
            zahl = int(input(f"Aufgabe {aufgabe+1} von 5: {a} + {b} : "))
            if zahl == c:
                print("*** Richtig ***")
                richtig += 1
            else:
                raise
        except:
            print("* Falsch *")

    differenz = time.time() - startzeit
    print(f"Ergebnis: {richtig:d} von 5 in "
          f"{differenz:.2f} Sekunden", end = "")
    if richtig == 5:
        print(", Highscore")
    else:
        print(", leider kein Highscore")
    return

gefunden = False
for i in range(len(hs_liste)):
    if differenz < hs_liste[i][1]:
        hs_liste.insert(i, [name, differenz])
        gefunden = True
        break
if not gefunden:
```

```
    hs_liste.append([ name, differenz ] )
    hs_anzeigen()
```

Listing 8.29 Datei »spiel_datei.py«, Teil 4 von 5

Nach der Eingabe des Namens wird er zur einheitlichen Ausgabe auf maximal zehn Zeichen verkürzt. Es folgen einige Programmteile, die bereits aus früheren Versionen des Spiels bekannt sind: Stellen der fünf Aufgaben, Eingabe der Lösungen, Bewertung der Lösungen, Messung der Zeit.

Werden nicht alle fünf Aufgaben richtig gelöst, kann kein Eintrag in die Highscore-Liste erfolgen.

Die Highscore-Liste wird Element für Element durchsucht. Wird ein Element gefunden, in dem eine größere als die neue Ergebniszeit eingetragen ist, so wird die neue Ergebniszeit mithilfe der Methode `insert()` an dieser Stelle in die Highscore-Liste eingefügt.

Erfolgte kein Eintrag durch Einfügen, wird die neue Ergebniszeit mithilfe der Methode `append()` ans Ende der Liste angehängt. Auf diese Weise ist dafür gesorgt, dass die Highscore-Liste immer aufsteigend nach Ergebniszeit sortiert ist. Sie wird nach jeder Änderung angezeigt.

Es folgt das Hauptprogramm mit dem Menü:

```
# Programm
random.seed()
hs_leSEN()

while True:
    try:
        menu = int(input("Bitte eingeben "
                          "(0: Ende, 1: Highscores, 2: Spielen): "))
    except:
        print("Falsche Eingabe")
        continue

    if menu == 0:
        break
    elif menu == 1:
        hs_anzeigen()
    elif menu == 2:
        spiel()
    else:
        print("Falsche Eingabe")
```

```
hs_schreiben()
```

Listing 8.30 Datei »spiel_datei.py«, Teil 5 von 5

Nach der Initialisierung des Zufallsgenerators wird die Highscore-Datei eingelesen. Es beginnt die Endlosschleife mit dem Hauptmenü. Der Benutzer kann 0, 1 oder 2 eingeben. Alle anderen Eingaben sind falsch.

- Nach der Eingabe von 1 wird die Highscore-Liste angezeigt, anschließend erscheint wieder das Hauptmenü.
- Nach der Eingabe von 2 beginnt das Spiel, anschließend erscheint wieder das Hauptmenü.
- Nach der Eingabe von 0 wird die Endlosschleife verlassen. Die Highscore-Liste wird in die Datei geschrieben. Das Programm endet.

8.12 Spiel, objektorientierte Version mit Highscore-Datei

Es folgt eine weitere objektorientierte Version des Kopfrechenspiels. Diesmal werden die Daten des Spielers (Name und Zeit) in einer CSV-Datei gespeichert. Diese CSV-Datei können Sie sich mithilfe von MS Excel unter Windows oder mit LibreOffice Calc unter Ubuntu Linux und macOS ansehen.

Neben den beiden Klassen `Spiel` und `Aufgabe` benötigen Sie die Klasse `Highscore`. Die Klasse `Aufgabe` hat sich gegenüber der Version in der Datei `spiel_oop.py` nicht verändert.

Zunächst die Importanweisung und das Hauptprogramm:

```
import random, time, glob
...
# Programm
while True:
    try:
        menu = int(input("Bitte eingeben "
                          "(0: Ende, 1: Highscores, 2: Spielen): "))
    except:
        print("Falsche Eingabe")
        continue

    if menu == 0:
        break
    elif menu == 1:
        hs = Highscore()
        print(hs)
    elif menu == 2:
        s = Spiel()
        s.messen(True)
        s.spielen()
        s.messen(False)
        print(s)
    else:
        print("Falsche Eingabe")
```

Listing 8.31 Datei »spiel_datei_oop.py«, Hauptprogramm

Es werden die Module `random` (zur Erzeugung der Zufallszahlen), `time` (zur Zeitmessung) und `glob` (zur Prüfung der Datei) benötigt.

Möchte der Benutzer die Highscore-Liste sehen, wird ein Objekt der Klasse `Highscore` erzeugt und ausgegeben. Möchte er spielen, wird ein Objekt der Klasse `Spiel` erzeugt. Nach der Messung der Zeit und dem Spielvorgang werden die Ergebnisse ausgegeben.

Es folgt die Klasse `Spiel`:

```
class Spiel:
    def __init__(self):
        random.seed()
        self.richtig = 0
        self.anzahl = 5
        s = input("Bitte geben Sie Ihren "
                  "Namen ein (max. 10 Zeichen): ")
        self.name = s[0:10]

    def spielen(self):
        for i in range(1, self.anzahl+1):
            a = Aufgabe(i, self.anzahl)
            print(a)
            self.richtig += a.beantworten()

    def messen(self, start):
        if start:
            self.startzeit = time.time()
        else:
            self.zeit = round(time.time() - self.startzeit, 2)

    def __str__(self):
        ausgabe = f"Richtig: {self.richtig} von" \
                  f" {self.anzahl} in {self.zeit} Sekunden"
        if self.richtig == self.anzahl:
            ausgabe += ", Highscore"
            hs = Highscore()
            hs.speichern(self.name, self.zeit)
            print(hs)
        else:
            ausgabe += ", leider kein Highscore"
        return ausgabe
```

Listing 8.32 Datei »spiel_datei_oop.py«, Klasse »Spiel«

Im Konstruktor der Klasse wird der Zufallsgenerator initialisiert. Der Zähler für die richtig gelösten Aufgaben wird auf 0, die Anzahl der Aufgaben auf 5 gesetzt. Es wird der Name des Spielers ermittelt. Dabei werden drei Eigenschaften der Klasse `Spiel` gesetzt: `richtig`, `anzahl` und `name`.

In der Methode `spielen()` werden insgesamt fünf Aufgaben gestellt und beantwortet. Die Methode `messen()` dient zur

Zeitmessung. Es wird die Eigenschaft `zeit` der Klasse `Spiel` gesetzt.

Löst der Spieler alle Aufgaben richtig, wird in der Ausgabemethode ein Objekt der Klasse `Highscore` erzeugt und gespeichert. Anschließend wird die Liste ausgegeben.

Die neue Klasse `Highscore` sieht wie folgt aus:

```
class Highscore:
    def __init__(self):
        self.liste = []
        if not glob.glob("highscore.csv"):
            return
        d = open("highscore.csv")
        zeile = d.readline()
        while zeile:
            teil = zeile.split("; ")
            name = teil[0]
            zeit = teil[1]
            zeit = zeit.replace(", ", ".")
            self.liste.append([name, float(zeit)])
            zeile = d.readline()
        d.close()

    def aendern(self, name, zeit):
        gefunden = False
        for i in range(len(self.liste)):
            if zeit < self.liste[i][1]:
                self.liste.insert(i, [name, zeit])
                gefunden = True
                break
        if not gefunden:
            self.liste.append([name, zeit])

    def speichern(self, name, zeit):
        self.aendern(name, zeit)
        d = open("highscore.csv", "w")
        for element in self.liste:
            name = element[0]
            zeit = str(element[1]).replace(".", ",")
            d.write(f"{name}; {zeit}\n")
        d.close()

    def __str__(self):
        if not self.liste:
            return "Keine Highscores vorhanden"

        ausgabe = " P. Name          Zeit\n"
        for i in range(len(self.liste)):
            ausgabe += f"{i+1:2d}. {self.liste[i][0]:10} \\\n"
            f"{self.liste[i][1]:6.2f} sec\n"
            if i >= 9:
                break
        return ausgabe
```

Listing 8.33 Datei »spiel_datei_oop.py«, Klasse »Highscore«

Im Konstruktor der Klasse wird die wichtigste Eigenschaft gesetzt: die Highscore-Liste mit dem Namen `liste`. Sie ist zunächst leer. Gibt es keine CSV-Datei, bleibt sie leer. Gibt es dagegen eine CSV-Datei, werden alle Zeilen daraus gelesen. Anschließend werden die Zeilen zerlegt. Aus dem Dezimalkomma (für MS Excel oder LibreOffice Calc) wird ein Dezimalpunkt. Die beiden Bestandteile einer Zeile werden der Highscore-Liste jeweils als Unter-Liste angehängt.

Die Methode `aendern()` wird klassenintern von der Methode `speichern()` benötigt. Ein neu ermittelter Highscore wird entweder in die Liste eingefügt oder der Liste angehängt.

In der Methode `speichern()` wird zunächst die Highscore-Liste geändert. Anschließend wird aus dem Dezimalpunkt ein Dezimalkomma für MS Excel oder LibreOffice Calc. Die gesamte Liste wird in der CSV-Datei gespeichert. Zudem wird sie in der Ausgabemethode veröffentlicht, falls sie nicht leer ist.

9 Internet

Dieses Kapitel beschäftigt sich mit dem Empfangen und Senden von Internetdaten und mit der Erstellung von Programmen, die auf einem Webserver laufen. Sie können mithilfe von Python auf Daten im Internet zugreifen, Daten ins Internet senden und Daten anderen Benutzern im Internet zur Verfügung stellen.

Zum Testen der Programme dieses Kapitels wird ein lokaler Webserver benötigt. XAMPP ist ein vorkonfiguriertes und einfach zu installierendes Paket, das neben einem *Apache-Webserver* weitere Software umfasst, zum Beispiel die Webserver-Sprache *PHP* und das Datenbanksystem *MySQL* bzw. seine Abspaltung *MariaDB*. Bezuglich der Beispiele in diesem Buch stellt es keinen Unterschied dar, ob die jeweilige XAMPP-Version mit MySQL oder mit MariaDB arbeitet.

Sie erreichen die Website zum Herunterladen von XAMPP über die Adresse <https://www.apachefriends.org>, sowohl für Windows als auch für Ubuntu Linux und für macOS. Die Installation von XAMPP und das Starten der beiden Server werden in [Abschnitt A.3](#), »Installation von XAMPP«, beschrieben.

Am Ende dieses Kapitels wird das bereits bekannte Kopfrechenspiel in einer Version vorgestellt, die das Spielen im Internet ermöglicht. Die Aufgaben werden aus dem Internet abgerufen, die Benutzereingaben ins Internet gesendet und die Spielergebnisse mithilfe der Serialisierung dauerhaft auf einem Webserver im Internet gespeichert.

9.1 Laden und Senden von Internetdaten

Das Modul `urllib` mit den Untermodulen `urllib.request` und `urllib.parse` kann zum Laden von Daten aus dem Internet und zum Senden von Daten in das Internet verwendet werden.

Bei existierender Verbindung zu einem Webserver, ob lokal oder im Internet, haben Sie folgende Möglichkeiten:

- Sie können mithilfe der Funktion `urlopen()` eine Verbindung zu einer URL öffnen und anschließend die Inhalte in eine Variable eines Python-Programms einlesen.
- Sie können mithilfe der Funktion `urlretrieve()` Daten direkt von einer URL in eine Datei auf der Festplatte kopieren.

Mit der Funktion `urlopen()` können Sie auch Daten zur weiteren Verarbeitung auf dem Webserver an eine URL senden.

9.1.1 Daten lesen

Im nachfolgenden Python-Programm `url_leSEN.py` wird der Inhalt der Internetseite `http://localhost/Python310/url_leSEN.htm` mithilfe der Funktion `urlopen()` aus dem Modul `urllib.request` in eine Liste gelesen. Diese Liste wird anschließend ausgegeben.

Es folgt zunächst der Inhalt der einfachen HTML-Datei `url_leSEN.htm`:

```
<!DOCTYPE html><html>
<head>
    <meta charset="utf-8">
    <title>Titelzeile</title>
</head>
<body>
    <b>Hallo Python</b>
</body>
</html>
```

Listing 9.1 Datei »url_leSEN.htm«

Standardmäßig wird XAMPP im Verzeichnis `C:\xampp\htdocs` installiert. Die Datei `url_leSEN.htm` wird im Verzeichnis

C:\xampp\htdocs\Python310 gespeichert. Alle HTML-Dateien dieses Kapitels sollten die Kodierung UTF-8 besitzen. Nutzen Sie zu ihrer Erstellung den Editor Notepad++, können Sie die Datei bei Bedarf über den Menüpunkt KODIERUNG konvertieren.

Es handelt sich nur um einfache HTML-Codebeispiele, da eine weitere Einführung in HTML nicht Gegenstand dieses Buchs ist.

Geben Sie die Adresse *http://localhost/Python310/url_leSEN.htm* in einem Webbrower ein, erscheint die in Abbildung 9.1 gezeigte Ansicht.

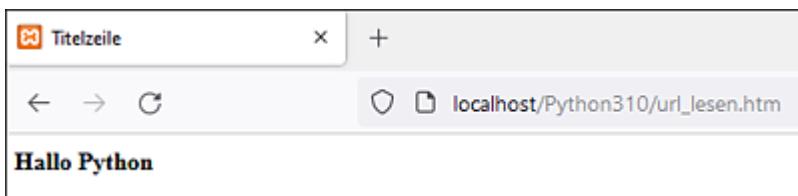


Abbildung 9.1 Erstes HTML-Dokument

Das Python-Programm zum Lesen des HTML-Codes dieser Internetseite sieht wie folgt aus:

```
import sys, urllib.request
try:
    u = urllib.request.urlopen \
        ("http://localhost/Python310/url_leSEN.htm")
except:
    print("Fehler")
    sys.exit(0)

li = u.readlines()
u.close()
for element in li:
    print(element)
```

Listing 9.2 Datei »url_leSEN.py«

Die Ausgabe lautet:

```
b'<!DOCTYPE html><html>\r\n'
b'<head>\r\n'
b'  <meta charset="utf-8">\r\n'
b'  <title>Titelzeile</title>\r\n'
b'</head>\r\n'
b'<body>\r\n'
b'  <b>Hallo Python</b>\r\n'
b'</body>\r\n'
b'</html>\r\n'
```

Erscheint eine Fehlermeldung, dann haben Sie eventuell vergessen, den lokalen Webserver zu starten (siehe [Abschnitt A.3](#), »Installation von XAMPP«).

Die Verbindung zur URL wird mithilfe der Funktion `urlopen()` geöffnet. Ihr Rückgabewert entspricht einem Dateiobjekt. Auf die geöffnete Datei können Sie, wie in [Abschnitt 8.3.2](#), »Lesen einer Textdatei«, beschreiben, zugreifen. Die Methode `readlines()` liefert eine Liste der Zeilen mit dem HTML-Code der Internetseite.

Jedes Element der Liste ist eine Zeichenkette, die als Byte-Literal ausgegeben wird, also als Zeichenketten des Datentyps `bytes` (siehe [Abschnitt 4.2.9](#), »Datentyp ›bytes‹«).

9.1.2 Daten kopieren

Das folgende Programm kopiert den HTML-Code der Internetseite `http://localhost/Python310/url_leSEN.htm` direkt in eine Datei auf der Festplatte.

```
import urllib.request
urllib.request.urlretrieve \
    ("http://localhost/Python310/url_leSEN.htm", "url_kopieren.htm")
```

Listing 9.3 Datei »url_kopieren.py«

Die Funktion `urlretrieve()` hat zwei Parameter: die URL und den Namen der Datei, in der der HTML-Code gespeichert wird (hier `url_kopieren.htm`). Diese Datei kann anschließend zum Beispiel offline mit einem Browser gelesen oder mit einem Editor bearbeitet werden.

9.1.3 Daten senden

Sie können dem Betrachter einer Website auch ermöglichen, spezifische Daten an den Webserver zu senden. Dies geschieht über Formulare, die der Betrachter ausfüllt und an den Webserver sendet.

Häufig werden diese Daten auf dem Webserver weiterverarbeitet und gespeichert, zum Beispiel in einer Datenbank. Außerdem kann eine individuelle Antwort an die Benutzer zurückgesendet werden.

HTML-PHP-Variante

Im folgenden Beispiel gibt der Betrachter seinen Vor- und Nachnamen ein und sendet diese Informationen an den Webserver. Er erhält daraufhin eine Bestätigung. Das wird, zunächst für die Ein- und Ausgabe in einem Browser, mithilfe einer HTML-Datei und einem zugehörigen PHP-Programm realisiert. [Abbildung 9.2](#) zeigt das Formular.

The screenshot shows a web browser window titled "Daten senden". The address bar indicates the URL is "localhost/Python310/senden_post.htm". The main content area contains the following text:
Bitte senden Sie Ihre Daten:
Nachname: Maier
Vorname: Werner
Daten absenden

Abbildung 9.2 Formular mit Beispieleingabe

Die Antwort des Webservers sehen Sie in [Abbildung 9.3](#).

The screenshot shows a web browser window titled "Daten empfangen". The address bar indicates the URL is "localhost/Python310/senden_post.php". The main content area contains the following text:
Ihre folgenden Daten wurden registriert:
Nachname: Maier
Vorname: Werner

Abbildung 9.3 Antwort des PHP-Programms

Das HTML-Formular gestaltet sich wie folgt:

```
<!DOCTYPE html><html>
<head>
    <meta charset="utf-8">
    <title>Daten senden</title>
</head>
<body>
    <b>Bitte senden Sie Ihre Daten:</b><p>
```

```

<form action="senden_post.php" method="post">
    <input name="nn"> Nachname<p>
    <input name="vn"> Vorname<p>
    <input type="submit">
</form>
</body>
</html>

```

Listing 9.4 Datei »senden_post.htm«

Das Formular wird mit `form` markiert, ein Eingabefeld mit `input`. Die im Formular verwendete POST-Methode ist wichtig für das Senden und Empfangen der Daten, sowohl für die HTML-PHP-Variante als auch für die später gezeigte Python-PHP-Variante.

Es folgt der PHP-Quellcode der zugehörigen Antwort:

```

<!DOCTYPE html><html>
<head>
    <meta charset="utf-8">
    <title>Daten empfangen</title>
</head>
<body>
    <b>Ihre folgenden Daten wurden registriert:</b><p>
    <?php
        $nn = htmlentities($_POST[ "nn" ]);
        $vn = htmlentities($_POST[ "vn" ]);
        echo "Nachname: $nn<br>Vorname: $vn";
    ?>
</body>
</html>

```

Listing 9.5 Datei »senden_post.php«

Es gibt in PHP einige vordefinierte Variablen, unter anderem das Feld `$_POST`. Werden die Daten mit der POST-Methode gesendet, werden aus den Namen der Eingabefelder automatisch Elemente des Felds `$_POST`. Die Elemente können angesprochen werden, indem Sie ihren Namen in Hochkommata und rechteckigen Klammern hinter dem Namen des Felds `$_POST` angeben. Die Eintragung im Texteingabefeld `vn` wird also zum Wert der Variablen `$_POST["vn"]` im Programm.

Beim Senden von Daten über das Internet ist die Absicherung gegen böswillige Benutzer ein wichtiges Thema. In einem Texteingabefeld könnte schädlicher Code eingegeben werden, der

nach der Übermittlung auf den Webserver zur Ausführung kommen könnte.

Daher wird die PHP-Funktion `htmlentities()` für den übermittelten Inhalt des Texteingabefelds aufgerufen. Sie wandelt alle HTML-spezifischen Zeichen in eine unschädliche Form um. Normaler Textinhalt wird von dieser Umwandlung nicht beeinflusst. Zeichen, die schädlichen Programmcode einleiten könnten, werden aber damit entschärft.

Wie bei HTML handelt es sich nur um einfache PHP-Codebeispiele, da auch eine weitere Einführung in PHP nicht Gegenstand dieses Buchs ist.

Python-PHP-Variante

Mithilfe des folgenden Python-Programms können diese Daten ebenfalls gesendet werden. Dabei wird auch das obige PHP-Programm *senden_post.php* auf dem Webserver angesprochen.

```
import urllib.request, urllib.parse

pnn = input("Bitte den Nachnamen eingeben: ")
pvn = input("Bitte den Vornamen eingeben: ")
dc = {b"nn":pnn, b"vn":pvn}
data = bytes(urllib.parse.urlencode(dc), "UTF-8")

u = urllib.request.urlopen \
    ("http://localhost/Python310/senden_post.php", data)
li = u.readlines()
u.close()
for element in li:
    print(element)
```

Listing 9.6 Datei »senden_post.py«

Zunächst werden die Eingabedaten vom Benutzer erfragt und in den beiden Variablen `pnn` und `pvn` gespeichert. Aus den Eingabedaten wird ein Dictionary erzeugt. Der Schlüssel ist der Name der Variablen, der Wert ist der Eingabewert des Benutzers.

Dieses Dictionary wird mithilfe der Funktionen `urlencode()` und `bytes()` in ein Format umgewandelt, das zur Codierung der

Sendedaten geeignet ist. Der zweite, optionale Parameter der Funktion `urlopen()` enthält diese Sendedaten. Anschließend wird der HTML-Code der Antwort des Webservers gelesen und ausgegeben.

Werden die gleichen Eingaben wie oben getätigt, lautet die Ausgabe:

```
Bitte den Nachnamen eingeben: Maier
Bitte den Vorname eingeben: Werner
b'<!DOCTYPE html><html>\r\n'
b'<head>\r\n'
b'  <meta charset="utf-8">\r\n'
b'  <title>Daten empfangen</title>\r\n'
b'</head>\r\n'
b'<body>\r\n'
b'  <b>Ihre folgenden Daten wurden registriert:</b><p>\r\n'
b'  Nachname: Maier<br>Vorname: Werner</body>\r\n'
b'</html>\r\n'
```

Wie man in der Ausgabe sieht, kommen die Daten auf dem Webserver an und werden vom PHP-Programm weiterverarbeitet. In diesem Fall hätte man zum Beispiel Nachname und Vorname in einer Datenbank auf dem Webserver speichern können.

9.2 Webserver-Programmierung

Das Modul `cgi` wird zur Webserver-Programmierung verwendet. Mit seiner Hilfe können Sie sogenannte CGI-Skripte erzeugen, die auf einem Webserver ausgeführt werden und Ergebnisse an die Betrachter senden.

Basis für die gesendeten, dynamisch erzeugten Ergebnisse sind unter anderem Benutzereingaben oder Inhalte von Dateien und Datenbanken, die auf dem Webserver liegen und mithilfe der CGI-Skripte gelesen und bearbeitet werden.

Wie im vorherigen Abschnitt wird ein lokaler Webserver für die Entwicklung und den Test der Programme benötigt.

Die CGI-Skripte werden im Browser über das Verzeichnis mit der Adresse `http://localhost/cgi-bin/Python310` abgerufen. Dieses Verzeichnis entspricht bei einer Standardinstallation von XAMPP unter Windows dem Festplattenverzeichnis `C:\xampp\cgi-bin\Python310`.

Es gibt auch Möglichkeiten, Python-Programme über bestimmte Module auszuführen, die bereits im Webserver eingebettet sind. Die Python-Programme, die diese Module nutzen, sind schneller als CGI-Skripte, allerdings gestaltet sich die Installation dieser Module auf dem lokalen Webserver sehr aufwendig. Im Rahmen dieses Einsteigerbuchs wird darauf nicht eingegangen.

Die CGI-Skripte in diesem Abschnitt werden unter Windows vorgeführt. Die Konfiguration der CGI-Schnittstelle ist unter Ubuntu Linux und unter macOS sehr aufwendig. Daher wird das Thema nur unter Windows behandelt.

9.2.1 Erstes Programm

Im Folgenden sehen Sie zunächst ein einfaches statisches CGI-Skript mit Python. Das Modul `cgi` wird hier noch nicht benötigt. In diesem Beispiel wird nur der Aufbau des Python-Programms und des HTML-Dokuments gezeigt:

```
#!C:\Python\python.exe
print("Content-type: text/html")
print()

print("<!DOCTYPE html><html>")
print("<head><meta charset=' utf-8' ><title>Hallo Python</title></head>")
print("<body>")
print("<h1>Hallo Python</h1>")
print("</body>")
print("</html>")
```

Listing 9.7 Datei »server_hallo.cgi«

In der ersten Zeile wird dem Webserver (unter Windows) die Information gegeben, dass die nachfolgenden Zeilen mithilfe von Python verarbeitet werden sollen. Der Webserver weiß damit, dass er keine Standarddatei vor sich hat, deren Daten einfach an den Browser des Benutzers gesendet werden. Stattdessen erstellt Python den HTML-Code.

Gleichzeitig wird der Ort mitgeteilt, an dem sich der Python-Interpreter auf dem Rechner befindet, auf dem auch der Webserver läuft. Dabei handelt es sich hier um die Datei `C:\Python\python.exe`. Die Zeile lautet dementsprechend:

```
#!C:\Python\python.exe
```

In der nächsten Zeile wird im sogenannten Header das Format des Dokumentinhalts (`Content-type: text/html`) festgelegt. Anschließend muss mithilfe von `print()` eine Leerzeile ausgegeben werden! Es folgt die Ausgabe der Zeilen des HTML-Quellcodes.

Die Ausgabe im Browser nach Eingabe der Adresse `http://localhost/cgi-bin/Python310/server_hallo.cgi` sehen Sie in [Abbildung 9.4](#).

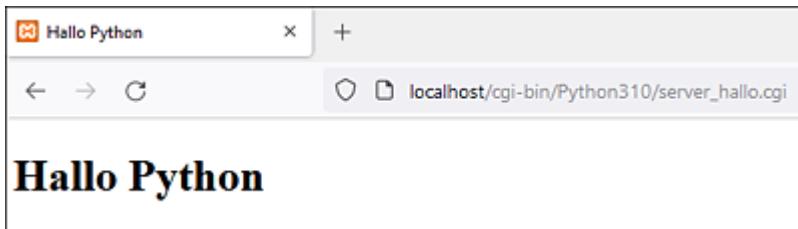


Abbildung 9.4 Erstes Python-HTML-Dokument

9.2.2 Beantworten einer Benutzereingabe

Im nächsten Beispiel soll der Betrachter seinen Vor- und Nachnamen eingeben und diese Informationen an den Webserver senden. Er erhält daraufhin eine Bestätigung. Abbildung 9.5 zeigt das Formular.

A screenshot of a web browser window titled "Daten senden". The address bar shows "localhost/Python310/server_antworten.htm". The page contains a form with the instruction "Bitte senden Sie Ihre Daten:". It has two input fields: one for "Nachname" containing "Maier" and one for "Vorname" containing "Werner". Below the fields is a button labeled "Daten absenden".

Abbildung 9.5 Formular mit Beispieleingabe

Die Antwort des Webservers sehen Sie in Abbildung 9.6.

A screenshot of a web browser window titled "Daten empfangen". The address bar shows "localhost/cgi-bin/Python310/server_antworten.cgi?nn=Maier&vn=Werner". The page displays the registered data: "Nachname: Maier" and "Vorname: Werner".

Abbildung 9.6 Antwort des Python-Programms

Unser bekanntes Beispiel realisieren wir nun mithilfe einer HTML-Datei und eines zugehörigen Python-Programms als CGI-Skript.

Zunächst der Code der HTML-Datei mit dem Formular:

```
<!DOCTYPE html><html>
<head>
    <meta charset="utf-8">
    <title>Daten senden</title>
```

```

</head>
<body>
    <b>Bitte senden Sie Ihre Daten:</b><p>
    <form action="/cgi-bin/Python310/server_antworten.cgi">
        <input name="nn"> Nachname<p>
        <input name="vn"> Vorname<p>
        <input type="submit">
    </form>
</body>
</html>

```

Listing 9.8 Datei »server_antworten.htm«

In der `form`-Markierung ist der Name der Datei angegeben, in der sich das Python-Programm befindet, das die Antwort zu diesem Formular sendet. Es handelt sich um die Datei *server_antworten.cgi* im Unterverzeichnis *Python310* des Standardverzeichnisses für CGI-Skripte.

Das zugehörige CGI-Skript sieht wie folgt aus:

```

#!/C:/Python/python.exe
import cgi, cgitb

cgitb.enable()
form = cgi.FieldStorage()
if "nn" in form:
    nn = form[ "nn" ].value
if "vn" in form:
    vn = form[ "vn" ].value

print("Content-type: text/html")
print()

print("<!DOCTYPE html><html>")
print("<head><meta charset=' utf-8'><title>Daten empfangen</title></head>")
print("<body>")

print("<p><b>Registrierte Daten:</b></p>")
print("<p>Nachname:", nn, "</p>")
print("<p>Vorname:", vn, "</p>")

print("</body>")
print("</html>")

```

Listing 9.9 Datei »server_antworten.cgi«

Es werden die Module `cgi` und `cgitb` eingebunden. Das Modul `cgi` wird für die Datenübermittlung benötigt. Das Modul `cgitb` können Sie während der Entwicklungszeit zur Fehlersuche nutzen.

Die Funktion `enable()` des Moduls `cgitb` sorgt dafür, dass Fehler während der Entwicklung des Python-Programms mit Kommentar im Browser ausgegeben werden. Ein Beispiel sehen Sie am Ende des Abschnitts. Nach Fertigstellung eines Programms sollten Sie die Zeile mit `cgitb.enable()` wieder auskommentieren.

Es wird ein Objekt (hier mit dem Namen `form`) der Klasse `FieldStorage` des Moduls `cgi` erzeugt. Dieses Objekt entspricht einem Dictionary und enthält alle ausgefüllten Elemente des Formulars mit ihren Werten.

Zunächst muss mithilfe des Operators `in` geprüft werden, ob der Benutzer einen Wert für ein bestimmtes Formularelement eingegeben hat oder nicht. Formularelemente ohne Wert werden nicht an das CGI-Programm übermittelt. Ihre Verwendung würde zu einem Laufzeitfehler führen. Anschließend kann der Wert des Formularelements zugewiesen werden.

Es folgt die Ausgabe des HTML-Codes, in dem die ermittelten Python-Variablen eingesetzt werden. Er wird eingeleitet durch die Header-Zeile, in der das Format des Dokumentinhalts festgelegt wird. Anschließend folgt wiederum eine Leerzeile.

9.2.3 Formularelemente mit mehreren Werten

Formularelemente können mehrere verschiedene Werte enthalten. Dies ist zum Beispiel bei einem Auswahlmenü des Typs `multiple` der Fall, in dem der Benutzer mehrere Werte markieren kann. Es können aber auch mehrere Formularelemente denselben Namen haben. Auch in diesem Fall wird zu einem Namen mehr als ein Wert übermittelt.

Wird nur ein Wert übermittelt, geschieht die Auswertung in der bereits bekannten Form. Werden mehrere Werte übermittelt, so stehen diese Werte in einer Liste zur Verfügung. Bei der

Auswertung muss daher festgestellt werden, ob kein Wert, genau ein Wert oder mehrere Werte vorhanden sind.

Das Formular in [Abbildung 9.7](#) enthält drei gleichnamige Textfelder.

The screenshot shows a web browser window titled "Daten senden". The address bar displays "localhost/Python310/server_werte.htm". The page content is a form with the following text:
Bitte senden Sie Ihre Daten:
Nachname 1: Maier
Nachname 2:
Nachname 3: Schmitz
A button labeled "Daten absenden" is at the bottom.

Abbildung 9.7 Formular mit Beispieleingabe

Der HTML-Code des Formulars:

```
<!DOCTYPE html><html>
<head>
    <meta charset="utf-8">
    <title>Daten senden</title>
</head>
<body>
    <b>Bitte senden Sie Ihre Daten:</b><p>
    <form action="/cgi-bin/Python310/server_werte.cgi">
        <input name="nn"> Nachname 1<p>
        <input name="nn"> Nachname 2<p>
        <input name="nn"> Nachname 3<p>
        <input type="submit">
    </form>
</body>
</html>
```

Listing 9.10 Datei »server_werte.htm«

Das Formular wird zur Auswertung an *server_werte.cgi* gesendet. Es enthält dreimal das Formularelement `nn`. Es können also maximal drei verschiedene Nachnamen eingegeben werden.

Die Antwort des Webservers zeigt [Abbildung 9.8](#).

The screenshot shows a web browser window titled "Daten empfangen". The address bar displays "localhost/cgi-bin/Python310/server_werte.cgi?nn=Maier&nn=&nn=Schmitz". The page content is a form with the following text:
Registrierte Daten:
Nachnamen:
Maier
Schmitz

Abbildung 9.8 Antwort des Python-Programms

Das zugehörige CGI-Skript sieht wie folgt aus:

```
#!/C:/Python/python.exe
import cgi, cgitb

cgitb.enable()
form = cgi.FieldStorage()

print("Content-type: text/html")
print()
print("<!DOCTYPE html><html>")
print("<head><meta charset=' utf-8'><title>Daten empfangen</title></head>")
print("<body>")

if "nn" in form:
    print("<p><b>Registrierte Daten:</b></p>")
    print("<p>Nachnamen:<br />")
    try:
        print(form[ "nn"] .value)
    except:
        for element in form[ "nn"]:
            print(element.value, "<br />")
    print("</p>")
else:
    print("<p><b>Keine Daten gesendet</b></p>")

print("</body>")
print("</html>")
```

Listing 9.11 Datei »server_werte.cgi«

Zunächst wird festgestellt, ob überhaupt ein Nachname eingegeben wurde. Ist das nicht der Fall, so wird ausgegeben:

Keine Daten gesendet.

Innerhalb eines `try-except`-Blocks wird versucht, einen einzelnen Nachnamen auszugeben. Werden mehrere Nachnamen übermittelt, schlägt das fehl. In diesem Fall werden alle Elemente der übermittelten Liste von Nachnamen ausgegeben.

9.2.4 Typen von Formularelementen

Im Folgenden sehen Sie die Auswertung verschiedener Typen von Formularelementen an einem größeren Beispiel – einer Onlinebestellung beim Python Pizza Service (siehe [Abbildung 9.9](#)).

Das Formular enthält die folgenden Typen von Formularelementen:

- zwei Radiobuttons für den Kundentyp
- ein Eingabefeld des Typs `password` für den Stammkunden-Code
- sechs einzeilige Eingabefelder für die Adresse
- ein einfaches Auswahlmenü für die Pizzasorte
- ein mehrfaches Auswahlmenü für die Zusätze zur Pizza
- ein Kontrollkästchen für den Express-Service
- ein mehrzeiliges Eingabefeld für Bemerkungen
- zwei Buttons zum Absenden und Zurücksetzen der Bestellung

The screenshot shows a web browser window titled "Daten senden" with the URL "localhost/Python310/server_pizza.htm". The page content is as follows:

Python Pizza Service

Stammkunde (5% Rabatt) Code
 Neukunde

Maier Nachname Werner Vorname
Pepperoniweg Straße 4 Hausnr.
63999 PLZ Teigdorf Ort

Sorte: Pizza Python (8.50 €)

Zusätze: Extra Pepperoni (1.00 €) Extra Oliven (1.20 €)

Express-Service (max. 30 Minuten, 1.50 € extra)

Bitte zweimal klingeln

Bemerkung:

Daten absenden Zurücksetzen

Abbildung 9.9 Python Pizza Service, Bestellung

Die Formularelemente, die eine einfache Auswahl verlangen, sind bereits mit einem Standardwert vorbesetzt. Darauf sollten Sie beim Formularentwurf unbedingt achten, damit die Benutzer nur sinnvolle Inhalte absenden können. Gleichzeitig muss im

auswertenden Programm nicht mehr geprüft werden, ob für diese Elemente ein Wert vorhanden ist. Das HTML-Formular sieht folgendermaßen aus:

```
<!DOCTYPE html><html>
<head>
    <meta charset="utf-8">
    <title>Daten senden</title>
</head>
<body>
    <p><b>Python Pizza Service</b></p>

    <form action="/cgi-bin/Python310/server_pizza.cgi">
        <p><input type="radio" name="ku" value="S"
            checked="checked">Stammkunde (5% Rabatt)
        <input type="password" name="kc" size="6"
            maxlength="6">Code<br />
        <input type="radio" name="ku" value="N"> Neukunde</p>

        <table>
            <tr>
                <td><input name="nn" size="12"> Nachname</td>
                <td><input name="vn" size="12"> Vorname</td>
            </tr>
            <tr>
                <td><input name="st" size="12"> Straße</td>
                <td><input name="hn" size="12"> Hausnr.</td>
            </tr>
            <tr>
                <td><input name="pz" size="12"> PLZ</td>
                <td><input name="or" size="12"> Ort</td>
            </tr>
        </table>

        <p>Sorte:</p>
        <select name="pt">
            <option value="Salami">
                Pizza Salami (6.00 &euro; )</option>
            <option value="Thunfisch">
                Pizza Thunfisch (6.50 &euro; )
            <option value="Quattro Stagioni">
                Pizza Quattro Stagioni (7.50 &euro; )</option>
            <option value="Python" selected="selected">
                Pizza Python (8.50 &euro; )</option>
        </select></p>

        <p>Zusätze:</p>
        <select name="zu" multiple size="2">
            <option value="Pepperoni">
                Extra Pepperoni (1.00 &euro; )</option>
            <option value="Oliven">
                Extra Oliven (1.20 &euro; )</option>
            <option value="Sardellen">
                Extra Sardellen (1.50 &euro; )</option>
        </select></p>
```

```

<p><input type="checkbox" name="ex">Express-Service  

    (max. 30 Minuten, 1.50 &euro; extra)</p>  

<p>Bemerkung: <textarea name="bm" rows="3"  

    cols="25"></textarea></p>  

<p><input type="submit"> <input type="reset"></p>  

</form>  

</body>  

</html>

```

Listing 9.12 Datei »server_pizza.htm«

Eingabefelder des Typs `radio` (Radiobuttons), die denselben Namen haben, bieten eine einfache Auswahl zwischen mehreren Alternativen, hier zwischen **STAMMKUNDE** und **NEUKUNDE**. Die Vorauswahl wird durch `checked` gekennzeichnet.

Eingaben in Eingabefeldern des Typs `password` sind nicht lesbar.

Einfache Menüs mit `select` dienen zur Auswahl einer Option, hier einer Pizzasorte. Eine Vorauswahl wird mit `selected` gekennzeichnet.

Mehratische Menüs mit `select` und `multiple` dienen zur Auswahl einer oder mehrerer Optionen, hier der Zusätze zu den Pizzen. Eine mehratische Auswahl erfolgt mithilfe der Tasten **[Space]** und **[Strg]**.

Mithilfe eines Kontrollkästchens (englisch: *checkbox*) übermitteln Sie eine zusätzliche Information, hier den Wunsch nach Express-Service.

Ein mehrzeiliges Eingabefeld (`textarea`) bietet die Möglichkeit, längere Texte einzutragen und zu übermitteln. Die Größe der Textarea wird mithilfe der Anzahl der Zeilen (`rows`) und Spalten (`cols`) festgelegt.

Die Antwort des Webservers auf die obige Bestellung (siehe [Abbildung 9.10](#)) erfolgt mit einem Python-CGI-Skript.

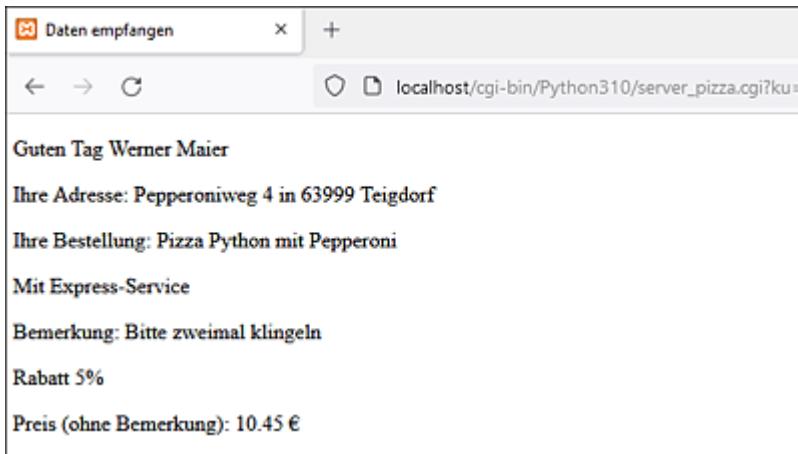


Abbildung 9.10 Python Pizza Service, Bestätigung der Bestellung

Das zugehörige CGI-Skript sieht wie folgt aus:

```
#!C:\Python\python.exe
import cgi, cgitb

def chk(element):
    global form
    if element in form:
        erg = form[element].value
    else:
        erg = ""
    return erg

cgitb.enable()
form = cgi.FieldStorage()

print("Content-type: text/html")
print()
print("<!DOCTYPE html><html>")
print("<head><meta charset=' utf-8'><title>Daten empfangen</title></head>")
print("<body>")

print("<p>Guten Tag", chk("vn"), chk("nn"), "</p>")
print("<p>Ihre Adresse:", chk("st"), chk("hn"),
     "in", chk("pz"), chk("or"), "</p>")
print("<p>Ihre Bestellung: Pizza", chk("pt"))

preisliste = {"Salami":6, "Thunfisch":6.5,
              "Quattro Stagioni":7.5, "Python":8.5}
preis = preisliste[ form["pt"].value]
zusatzliste = {"Pepperoni":1, "Oliven":1.2,
               "Sardellen":1.5}

if "zu" in form:
    try:
        print("mit", form["zu"].value)
        preis += zusatzliste[ form["zu"].value]
    except:
        for element in form["zu"]:
            print(", mit", element.value)
            preis += zusatzliste[ element.value]
```

```

print("</p>")

if "ex" in form:
    print("<p>Mit Express-Service</p>")
    preis += 1.5
if "bm" in form:
    print("<p>Bemerkung:</p>", form[ "bm" ].value, "</p>")
if "kc" in form:
    if form[ "ku" ].value == "S" \
        and form[ "kc" ].value == "Bingo":
        preis = preis * 0.95
    print("<p>Rabatt 5%</p>")

print(f"Preis (ohne Bemerkung): {preis:.2f} &euro; ")
print("</body>")
print("</html>")

```

Listing 9.13 Datei »server_pizza.cgi«

Zunächst wird die Prüffunktion `chk()` für diejenigen Formularelemente definiert, die maximal einen Wert übermitteln können. Sie liefert den Wert des geprüften Elements oder eine leere Zeichenkette zurück.

Nach der Überschrift folgt die Anrede. Zur Angabe des Vornamens und des Nachnamens wird die Prüffunktion genutzt. Es folgt die Bestätigung der Adresse, ebenfalls mit Nutzung der Prüffunktion.

Die Pizzasorte wird angegeben, und der zugehörige Preis wird über ein Dictionary berechnet. Die Existenz des Elements muss nicht überprüft werden, da bereits eine Vorauswahl festgelegt ist.

Die optionalen Zusätze für die Pizza werden in der Ausgabe und bei der Berechnung des Preises (über ein Dictionary) hinzugefügt. Es muss geprüft werden, ob der Benutzer Zusätze gewählt hat.

Es folgen Prüfung, Ausgabe und Preisberechnung für den Express-Service und die Ausgabe der Bemerkung.

Nach Berechnung des Stammkunden-Rabatts wird der Preis ausgegeben. Dies geschieht allerdings nur, sofern der Code »Bingo« im Passwortfeld eingegeben wird.

9.3 Browser aufrufen

Das Modul `webbrowser` enthält Funktionen, um Internetseiten über einen Browser abzurufen. Es genügt, die Funktion `open()` aufzurufen.

Als Parameter geben Sie die URL der gewünschten Internetseite an:

```
import webbrowser
webbrowser.open("https://www.rheinwerk-verlag.de")
print("Ende")
```

Listing 9.14 Datei »browser.py«

Der Standard-Webbrowser des Systems wird als unabhängiges Programm aufgerufen. Danach läuft das Python-Programm weiter.

9.4 Spiel, Version für das Internet

In dieser Version des Kopfrechenspiels kann die Spielerin zunächst in einem HTML-Formular auf einer Internetseite ihren Namen eingeben. Anschließend werden ihr, wiederum in einem HTML-Formular, fünf Additionsaufgaben mit Zahlen aus dem Bereich von 10 bis 30 gestellt.

Die Spielerin gibt ihre Lösungen in die fünf zugehörigen Eingabefelder ein. Pro Aufgabe hat sie nur einen Versuch. Am Ende wird die Gesamtzeit ermittelt, die sie für ihre Lösungsversuche benötigt hat.

Löst die Spielerin alle fünf Aufgaben richtig, werden ihr Name und die benötigte Zeit in eine Highscore-Liste aufgenommen, die mithilfe der Serialisierung dauerhaft auf dem Webserver abgespeichert wird.

9.4.1 Eingabebeispiel

Es folgt ein typischer Durchlauf durch das Programm.

Abbildung 9.11 zeigt zunächst das Formular für die Eingabe des Namens.

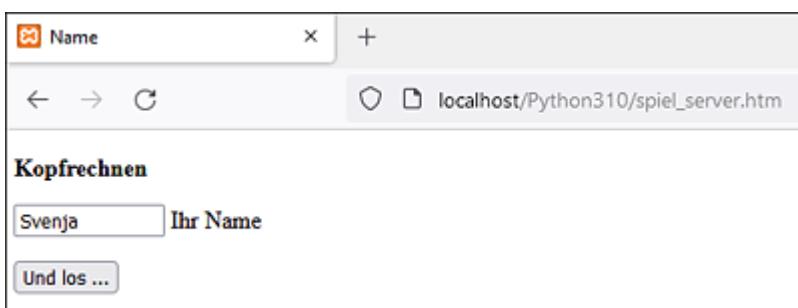


Abbildung 9.11 Beginn des Spiels

Es folgen die fünf Aufgaben, siehe Abbildung 9.12.

Aufgaben

Kopfrechnen

Hello Svenja, Ihre Aufgaben

1. $13 + 16 =$
2. $14 + 14 =$
3. $12 + 11 =$
4. $26 + 11 =$
5. $15 + 18 =$

Fertig

Abbildung 9.12 Fünf Aufgaben mit Lösungsversuchen

Die Bewertung der Ergebnisse und die Highscore-Liste sehen Sie in [Abbildung 9.13](#).

Auswertung

Kopfrechnen

Hello Svenja, Ihr Ergebnis

5 von 5 in 11.36 Sekunden, Highscore

Platz	Name	Zeit
1	Svenja	10.44 sec
2	Lars	11.26 sec
3	Svenja	11.36 sec

[Zum Start](#)

Abbildung 9.13 Bewertung, Highscore

Die Spielerin gibt ihren Namen ein und betätigt den Button **UND LOS** Sie erhält fünf Aufgaben, gibt ihre Lösungen ein und betätigt den Button **FERTIG**. Ihre Eingaben werden ausgewertet. Es erscheint die Highscore-Liste, gegebenenfalls mit einem neuen Eintrag für die aktuelle Spielerin.

Das Programm hat einen kleinen Schönheitsfehler: Es misst nicht die Zeit, die die Spielerin zum Lösen der Aufgaben tatsächlich benötigt, sondern die Zeit, die zwischen dem Erscheinen der fünf Aufgaben und dem Erscheinen der Auswertungsseite verstreicht. Somit fließt neben der Schnelligkeit der Spielerin auch die Qualität der Internetverbindung in das Ergebnis ein. Eine aufwendigere Lösung würde dieses Problem umgehen.

9.4.2 Aufbau des Programms

Das Programm besteht aus drei Dateien:

Erstes Formular

Bei der ersten Datei handelt es sich um eine reine HTML-Datei (http://localhost/Python310/spiel_server.htm). Darin steht das Formular mit dem Feld zur Eingabe des Namens.

Zweites Formular

Bei der zweiten Datei handelt es sich um ein Python-CGI-Skript (http://localhost/cgi-bin/Python310/spiel_server_a.cgi). Darin wird der Eingabewert aus dem ersten Formular entgegengenommen, und das zweite Formular wird dynamisch aufgebaut. Das zweite Formular enthält fünf Eingabefelder für die Lösungsversuche der fünf Aufgaben.

Außerdem enthält das zweite Formular insgesamt sieben versteckte Felder. In diesen Feldern werden die fünf richtigen Lösungen der Aufgaben, der Name der Spielerin aus dem ersten Formular und die ermittelte Startzeit unsichtbar an die dritte Datei übermittelt.

Auswertung

Die dritte Datei ist wiederum ein Python-CGI-Skript (http://localhost/cgi-bin/Python310/spiel_server_b.cgi). In diesem Skript werden insgesamt zwölf Werte aus dem zweiten Formular entgegengenommen. Dies sind:

- der Spielername
- die ermittelte Startzeit
- die fünf richtigen Lösungen

- die fünf Eingaben der Benutzerin

Aus der Startzeit und der festgestellten Endzeit wird die Ergebniszeit ermittelt, die die Benutzerin benötigt hat. Die fünf richtigen Lösungen und die fünf Eingaben der Benutzerin werden miteinander verglichen.

Sind alle Lösungen richtig, werden Name und Zeit in die Highscore-Liste eingetragen, die anschließend angezeigt wird. Außerdem wird ein Hyperlink eingeblendet, mit dem die Benutzerin wieder zum Start des Programms gelangt.

9.4.3 Code des Programms

Es folgt der Code des ersten Formulars:

```
<!DOCTYPE html><html>
<head>
    <meta charset="utf-8">
    <title>Name</title>
</head>
<body>
    <p><b>Kopfrechnen</b></p>
    <form action="/cgi-bin/Python310/spiel_server_a.cgi">
        <p><input name="name" size="12"> Ihr Name</p>
        <p><input type="submit" value="Und los ..."></p>
    </form>
</body>
</html>
```

Listing 9.15 Datei »spiel_server.htm«

Das erste Formular enthält das Element `name` und den Button `UND LOS ...` zum Absenden des Formulars. Seine Betätigung ruft das zweite Formular auf. Dessen Code sieht wie folgt aus:

```
#!C:\Python\python.exe
import cgi, cgitb, sys, random, time
cgitb.enable()
random.seed()
form = cgi.FieldStorage()

print("Content-type: text/html")
print()

print("<!DOCTYPE html><html>")
print("<head><meta charset=' utf-8' ><title>Aufgaben</title></head>")
print("<body>")
```

```

print("<p><b>Kopfrechnen</b></p>")

if not "name" in form:
    print("<p>Kein Name, kein Spiel</p>")
    print("<a href=' http://localhost/Python310/spiel_server.htm' >Zurück</a>")
    print("</body>")
    print("</html>")
    sys.exit(0)

print("<form action='spiel_server_b.cgi' >")
print(f"<p>Hallo <b>{form['name'].value}</b>, Ihre Aufgaben</p>")
print("<input name='name' type='hidden' value=' " + form['name'].value + "' >")

startzeit = time.time()
print(f"<input name='startzeit' type='hidden' value=' {startzeit}' >")

print("<table border='0' >")
for aufgabe in range(5):
    a = random.randint(10, 30)
    b = random.randint(10, 30)
    c = a + b

    print("<tr>")
    print(f"<td> {aufgabe+1}. </td>")
    print(f"<td> {a} </td>")
    print(" <td> + </td>")
    print(f"<td> {b} </td>")
    print(" <td> = </td>")
    print(f"<td><input name='ein{aufgabe}' size='12' ></td>")
    print("</tr>")

    print(f"<input name='erg{aufgabe}' type='hidden' value=' {c}' >")

print("</table>")
print("<p><input type='submit' value='Fertig' ></p>")
print("</form>")
print("</body>")
print("</html>")

```

Listing 9.16 Datei »spiel_server_a.cgi«

Die Eingaben aus dem ersten Formular werden in die Python-Variable `form` übernommen.

Sollte kein Spielername eingegeben worden sein, wird nur noch ein Hyperlink angeboten, der zurück zum ersten Formular führt. Ohne Spielername geht es nicht weiter. Daher enden das HTML-Dokument und das Python-CGI-Skript.

Der übermittelte Spielername wird angezeigt, und die Startzeit wird gespeichert. Die beiden Werte werden für die spätere Auswertung in versteckten Formularelementen eingetragen.

Es wird eine HTML-Tabelle aufgebaut. Darin werden in einer `for`-Schleife die fünf Aufgaben und die fünf Eingabefelder für die Lösungsversuche eingetragen. Die Namen der fünf Eingabefelder werden dynamisch zusammengesetzt: `ein0` bis `ein4`.

Parallel dazu werden die fünf richtigen Lösungen für die spätere Auswertung in weiteren versteckten Formularelementen eingetragen, die die Namen `erg0` bis `erg4` haben. Es folgen das Ende der Tabelle, der Button zum Absenden und das Ende des HTML-Dokuments.

Kommen wir zur letzten Seite, der Auswertungsseite. Sie enthält ein Hauptprogramm und vier Funktionen:

- `hs_leSEN()`: Lesen der Highscores aus der Datei in eine Liste
- `hs_schreiben()`: Schreiben der Highscore-Liste in die Datei
- `hs_anzeigen()`: Anzeigen der Highscore-Liste im Browser
- `hs_hinzu()`: Einfügen der ermittelten Zeit in die Highscore-Liste

Die beiden erstgenannten Funktionen kennen Sie bereits aus der letzten Version des Spiels. Das Anzeigen der Highscore-Liste wird an die Ausgabe im Browser angepasst. Statt einer formatierten Ausgabe auf dem Bildschirm wird eine HTML-Tabelle erzeugt. Die Funktion `hs_hinzu()` ist ein Teil der Funktion `spiel()` aus einer anderen Spielversion. Sie erkennen daran die leichte Wiederverwendbarkeit von modular aufgebaute Programmen.

Es folgt der Code der Auswertungsseite:

```
#!/C:/Python/python.exe
import cgi, cgitb, time, glob, pickle
cgitb.enable()

def hs_leSEN():
    global hs_liste
    if not glob.glob("highscore.bin"):
        hs_liste = []
        return
    d = open("highscore.bin", "rb")
    hs_liste = pickle.load(d)
```

```

d.close()

def hs_anzeigen():
    print("<table>")
    print("<tr><td>Platz</td><td>&nbsp; Name&nbsp;</td>""
          "<td align=' right' >Zeit</td></tr>")
    for i in range(len(hs_liste)):
        print("<tr>")
        print(f"<td align=' right' >{i+1}</td>")
        print(f"<td> {hs_liste[ i][ 0]} </td>")
        print(f"<td align=' right' >{hs_liste[ i][ 1] :.2f} sec</td>")
        print("</tr>")
        if i >= 9:
            break
    print("</table>")

def hs_schreiben():
    d = open("highscore.bin", "wb")
    pickle.dump(hs_liste,d)
    d.close()

def hs_hinzu(name, zeit):
    gefunden = False
    for i in range(len(hs_liste)):
        if zeit < hs_liste[ i][ 1]:
            hs_liste.insert(i, [ name, zeit])
            gefunden = True
            break
    if not gefunden:
        hs_liste.append([ name, zeit])

# Programm
form = cgi.FieldStorage()
differenz = round(time.time() - float(form[ "startzeit"].value), 2)

print("Content-type: text/html")
print()

print("<!DOCTYPE html><html>")
print("<head><meta charset=' utf-8'><title>Auswertung</title></head>")
print("<body>")
print("<p><b>Kopfrechnen</b></p>")

print(f"<p>Hallo <b>{form[ 'name'].value}</b>, Ihr Ergebnis</p>")

richtig = 0
for aufgabe in range(5):
    if f"ein{aufgabe}" in form:
        if form[ f"ein{aufgabe}"].value == form[ f"erg{aufgabe}"].value:
            richtig += 1

print(f"<p>{richtig} von 5 in {differenz:.2f} Sekunden", end = "")

if richtig == 5:
    print(", Highscore</p>")
    hs_leSEN()
    hs_hinzu(form[ "name"].value, differenz)
    hs_schreiben()

```

```

    hs_anzeigen()
else:
    print(", leider kein Highscore</p>")

print("<p><a href='http://localhost/Python310/spiel_server.htm' >Zum Start</a>
</p>")

print("</body>")
print("</html>")

```

Listing 9.17 Datei »spiel_server_b.cgi«

Die Funktion `hs_anzeigen()` enthält eine HTML-Tabelle mit der Überschrift und den einzelnen Zeilen für die Highscore-Daten. Die Zahlen werden formatiert ausgegeben.

An die Funktion `hs_hinzu()` werden Name und ermittelte Zeit der Spielerin übergeben. Diese beiden Werte werden an der passenden Stelle in die Highscore-Liste eingefügt oder hinten angehängt.

Im Hauptprogramm wird die Endzeit ermittelt. Aus der übermittelten Startzeit und der Endzeit wird die Ergebniszeit errechnet.

Die fünf Lösungen werden mit den fünf Eingaben verglichen, und die Anzahl der gelösten Aufgaben wird berechnet. Werden alle Aufgaben richtig gelöst, passiert Folgendes:

- Die Highscores aus der Datei werden eingelesen.
- Der neue Highscore wird mit Namen und Ergebniszeit hinzugefügt.
- Die Highscore-Liste wird wieder in die Datei geschrieben.
- Die Highscores werden angezeigt.

Schließlich wird der Hyperlink zum Start des Programms angegeben.

10 Datenbanken

Eine Datenbank dient zur Speicherung größerer Datenmengen, zur Bearbeitung ausgewählter Daten und zur übersichtlichen Darstellung von Daten. Mithilfe eines Datenbanksystems und der Sprache SQL (*Structured Query Language*) haben Sie Zugriff auf Datenbanken. In diesem Buch arbeiten wir mit zwei verschiedenen SQL-basierten Datenbanksystemen:

- SQLite: Dieses System basiert auf Textdateien und ist für kleinere Datenmengen geeignet.
- MySQL: Dabei handelt es sich um ein komplexes Datenbanksystem, das auch für große Datenmengen geeignet ist.

10.1 Aufbau von Datenbanken

Innerhalb einer Datenbank befinden sich verschiedene Tabellen. In Tabelle 10.1 sehen Sie eine einfache Tabelle mit den Daten von Personen.

Name	Vorname	Personalnummer	Gehalt	Geburtstag
Maier	Hans	6714	3.500,00	15.03.62
Schmitz	Peter	81343	3.750,00	12.04.58
Mertens	Julia	2297	3.621,50	30.12.59

Tabelle 10.1 Beispiel mit Daten von Personen

Die Begriffe in der ersten Zeile bezeichnet man als die *Felder* der Tabelle. Es folgen die einzelnen *Datensätze* der Tabelle, hier sind es drei.

Niemand legt für drei Datensätze eine Datenbank mit einer Tabelle an. Die vorliegende Struktur könnte aber auch für mehrere Tausend Datensätze verwendet werden. Die Felder haben jeweils einen bestimmten Datentyp. In diesem Fall gibt es die Datentypen *Text*, *Ganze Zahl*, *Zahl mit Nachkommastellen* und *Datumsangabe*.

Eine Datenbank wird in den folgenden Schritten erzeugt:

1. Anlegen der Datenbank
2. Anlegen von Tabellen durch Angabe der Struktur
3. Eingeben der Datensätze in die Tabellen

Sie können die Struktur einer Datenbank auch noch verändern, wenn sie bereits Daten enthält. Allerdings kann es dabei leicht zu Datenverlusten kommen. Daher sollten Sie die Struktur besser vorher gründlich planen.

Sie werden mit SQL-Anweisungen arbeiten. Diese dienen zur Erzeugung der Struktur von Datenbanken und Tabellen und zum Bearbeiten der Datensätze (Erzeugen, Anzeigen, Ändern, Löschen).

10.2 SQLite

Bei SQLite handelt es sich um ein kleines und einfach zu nutzendes Datenbanksystem. Es wird über das Modul `sqlite3` direkt in Python eingebunden, arbeitet auf der Basis von Textdateien und kann für kleinere Datenmengen genutzt werden. Es erfordert keine zusätzlichen Installationen.

SQLite bietet standardmäßig die folgenden Datentypen:

- `TEXT` – für Zeichenketten
- `INTEGER` – für ganze Zahlen
- `REAL` – für Zahlen mit Nachkommastellen
- `BLOB` – für *binary large objects*, also große binäre Datenmengen
- `NULL` – entspricht `None` in Python

Für die Kontrolle und die richtige Konvertierung anderer Datentypen von SQLite nach Python, zum Beispiel für die Umwandlung einer SQLite-Zeichenkette in eine Python-Variable für eine Datumsangabe, muss die Entwicklerin selbst sorgen.

Hinweis

Die oben genannten Datentypen und andere SQL-spezifische Angaben werden in Großbuchstaben notiert. Das ist für SQL nicht notwendig, dient aber zur deutlicheren Darstellung.

10.2.1 Datenbank, Tabelle und Datensätze

Im folgenden Programm wird zunächst eine SQLite-Datenbank erzeugt und eine Tabelle mit einem eindeutigen Index angelegt. Anschließend werden drei Datensätze in der Tabelle angelegt. Dazu werden die Struktur und die Daten aus [Tabelle 10.1](#) am Anfang dieses Kapitels genutzt.

Der eindeutige Index dient zur eindeutigen Identifizierung von Datensätzen. In der Tabelle mit den Daten von Personen ist dazu das Feld `personalnummer` geeignet. Das Programm lautet:

```
import os, sys, sqlite3
if os.path.exists("firma.db"):
    print("Datenbank bereits vorhanden")
    sys.exit(0)

connection = sqlite3.connect("firma.db")
cursor = connection.cursor()
sql = "CREATE TABLE personen(\" \
    \"name TEXT, \" \
    \"vorname TEXT, \" \
    \"personalnummer INTEGER PRIMARY KEY, \" \
    \"gehalt REAL, \" \
    \"geburtstag TEXT\")"
cursor.execute(sql)

sql = "INSERT INTO personen VALUES(' Maier' , \" \
    '\" Hans', 6714, 3500, '15.03.1962' )"
cursor.execute(sql)
connection.commit()
sql = "INSERT INTO personen VALUES(' Schmitz' , \" \
    '\" Peter', 81343, 3750, '12.04.1958' )"
cursor.execute(sql)
connection.commit()
sql = "INSERT INTO personen VALUES(' Mertens' , \" \
    '\" Julia', 2297, 3621.5, '30.12.1959' )"
cursor.execute(sql)
connection.commit()

connection.close()
```

Listing 10.1 Datei »sqlite_erzeugen.py«

Zunächst wird geprüft, ob die Datenbankdatei `firma.db` bereits existiert. Ist das der Fall, wurde das Programm bereits einmal aufgerufen, die Tabelle `personen` existiert und ist mit Daten gefüllt. Ein zweiter Aufruf würde zu einem Fehler führen. Möchten Sie die Datenbank neu anlegen, müssen Sie die Datenbankdatei `firma.db` zunächst löschen.

Anschließend wird mithilfe der Funktion `connect()` aus dem Modul `sqlite3` eine Verbindung zur Datenbank hergestellt. Da die Datenbankdatei noch nicht existiert, wird sie nun erzeugt. Der Rückgabewert der Funktion `connect()` ist ein Objekt der Klasse `sqlite3.Connection`. Über diese Verbindung kann jetzt auf die Datenbank zugegriffen werden.

Für das Verbindungsobjekt wird die Methode `cursor()` aufgerufen. Sie liefert als Rückgabewert ein Objekt der Klasse `sqlite3.cursor`. Über dieses `cursor`-Objekt können SQL-Befehle an die Datenbank gesendet und die Ergebnisse empfangen werden.

Es wird eine Zeichenkette zusammengesetzt, die einen SQL-Befehl enthält. SQL-Befehle werden mithilfe der Methode `execute()` des `cursor`-Objekts an die Datenbank gesendet.

Die SQL-Anweisung `CREATE TABLE` erzeugt eine Tabelle in einer Datenbank. Hinter dem Namen der Tabelle (hier: `personen`) werden in Klammern die Namen der einzelnen Felder der Tabelle mit ihrem jeweiligen Datentyp angegeben. Die Felder `name`, `vorname` und `geburtstag` sind vom Typ `TEXT`. Das Feld `gehalt` ist vom Typ `REAL`, das Feld `personalnummer` vom Typ `INTEGER`.

Auf dem Feld `personalnummer` wird mithilfe des Zusatzes `PRIMARY KEY` ein Primärschlüssel definiert. Dabei handelt es sich um einen eindeutigen Index, der dafür sorgt, dass es keine zwei Datensätze geben kann, die dieselbe Personalnummer besitzen.

Mithilfe der SQL-Anweisung `INSERT INTO` werden Datensätze eingefügt. Es handelt sich um eine Aktionsabfrage, die zu einer Änderung in der Tabelle führt. Nach dem Namen der Tabelle und dem Schlüsselwort `VALUES` werden die einzelnen Werte des Datensatzes für die Felder der Tabelle in Klammern angegeben. Dabei ist es wichtig, Zeichenketten in einfache Anführungsstriche zu setzen. Außerdem muss die Reihenfolge der Felder derjenigen bei der Erzeugung der Tabelle entsprechen.

Auf eine Aktionsabfrage sollte immer ein Aufruf der Methode `commit()` des Verbindungsobjekts folgen, damit die Änderungen unmittelbar ausgeführt werden.

Zuletzt wird die Verbindung zur Datenbank mithilfe der Methode `close()` wieder geschlossen.

10.2.2 Daten anzeigen

Alle Datensätze der Tabelle werden mit allen Inhalten angezeigt:

```
import sqlite3
connection = sqlite3.connect("firma.db")
cursor = connection.cursor()
sql = "SELECT * FROM personen"
cursor.execute(sql)

for dsatz in cursor:
    print(dsatz[ 0 ], dsatz[ 1 ], dsatz[ 2 ],
          dsatz[ 3 ], dsatz[ 4 ])
connection.close()
```

Listing 10.2 Datei »sqlite_anzeigen.py«

Das Programm erzeugt diese Ausgabe:

```
Mertens Julia 2297 3621.5 30.12.1959
Maier Hans 6714 3500.0 15.03.1962
Schmitz Peter 81343 3750.0 12.04.1958
```

Nach Erzeugung der Datenbankverbindung und des Datensatz-Cursors wird eine SQL-Abfrage mithilfe der Methode `execute()` gesendet.

Mit der SQL-Anweisung `SELECT` werden Auswahlabfragen zur Auswahl von Datenbankinhalten formuliert. Die Anweisung `SELECT * FROM personen` liefert alle Datensätze mit allen Feldern der Tabelle. Der Stern (*) steht für »alle Felder«.

Nach Ausführung der Methode `execute()` steht das Ergebnis der Abfrage im `cursor`-Objekt zur Verfügung. Dabei handelt es sich um die Datensätze, die zur Abfrage passen. Die Abfolge dieser Datensätze kann mithilfe einer Schleife durchlaufen werden. Jeder Datensatz entspricht einem Tupel, das aus den Werten für die einzelnen Felder in der gleichen Reihenfolge wie bei der Erzeugung der Tabelle besteht.

Da es sich bei `SELECT` nicht um eine Aktionsabfrage handelt (die eine Änderung in der Datenbank vornimmt), sondern nur um eine Auswahlabfrage (die Informationen ermittelt), muss hier die Methode `commit()` nicht aufgerufen werden.

Zuletzt wird die Verbindung zur Datenbank wieder beendet.

10.2.3 Daten auswählen, Operatoren

Mithilfe von `SELECT` und `WHERE` können Sie einen oder mehrere Datensätze herausfiltern. Es folgen einige Beispiele:

```
import sqlite3
connection = sqlite3.connect("firma.db")
cursor = connection.cursor()

cursor.execute("SELECT name, vorname FROM personen")
for dsatz in cursor:
    print(dsatz[ 0 ], dsatz[ 1 ])
print()

cursor.execute("SELECT * FROM personen WHERE gehalt > 3600")
for dsatz in cursor:
    print(dsatz[ 0 ], dsatz[ 3 ])
print()

cursor.execute("SELECT * FROM personen WHERE name = 'Schmitz' ")
for dsatz in cursor:
    print(dsatz[ 0 ], dsatz[ 1 ])
print()

cursor.execute("SELECT * FROM personen WHERE gehalt >= 3600 AND gehalt <= 3650")
for dsatz in cursor:
    print(dsatz[ 0 ], dsatz[ 3 ])

connection.close()
```

Listing 10.3 Datei »sqlite_auswaehlen.py«

Erzeugt wird die Ausgabe:

```
Mertens Julia
Maier Hans
Schmitz Peter
```

```
Mertens 3621.5
Schmitz 3750.0
```

```
Schmitz Peter
```

```
Mertens 3621.5
```

Es werden vier voneinander unabhängige SQL-Abfragen ausgeführt.

Wenn Sie nur die Werte bestimmter Felder sehen möchten, notieren Sie die Namen dieser Felder zwischen `SELECT` und `FROM`. Die Anweisung `SELECT name, vorname FROM personen` liefert nur die Inhalte der beiden genannten Felder aller Datensätze der Tabelle.

Mithilfe von `WHERE` können Sie die Auswahl der Datensätze einschränken, indem Sie eine Bedingung mit Vergleichsoperatoren erstellen.

Die Anweisung `SELECT * FROM personen WHERE gehalt > 3600` liefert die Inhalte aller Felder für die Datensätze, bei denen der Wert des Felds `gehalt` oberhalb der genannten Grenze liegt (siehe auch [Tabelle 10.2](#)).

Bei Zeichenketten kann ebenfalls ein Vergleich stattfinden. Dabei ist allerdings auf die *einfachen* Anführungsstriche zu achten. Die SQL-Anweisung `SELECT * FROM personen WHERE name = 'Schmitz'` liefert nur die Datensätze, bei denen der Name gleich »Schmitz« ist.

Logische Operatoren ermöglichen die Verknüpfung mehrerer Bedingungen. Die Anweisung `SELECT * FROM personen WHERE gehalt >= 3600 AND gehalt <= 3650` liefert nur die Datensätze, bei denen der Wert des Felds `gehalt` innerhalb der genannten Grenzen liegt (siehe auch [Tabelle 10.3](#)).

Operator	Erläuterung
=	gleich
<>	ungleich
>	größer als
>=	größer als oder gleich
<	kleiner als
<=	kleiner als oder gleich

Tabelle 10.2 Vergleichsoperatoren in SQL

Operator	Erläuterung
NOT	logisches NICHT: Der Wahrheitswert einer Bedingung wird umgekehrt.
AND	logisches UND: Beide Bedingungen müssen zutreffen.
OR	logisches ODER: Nur eine der Bedingungen muss zutreffen.

Tabelle 10.3 Logische Operatoren in SQL

10.2.4 Operator »LIKE«

Der Operator `LIKE` dient zum Vergleich von Zeichenketten. Das kann zur Auswahl von Datensätzen eingesetzt werden. Dabei können auch Platzhalter verwendet werden. Sie können also Abfragen bilden wie:

- Suche alle Datensätze, die mit ... beginnen.
- Suche alle Datensätze, die mit ... enden.
- Suche alle Datensätze, die ... enthalten.

Einige Beispiele:

```
import sqlite3
connection = sqlite3.connect("firma.db")
cursor = connection.cursor()

cursor.execute("SELECT * FROM personen WHERE name LIKE 'm%' ")
for dsatz in cursor:
    print(dsatz[ 0 ] , dsatz[ 1 ] )
print()

cursor.execute("SELECT * FROM personen WHERE name LIKE '%i%' ")
for dsatz in cursor:
    print(dsatz[ 0 ] , dsatz[ 1 ] )
print()
```

```

cursor.execute("SELECT * FROM personen WHERE name LIKE 'M__er' ")
for dsatz in cursor:
    print(dsatz[ 0 ] , dsatz[ 1 ] )

connection.close()

```

Listing 10.4 Datei »sqlite_like.py«

Es wird diese Ausgabe erzeugt:

```
Mertens Julia
Maier Hans
```

```
Maier Hans
Schmitz Peter
```

```
Maier Hans
```

Es werden drei voneinander unabhängige SQL-Abfragen ausgeführt.

Der Platzhalter % (Punktzeichen) steht für eine unbestimmte Anzahl beliebiger Zeichen. Die Anweisung `SELECT * FROM personen WHERE name LIKE 'm%'` liefert nur die Datensätze, bei denen der Name mit dem Buchstaben »m« beginnt (unabhängig von Groß- und Kleinschreibung).

Steht das Punktzeichen vor *und* nach einem bestimmten Zeichen, werden alle Datensätze gesucht, in denen dieses Zeichen vorkommt. Die Anweisung `SELECT * FROM personen WHERE name LIKE '%i%` liefert nur die Datensätze, in denen der Buchstabe »i« (oder »I«) vorkommt.

Der Platzhalter _ (Unterstrich) steht für ein einzelnes beliebiges Zeichen. Die Anweisung `SELECT * FROM personen WHERE name LIKE 'M__er'` liefert nur die Datensätze, bei denen der Name mit »M« beginnt und nach zwei weiteren Zeichen mit »er« endet. Dies trifft zum Beispiel auf »Maier«, »Mayer«, »Meier« und »Meyer« zu.

10.2.5 Sortierung der Ausgabe

Mithilfe von ORDER BY werden die Datensätze in sortierter Form geliefert. Es kann nach einem oder mehreren Feldern, aufsteigend oder absteigend sortiert werden. Ein Beispiel:

```
import sqlite3
connection = sqlite3.connect("firma.db")
cursor = connection.cursor()

cursor.execute("SELECT * FROM personen ORDER BY gehalt DESC")
for dsatz in cursor:
    print(dsatz[ 0 ], dsatz[ 1 ], dsatz[ 3 ])
print()

cursor.execute("SELECT * FROM personen ORDER BY name, vorname")
for dsatz in cursor:
    print(dsatz[ 0 ], dsatz[ 1 ])

connection.close()
```

Listing 10.5 Datei »sqlite_sortieren.py«

Nehmen wir an, für diese Abfrage wurde zuvor ein weiterer Datensatz hinzugefügt für die Person Wolfgang Maier mit einem Gehalt von 3.810 Euro. Dann lautet die Ausgabe:

```
Maier Wolfgang 3810.0
Schmitz Peter 3750.0
Mertens Julia 3621.5
Maier Hans 3500.0

Maier Hans
Maier Wolfgang
Mertens Julia
Schmitz Peter
```

Es werden zwei voneinander unabhängige SQL-Abfragen ausgeführt.

Die Anweisung SELECT * FROM personen ORDER BY gehalt DESC liefert alle Datensätze, absteigend nach Gehalt sortiert. Der Zusatz DESC steht für *descending* (deutsch: absteigend). Für eine aufsteigende Sortierung könnten Sie den Zusatz ASC (*ascending*, deutsch: aufsteigend) verwenden. Allerdings ist das die Standardsortierung, daher kann ASC weggelassen werden.

Die Anweisung SELECT * FROM personen ORDER BY name, vorname liefert alle Datensätze, aufsteigend nach Namen sortiert. Bei gleichem Namen wird nach Vornamen sortiert.

10.2.6 Auswahl nach Eingabe

Der Benutzer kann Datensätze auch selbst auswählen. Mit dem folgenden Programm werden nur die Datensätze angezeigt, die dem eingegebenen Namen entsprechen oder die den eingegebenen Namensteil enthalten.

```
import sqlite3
connection = sqlite3.connect("firma.db")
cursor = connection.cursor()

eingabe = input("Bitte den gesuchten Namen eingeben: ")
sql = "SELECT * FROM personen WHERE name LIKE ?"
cursor.execute(sql, (eingabe,))
for dsatz in cursor:
    print(dsatz[ 0 ], dsatz[ 1 ])
print()

eingabe = input("Bitte den gesuchten Namensteil eingeben: ")
sql = "SELECT * FROM personen WHERE name LIKE ?"
eingabe = '%' + eingabe + '%'
cursor.execute(sql, (eingabe,))
for dsatz in cursor:
    print(dsatz[ 0 ], dsatz[ 1 ])

connection.close()
```

Listing 10.6 Datei »sqlite_eingabe.py«

Die Ausgabe (mit Eingabe des Benutzers) sieht wie folgt aus:

```
Bitte den gesuchten Namen eingeben: Maier
Maier Hans

Bitte den gesuchten Namensteil eingeben: r
Mertens Julia
Maier Hans
```

Die erste Eingabe des Benutzers lautet »Maier«. Es wird ein SQL-Befehl gesendet, der diese Eingabe nutzt. Zur Abwehr einer möglichen Einschleusung von schädlichem SQL-Code (*SQL-Injection*) durch den Benutzer des Programms wird die Parameter-Substitution der Datenbankschnittstelle von Python verwendet: Für jedes Fragezeichen im SQL-Code wird ein Element des Tupels verwendet, das als zweiter, optionaler Parameter der Methode `execute()` angegeben wird. Hier gibt es nur ein Fragezeichen, daher enthält das Tupel nur ein Element, und zwar

die Eingabe. Es werden alle Datensätze mit dem Namen »Maier« geliefert.

Die zweite Eingabe des Benutzers ist das Zeichen »r«. Diese Eingabe wird zwischen die Prozentzeichen gesetzt, die als Platzhalter dienen. Es werden alle Datensätze geliefert, deren Name ein »r« enthält.

10.2.7 Datensätze ändern

Die Anweisung `UPDATE` dient zur Änderung von Inhalten in einem oder mehreren Datensätzen. Sie ähnelt in ihrem Aufbau der Anweisung `SELECT`. Wählen Sie Ihre Auswahlkriterien sorgfältig, damit Sie nicht versehentlich die falschen Änderungen vornehmen.

```
sql = "UPDATE personen SET gehalt = 3800"
```

Diese Anweisung würde bei sämtlichen Datensätzen der Tabelle `personen` den Wert für das Feld `gehalt` auf den Wert 3800 setzen – diese Aktion ist sicherlich nicht beabsichtigt.

Nehmen Sie daher eine Einschränkung vor, am besten über das Feld mit dem eindeutigen Index (hier das Feld `personalnummer`).

```
sql = "UPDATE personen SET gehalt = 3780 WHERE personalnummer = 81343"
```

Das obige Beispiel sieht, mit einer Ausgabe vor der Änderung und einer Ausgabe nach der Änderung, wie folgt aus:

```
import sqlite3
def ausgabe():
    sql = "SELECT * FROM personen"
    cursor.execute(sql)
    for dsatz in cursor:
        print(dsatz[ 0 ] , dsatz[ 1 ] , dsatz[ 2 ] , dsatz[ 3 ] )
    print()

connection = sqlite3.connect("firma.db")
cursor = connection.cursor()
ausgabe()

sql = "UPDATE personen SET gehalt = 3780 WHERE personalnummer = 81343"
cursor.execute(sql)
connection.commit()
```

```
ausgabe()
connection.close()
```

Listing 10.7 Datei »sqlite_aendern.py«

Folgende Ausgabe wird erzeugt:

```
Mertens Julia 2297 3621.5
Maier Hans 6714 3500.0
Schmitz Peter 81343 3750.0
```

```
Mertens Julia 2297 3621.5
Maier Hans 6714 3500.0
Schmitz Peter 81343 3780.0
```

Wie Sie sehen, wird nur ein Datensatz verändert. Nachfolgend ein Beispiel für die Änderung mehrerer Feldinhalte eines ausgewählten Datensatzes:

```
sql = "UPDATE personen SET gehalt = 3780, vorname = 'Hans-Peter' " \
      "WHERE personalnummer = 81343"
```

Es darf keine Änderung durchgeführt werden, die dazu führt, dass zwei oder mehr Datensätze den gleichen Inhalt in dem Feld haben, auf dem der eindeutige Index liegt, hier das Feld personalnummer.

So würde die Anweisung ...

```
sql = "UPDATE personen SET personalnummer = 6714 " \
      "WHERE personalnummer = 81343"
```

... zur folgenden Fehlermeldung führen:

```
sqlite3.IntegrityError: UNIQUE constraint failed: personen.personalnummer
```

Die Einträge im Feld mit dem PRIMARY KEY sind einzigartig.

10.2.8 Datensätze löschen

Mit der SQL-Anweisung `DELETE` löschen Sie Datensätze. Wählen Sie Ihre Auswahlkriterien besonders sorgfältig, damit Sie nicht versehentlich die falschen Datensätze oder alle Datensätze löschen. Ein Beispiel:

```
import sqlite3
def ausgabe():
    sql = "SELECT * FROM personen"
    cursor.execute(sql)
```

```

for dsatz in cursor:
    print(dsatz[ 0 ] , dsatz[ 1 ] , dsatz[ 2 ] , dsatz[ 3 ] )
print()

connection = sqlite3.connect("firma.db")
cursor = connection.cursor()
ausgabe()

sql = "DELETE FROM personen WHERE personalnummer = 8339"
cursor.execute(sql)
connection.commit()

ausgabe()
connection.close()

```

Listing 10.8 Datei »sqlite_loeschen.py«

Nehmen wir an, auch für diese Abfrage wurde zuvor ein weiterer Datensatz hinzugefügt für die Person Wolfgang Maier mit der Personalnummer 8339 und einem Gehalt von 3.810 Euro. Die Ausgabe lautet:

```

Mertens Julia 2297 3621.5
Maier Hans 6714 3500.0
Maier Wolfgang 8339 3810.0
Schmitz Peter 81343 3780.0

Mertens Julia 2297 3621.5
Maier Hans 6714 3500.0
Schmitz Peter 81343 3780.0

```

Der Datensatz mit der Personalnummer 8339 wird gelöscht. Das zeigt der Vergleich der Ausgaben vor dem Löschen und nach dem Löschen.

10.3 SQLite auf dem Webserver

Sie können eine SQLite-Datenbank auch auf einem Webserver zur dauerhaften Speicherung von Daten einsetzen. In diesem Abschnitt greifen wir auf eine SQLite-Datenbank über ein Python-CGI-Interface zu. Dabei nutzen Sie Ihre Kenntnisse aus [Abschnitt 9.2](#), »Webserver-Programmierung«, und aus [Abschnitt 10.2](#), »SQLite«.

Da es sich bei einer SQLite-Datenbank um eine einfache Textdatei handelt, sollte diese in einem eigenen, per Passwort geschützten Verzeichnis liegen. Damit verhindern Sie, dass die Datei einfach komplett heruntergeladen werden kann. Sie sollte nur über das von Ihnen bereitgestellte Python-CGI-Interface genutzt werden können.

Es folgt das Programm:

```
#!/C:/Python/python.exe
import sqlite3
print("Content-type: text/html")
print()

print("<!DOCTYPE html><html>")
print("<head><meta charset=' utf-8'><title>Datenbank</title></head>")
print("<body>")

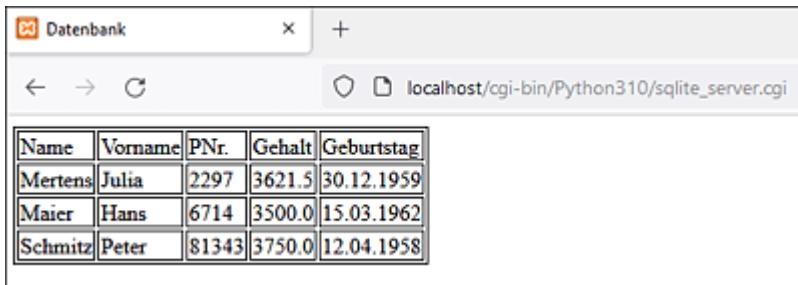
connection = sqlite3.connect("sqlite/firma.db")
cursor = connection.cursor()
cursor.execute("SELECT * FROM personen")

print("<table style=' border:1px solid black;' >")
for name in "Name", "Vorname", "PNr.", "Gehalt", "Geburtstag":
    print(f"<td style=' border:1px solid black;' >{name}</td>")
for dsatz in cursor:
    print("<tr>")
    for feld in dsatz:
        print(f"<td style=' border:1px solid black;' >{feld}</td>")
    print("</tr>")
print("</table>")

connection.close()
print("</body>")
print("</html>")
```

Listing 10.9 Datei »sqlite_server.cgi«

Abbildung 10.1 zeigt die Ausgabe im Browser nach Eingabe der Adresse http://localhost/cgi-bin/Python310/sqlite_server.cgi.



The screenshot shows a web browser window titled "Datenbank". The address bar displays "localhost/cgi-bin/Python310/sqlite_server.cgi". The main content area shows a table with the following data:

Name	Vorname	PNr.	Gehalt	Geburtstag
Mertens	Julia	2297	3621.5	30.12.1959
Maier	Hans	6714	3500.0	15.03.1962
Schmitz	Peter	81343	3750.0	12.04.1958

Abbildung 10.1 Datenbankinhalte auf dem Webserver

Die Datei *sqlite_server.cgi* mit dem Python-CGI-Programm befindet sich im Festplattenverzeichnis *C:\xampp\cgi-bin\Python310*. Der Aufbau und die Erstellung der SQLite-Datenbankdatei *firma.db* sind aus dem vorherigen Abschnitt bekannt. Die Datei liegt im Unterverzeichnis *C:\xampp\cgi-bin\Python310\sqlite*. Dieses Verzeichnis sollte geschützt werden. Die Standardprogramme zur Verwaltung eigener Websites bieten Möglichkeiten zum Schutz von Verzeichnissen.

Das HTML-Dokument wird begonnen. Es wird eine Verbindung zur Datenbank aufgenommen. Eine Datenbankabfrage wird abgesendet, das Ergebnis steht über den Datensatz-Cursor zur Verfügung.

Nun wird eine HTML-Tabelle mit Rahmen erstellt. Es folgt die erste Tabellenzeile mit der Überschrift. Sie enthält die Bezeichnungen der einzelnen Felder. Für jeden Datensatz wird jeweils eine Tabellenzeile erzeugt. In den einzelnen Zellen der Tabellenzeile stehen die Werte des Datensatzes. Die HTML-Tabelle wird beendet.

Die Verbindung zur Datenbank wird geschlossen, und das HTML-Dokument wird beendet.

10.4 MySQL

MySQL ist ein komplexes Datenbankmanagementsystem. In diesem Abschnitt lernen Sie es mithilfe einiger Programme kennen. Bezuglich der Beispiele in diesem Buch stellt es keinen Unterschied dar, ob mit MySQL oder mit seiner Abspaltung MariaDB gearbeitet wird.

Die SQL-Befehle zur Auswahl bzw. Änderung von Daten kennen Sie bereits aus [Abschnitt 10.2](#), »SQLite«. Daher wird in diesem Abschnitt nur eine Datenbank mit einer Tabelle und Datensätzen erzeugt und auf einfache Weise abgefragt.

MySQL bietet eine ganze Reihe von Datentypen, unter anderem:

- `VARCHAR` – für Zeichenketten
- `INT` – für ganze Zahlen
- `DOUBLE` – für Zahlen mit Nachkommastellen
- `DATE` – für Datumsangaben

10.4.1 XAMPP und Connector/Python

Zum Testen der Programme wird ein lokaler MySQL-Datenbankserver benötigt. Diesen finden Sie im Paket XAMPP, das bereits vorkonfiguriert und einfach zu installieren ist. Es umfasst neben einem MySQL-Datenbankserver einen *Apache-Webserver*, die Webserver-Sprache *PHP*, das Datenbankverwaltungsprogramm *phpMyAdmin* und weitere Software.

Sie erreichen die Website zum Herunterladen von XAMPP über die Adresse <https://www.apachefriends.org>, sowohl für Windows als auch für Ubuntu Linux und für macOS. Die Installation von

XAMPP und das Starten der beiden Server werden in [Abschnitt A.3](#), »Installation von XAMPP«, beschrieben.

Der Treiber *Connector/Python* bietet eine Schnittstelle zwischen Python und MySQL. Sie installieren ihn mit dem Paketverwaltungsprogramm *pip* (siehe [Abschnitt A.1](#)) wie folgt:

```
pip install mysql-connector-python
```

10.4.2 Datenbank, Tabelle und Datensätze

Im folgenden Programm wird, ähnlich wie für SQLite in [Abschnitt 10.2.1](#), »Datenbank, Tabelle und Datensätze«, zunächst eine MySQL-Datenbank erzeugt und eine Tabelle mit einem eindeutigen Index angelegt. Anschließend werden drei Datensätze in der Tabelle angelegt:

```
import sys, mysql.connector
try:
    connection = mysql.connector.connect \
        (host = "localhost", user = "root", passwd = "")
except:
    print("Keine Verbindung zum Server")
    sys.exit(0)

cursor = connection.cursor()

cursor.execute("CREATE DATABASE IF NOT EXISTS firma")
connection.commit()

cursor.execute("USE firma")
connection.commit()

cursor.execute("CREATE TABLE IF NOT EXISTS personen"
               "(name VARCHAR(30), vorname VARCHAR(25),"
               "personalnummer INT(11), gehalt DOUBLE, "
               "geburtstag DATE, PRIMARY KEY (personalnummer))"
               "ENGINE=InnoDB DEFAULT CHARSET=UTF8")
connection.commit()

try:
    cursor.execute("INSERT INTO personen VALUES(' Maier' , " \
                   "' Hans' , 6714, 3500, '1962-03-15' )")
    connection.commit()
    cursor.execute("INSERT INTO personen VALUES(' Schmitz' , " \
                   "' Peter' , 81343, 3750, '1958-04-12' )")
    connection.commit()
    cursor.execute("INSERT INTO personen VALUES(' Mertens' , " \
                   "' Julia' , 2297, 3621.5, '1959-12-30' )")
    connection.commit()
except:
    print("Fehler bei der Verarbeitung der Daten")
```

```
    connection.commit()
except:
    print("Fehler beim Erstellen")

cursor.close()
connection.close()
```

Listing 10.10 Datei »mysql_erzeugen.py«

Nach dem Import des Moduls `mysql.connector` wird mithilfe der Funktion `connect()` eine Verbindung zum Datenbankserver hergestellt. Dabei werden der Name des Servers (hier: `localhost`), der Name des Benutzers (hier: `root`) und das Passwort (hier: kein Passwort) angegeben. Läuft der MySQL-Datenbankserver nicht, schlägt die Verbindungsaufnahme fehl. Der Rückgabewert der Funktion `connect()` ist ein `connection`-Objekt. Über diese Verbindung kann auf die Datenbank zugegriffen werden.

Anschließend wird die Methode `cursor()` genutzt, um ein `cursor`-Objekt zu erstellen. Über diesen Cursor können SQL-Befehle an die Datenbank gesendet und die Ergebnisse empfangen werden.

Die Methode `execute()` dient zum Senden von SQL-Befehlen. Der SQL-Befehl `CREATE DATABASE firma` erzeugt eine neue Datenbank mit dem Namen `firma`. Der Zusatz `IF NOT EXISTS` sorgt dafür, dass sie nur erzeugt wird, wenn sie noch nicht existiert.

Der Aufruf der Methode `commit()` des `connection`-Objekts führt dazu, dass die SQL-Anweisung unmittelbar ausgeführt wird.

Mithilfe des SQL-Befehls `USE firma` wird die Datenbank `firma` nach ihrer Erzeugung ausgewählt.

Die Tabelle `personen` wird mit fünf Feldern erzeugt. Die Felder `name` und `vorname` sind vom Typ `VARCHAR`. Es wird eine Länge von maximal 30 bzw. 25 Zeichen gewählt. Das Feld `personalnummer` ist vom Typ `INT`. Die Angabe 11 ist dabei der Standardwert. Das Feld `gehalt` ist vom Typ `DOUBLE` für Zahlen mit Nachkommastellen. Das Feld `geburtstag` ist vom Typ `DATE` für Datumsangaben. Der Zusatz `IF NOT EXISTS` bewirkt auch hier, dass die Tabelle nur erzeugt wird, wenn sie noch nicht existiert.

Auf dem Feld `personalnummer` wird ein Primärschlüssel definiert. Damit wird dafür gesorgt, dass es keine zwei Einträge geben kann, die dieselbe Personalnummer haben.

Mithilfe des SQL-Befehls `INSERT INTO` werden drei Datensätze erzeugt. Die Daten werden in der Reihenfolge der Felder angegeben. Zeichenketten stehen in einfachen Anführungsstrichen. Nachkommastellen werden mit einem Dezimalpunkt abgetrennt. Eine Datumsangabe erfolgt in der Form `JJJJ-MM-TT` und ebenfalls innerhalb von Anführungsstrichen.

Ein zweiter Aufruf desselben Programms würde zu einer Fehlermeldung führen, da auf dem Feld `personalnummer` ein Primärschlüssel definiert ist. Es kann kein Eintrag erzeugt werden, der dieselbe Personalnummer wie ein bereits existierender Datensatz hat.

Am Ende dieser Aktionen werden das `cursor`-Objekt und das `connection`-Objekt jeweils mit der Methode `close()` wieder geschlossen.

Das Programm hat keine Ausgabe. XAMPP enthält das Programm phpMyAdmin zur Verwaltung der Datenbanken des MySQL-Servers. Mithilfe dieses Programms können Sie sich von der Existenz der neuen Datenbank, der Tabelle und der Datensätze überzeugen.

10.4.3 Daten anzeigen

Mithilfe des nachfolgenden Programms werden alle Datensätze der Tabelle mit allen Inhalten angezeigt:

```
import sys, mysql.connector
try:
    connection = mysql.connector.connect(host = "localhost", \
                                          user = "root", passwd = "", db = "firma")
except:
    print("Keine Verbindung zum Server")
    sys.exit(0)
```

```
cursor = connection.cursor()
cursor.execute("SELECT * FROM personen")
result = cursor.fetchall()
cursor.close()
connection.close()

for datensatz in result:
    for feld in datensatz:
        print(f"{feld} ", end="")
    print()
```

Listing 10.11 Datei »mysql_anzeigen.py«

Das Programm erzeugt die Ausgabe:

```
Mertens Julia 2297 3621.5 1959-12-30
Maier Hans 6714 3500.0 1962-03-15
Schmitz Peter 81343 3750.0 1958-04-12
```

Die Funktion `connect()` wird mit einem vierten Parameter aufgerufen. Auf diese Weise wird nicht nur eine Verbindung zum Datenbankserver hergestellt, sondern auch zu der Datenbank auf dem Server.

Mithilfe des SQL-Befehls `SELECT` werden alle Datensätze abgefragt. Da es sich nicht um eine Aktionsabfrage handelt, sondern nur um eine Auswahlabfrage, muss hier die Methode `commit()` nicht aufgerufen werden.

Die Methode `fetchall()` des `cursor`-Objekts liefert als Rückgabewert eine Liste, die das Ergebnis der Abfrage enthält. Die einzelnen Datensätze mit den Werten der Felder werden ausgegeben.

10.5 Spiel, Version mit Highscore-Datenbank

Es folgt eine weitere Version des Kopfrechenspiels. Darin werden die Daten des Spielers (Name und Zeit) dauerhaft in einer SQLite-Datenbank gespeichert, sodass eine Highscore-Liste geführt werden kann.

Der Unterschied zur Version mit der Highscore-Datei (siehe [Abschnitt 8.11, »Spiel, Version mit Highscore-Datei«](#)), besteht darin, dass keine eigene Liste geführt werden muss. Stattdessen werden die neuen Werte direkt und »unsortiert« in die Datenbank geschrieben. Später werden die vorhandenen Werte direkt und in sortierter Form aus der Datenbank gelesen und auf dem Bildschirm ausgegeben.

Dies hat einige Vorteile:

- Es muss kein Austausch zwischen Liste und Datei stattfinden.
- Die Sortierung wird von SQL übernommen und muss nicht innerhalb des Python-Programms erfolgen.
- Bei einem vorzeitigen Abbruch des Programms gehen keine Daten verloren, da sie nach jedem Spiel gespeichert werden.

Es folgt der erste Teil des Programms:

```
import random, time, glob, sqlite3

def hs_anzeigen():
    if not glob.glob("highscore.db"):
        print("Keine Highscores vorhanden")
        return

    con = sqlite3.connect("highscore.db")
    cursor = con.cursor()
    sql = "SELECT * FROM daten ORDER BY zeit LIMIT 10"
    cursor.execute(sql)

    print(" P. Name          Zeit")
    i = 1
    for dsatz in cursor:
```

```

    print(f" {i:2d}. {dsatz[ 0 ] :10} {dsatz[ 1 ] :5.2f} sec")
    i = i+1
con.close()

```

Listing 10.12 Datei »spiel_sqlite.py«, Teil 1 von 3

In der Funktion `hs_anzeigen()` wird festgestellt, ob die Datenbankdatei existiert. Ist dies der Fall, wird eine Verbindung aufgenommen.

Mithilfe der Anweisung `SELECT` werden die ersten zehn Datensätze ausgewählt, aufsteigend sortiert nach der Spieldauer. Der Zusatz `LIMIT` in der SQL-Abfrage begrenzt dabei die Anzahl. Nach der Ausgabe der Datensätze wird die Verbindung wieder beendet.

Der zweite Teil des Programms:

```

def spiel():
    ...
    if not glob.glob("highscore.db"):
        con = sqlite3.connect("highscore.db")
        cursor = con.cursor()
        sql = "CREATE TABLE daten(name TEXT, zeit REAL)"
        cursor.execute(sql)
        con.close()

        con = sqlite3.connect("highscore.db")
        cursor = con.cursor()
        sql = f"INSERT INTO daten VALUES(' {name}' , {differenz})"
        cursor.execute(sql)
        con.commit()
        con.close()

    hs_anzeigen()

```

Listing 10.13 Datei »spiel_sqlite.py«, Teil 2 von 3

Der Anfang der Funktion `spiel()` unterscheidet sich nicht von dem der Version mit der Highscore-Datei (siehe [Abschnitt 8.11](#), »Spiel, Version mit Highscore-Datei«). Erst bei der Speicherung zeigen sich die Änderungen.

Es wird zunächst festgestellt, ob die Datenbankdatei existiert. Ist dies nicht der Fall, wird die Datenbank mit der Tabelle erzeugt. Die neu ermittelten Werte werden als neuer Datensatz in die Datenbank geschrieben. Anschließend wird die Highscore-Liste angezeigt.

Das Hauptprogramm:

```
random.seed()

while True:
    try:
        menu = int(input("Bitte eingeben "
                          "(0: Ende, 1: Highscores, 2: Spielen): "))
    except:
        print("Falsche Eingabe")
        continue

    if menu == 0:
        break
    elif menu == 1:
        hs_anzeigen()
    elif menu == 2:
        spiel()
    else:
        print("Falsche Eingabe")
```

Listing 10.14 Datei »spiel_sqlite.py«, Teil 3 von 3

Gegenüber der Version mit Highscore-Datei (siehe [Abschnitt 8.11](#)) fehlen die Aufrufe der beiden Funktionen zum Austausch der Daten zwischen Liste und Datei. Im Hauptprogramm läuft eine Endlosschleife. Der Benutzer kann sich die Highscores anzeigen lassen, spielen oder das Programm beenden.

10.6 Spiel, objektorientierte Version mit Highscore-Datenbank

Diese Version des Kopfrechenspiels ist auf der objektorientierten Version mit Highscore-Datei aus [Abschnitt 8.12](#), »Spiel, objektorientierte Version mit Highscore-Datei«, aufgebaut. Nur die Klasse `Highscore` wurde verändert, damit die Vorteile einer Datenbank genutzt werden können:

```
import random, time, glob, sqlite3
...
class Highscore:
    def speichern(self, name, zeit):
        if not glob.glob("highscore.db"):
            con = sqlite3.connect("highscore.db")
            cursor = con.cursor()
            sql = "CREATE TABLE daten(name TEXT, zeit REAL)"
            cursor.execute(sql)
            con.close()

        con = sqlite3.connect("highscore.db")
        cursor = con.cursor()
        sql = f"INSERT INTO daten VALUES ('{name}', {zeit})"
        cursor.execute(sql)
        con.commit()
        con.close()

    def __str__(self):
        if not glob.glob("highscore.db"):
            return "Keine Highscores vorhanden"

        con = sqlite3.connect("highscore.db")
        cursor = con.cursor()
        sql = "SELECT * FROM daten ORDER BY zeit LIMIT 10"
        cursor.execute(sql)

        ausgabe = " P. Name           Zeit\n"
        i = 1
        for dsatz in cursor:
            ausgabe += f"{i:2d}. {dsatz[0]:10} {dsatz[1]:5.2f} sec\n"
            i = i+1

        con.close()
        return ausgabe
```

Listing 10.15 Datei »spiel_sqlite_oop.py«

In der Klasse `Highscore` gibt es nur noch die beiden Methoden zum Speichern und zum Ausgeben des Highscores.

In der Methode `speichern()` wird geprüft, ob es eine Datenbankdatei gibt. Gibt es sie nicht, werden die Datenbank und die Tabelle erzeugt. Die neu ermittelten Daten werden hinzugefügt.

In der Ausgabemethode werden die Datensätze in sortierter Form ausgewählt und anschließend formatiert ausgegeben.

11 Benutzeroberflächen

Die bisher dargestellten Programme werden über Eingaben in der Kommandozeile bedient. Möchten Sie den Benutzern dagegen eine komfortable grafische Oberfläche zur Bedienung zur Verfügung stellen, so bietet sich bei Python die Nutzung der bewährten und weitgehend einheitlichen Bibliothek *Tk* an.

Das Modul `tkinter` stellt eine Schnittstelle (englisch: *interface*) zu dieser Bibliothek dar. Es steht nach der Installation von Python sowohl unter Windows als auch unter Ubuntu Linux und macOS bereits zur Verfügung. Es bietet eine Reihe von Klassen zur Erzeugung der Elemente der Oberfläche. Die ereignisorientierte Programmierung rückt dabei in den Vordergrund.

Eine Alternative zur Tk-Bibliothek bietet die ebenfalls weitgehend einheitliche Bibliothek *Qt*. Die Schnittstelle *PyQt* für Python ist Thema des nachfolgenden Kapitels.

11.1 Einführung

Die Erstellung einer Anwendung mit einer Benutzeroberfläche, einer *GUI* (*Graphical User Interface*), umfasst die folgenden wichtigen Schritte:

- Erstellung eines Fensters mit allen Elementen der Anwendung
- Erzeugung und Anordnung von Steuerelementen (hier *Widgets* genannt) innerhalb des Fensters zur Bedienung der Anwendung

- Start einer Endlosschleife, in der die Bedienung der Widgets durch den Benutzer zu Ereignissen und damit zum Ablauf der Programmteile führt

11.1.1 Erstes GUI-Programm

Das folgende Programm dient als Einstieg in die GUI-Programmierung:

```
import tkinter

def ende():
    fenster.destroy()

fenster = tkinter.Tk()
fenster.resizable(0, 0)

buEnde = tkinter.Button(fenster, text="Ende", command=ende, width=10)
buEnde.pack(padx=30, pady=30)

fenster.mainloop()
```

Listing 11.1 Datei »gui_erstes.py«

Das Ergebnis präsentiert sich abhängig vom verwendeten Betriebssystem unterschiedlich. In jedem Fall sieht ein Benutzer eine Oberfläche in gewohntem Design.

Die Ergebnisse für Windows 10, Ubuntu Linux und macOS zeigen [Abbildung 11.1](#) bis [Abbildung 11.3](#). Im weiteren Verlauf dieses Kapitel werden die Oberflächen angezeigt, die der Benutzer unter Windows 10 sieht.

Betätigt der Benutzer den Button ENDE, wird die Anwendung beendet.

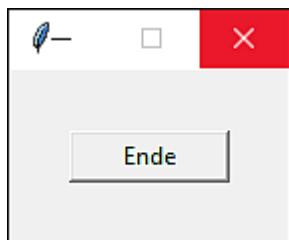


Abbildung 11.1 Erstes Programm unter Windows 10

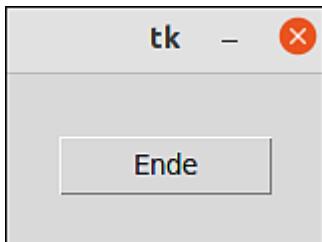


Abbildung 11.2 Erstes Programm unter Ubuntu Linux



Abbildung 11.3 Erstes Programm unter macOS

Das importierte Modul `tkinter` wird mit seinen Klassen zur Erzeugung der Oberflächenelemente benötigt. Es wird die Funktion `ende()` definiert, in der das Anwendungsfenster mithilfe der Methode `destroy()` geschlossen und damit die gesamte Anwendung beendet wird.

Es wird ein Objekt der Klasse `Tk` erzeugt. Dieses Objekt entspricht einem Fenster – in diesem Fall dem Anwendungsfenster, das alle Widgets der Anwendung enthält. Der Konstruktor der Klasse `Tk` liefert eine Referenz auf das Objekt zurück, die in der Variablen `fenster` gespeichert wird. Über diese Variable können Sie auf das Anwendungsfenster zugreifen.

Falls Sie die Methode `resizable()` mit den Werten 0 und 0 aufrufen, legen Sie damit fest, dass die Größe des Anwendungsfensters nicht geändert werden kann.

Es wird ein Button-Widget erzeugt, also ein Objekt der Klasse `Button`. Es dient als Schaltfläche zum Auslösen von Aktionen. Mithilfe der Parameter des Konstruktors werden die Eigenschaften eines Widgets festgelegt. Viele Eigenschaften eines Widgets können später noch verändert werden.

Der erste Parameter gibt an, in welches umgebende Element der Button eingebettet wird – in diesem Fall in das

Anwendungsfenster. Sie können über die Variable `buEnde` auf den Button zugreifen.

Der benannte Parameter `text` legt die Aufschrift des Buttons fest. Der benannte Parameter `command` bestimmt den Namen der Callback-Funktion, die ausgeführt wird, falls der Button betätigt wird, siehe auch [Abschnitt 5.6.9](#), »Funktion als Parameter«. Hier handelt es sich um die Funktion `ende()`. Der Name der Funktion wird dabei ohne Klammern angegeben. Mit dem benannten Parameter `width` legen Sie die Breite des Buttons fest. Der Rückgabewert des Konstruktors ist eine Referenz auf das `Button`-Objekt.

Ein Widget erscheint erst im Anwendungsfenster, nachdem es mithilfe eines Geometrie-Managers angeordnet wurde. Durch den Aufruf der Methode `pack()` wird der Geometrie-Manager `pack` aufgerufen. Er ordnet Widgets auf einfache Weise nacheinander im umgebenden Element an.

Mit den Parametern `padx` und `pady` können Sie einen Abstand in X-Richtung bzw. in Y-Richtung zwischen dem Widget und dem Rand des umgebenden Elements erzeugen.

Die Größe des Anwendungsfensters richtet sich bei der Nutzung des Geometrie-Managers `pack` nach den enthaltenen Widgets, unter Beachtung des Abstands.

Am Ende wird mit der Methode `mainloop()` eine Endlosschleife für das Anwendungsfenster aufgerufen. Sie sorgt dafür, dass es dauerhaft angezeigt wird, Benutzerereignisse empfängt und diese weiterleitet.

Hinweis

Sie können GUI-Anwendungen wie andere Anwendungen entweder aus IDLE heraus oder über die Kommandozeile starten.

11.1.2 Anordnung von Widgets

Mithilfe des nachfolgenden Programms lernen Sie mehr über die Anordnung von Widgets im Anwendungsfenster:

```
import tkinter

def hallo():
    lbAusgabe[ "text"] = "Hallo"

def ende():
    fenster.destroy()

fenster = tkinter.Tk()
fenster.title("GUI")
fenster.resizable(0, 0)

buHallo = tkinter.Button(fenster, text="Hallo", command=hallo, width=10)
buHallo.grid(row=0, column=0, sticky="w", padx=5, pady=5)

lbAusgabe = tkinter.Label(fenster, text="(leer)",
    anchor="w", relief="sunken", width=20)
lbAusgabe.grid(row=1, column=0, padx=5, pady=5)

buEnde = tkinter.Button(fenster, text="Ende", command=ende, width=10)
buEnde.grid(row=2, column=0, padx=5, pady=5)

fenster.mainloop()
```

Listing 11.2 Datei »gui_anordnung.py«

Die Ausgabe des Programms sehen Sie in [Abbildung 11.4](#).

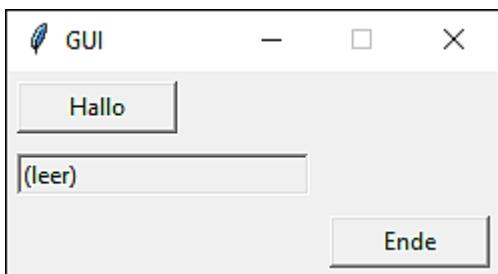


Abbildung 11.4 Anordnung von Widgets

Das Anwendungsfenster enthält drei Widgets, und zwar zwei Buttons und ein Label-Widget, also ein Objekt der Klasse `Label`. Es dient zur Ausgabe eines Texts oder zur Anzeige eines Bilds und kann hier mithilfe der Variablen `lbAusgabe` erreicht werden. Die Parameter `text` und `width` haben dieselbe Bedeutung wie bei einem Button.

Betätigt die Benutzerin den Button `HALLO`, erscheint der Text »Hallo« im Ausgabefeld darunter. Dazu wird der Eigenschaft `text` des Labels in der Funktion `hallo()` ein neuer Wert zugewiesen.

Für das Anwendungsfenster wird die Methode `title()` aufgerufen. Sie dient zur Zuweisung des Titels, der in der Kopfzeile des Fensters zu sehen ist.

Für die drei Widgets wird die Methode `grid()` aufgerufen, zur Nutzung des Geometrie-Managers `grid` mit seinen vielfältigen Möglichkeiten. Dabei werden die Widgets in einem `grid` (deutsch: Gitter oder Raster) angeordnet, ähnlich wie in den Zellen einer Tabelle. Mit den Parametern `row` (deutsch: Zeile) und `column` (deutsch: Spalte) bestimmen Sie die Position der Zelle innerhalb des Rasters. Die Nummerierung beginnt jeweils bei 0.

Ist das umgebende Element (hier: die Zelle) größer als ein Widget, können Sie mithilfe des Parameters `sticky` (deutsch: klebend) festlegen, zu welchem Rand des umgebenden Elements das Widget orientiert ist.

Mögliche Werte sind:

- `w`: zum westlichen, also zum linken Rand
- `e`: zum östlichen, also zum rechten Rand
- `n`: zum nördlichen, also zum oberen Rand
- `s`: zum südlichen, also zum unteren Rand

Es können auch Kombinationen aus diesen Werten gebildet werden. Wird `sticky` nicht verwendet, wird das Widget im Zentrum der Zelle platziert.

Im vorliegenden Beispiel ist das Label breiter als der Button und bestimmt damit die Breite der linken Spalte. Der Button wird mithilfe des Parameters `sticky` am linken Rand der Zelle angeordnet.

Mit dem Parameter `relief` können Sie die Darstellung eines Widgets beeinflussen. Mögliche Werte sind:

- `flat`: Das Label wird flach dargestellt, ohne erkennbaren Rand.
- `groove`: Das Label erhält einen dünnen vertieften Rahmen.
- `ridge`: Das Label erhält einen dünnen erhobenen Rahmen.
- `raised`: Das Label wird als Ganzes erhoben dargestellt wie ein Button.
- `sunken`: Das Label wird als Ganzes vertieft dargestellt.
- `solid`: Das Label erhält einen deutlichen schwarzen Rahmen.

Ist ein Widget größer als der darin dargestellte Text, können Sie mithilfe des Parameters `anchor` (deutsch: Anker) festlegen, an welcher Seite des Widgets der Text verankert wird, wie also der Text ausgerichtet wird. Die möglichen Werte sind dieselben wie beim Parameter `sticky`, also `w`, `e`, `n`, `s` oder eine Kombination aus diesen Werten. Wird `anchor` nicht verwendet, wird der Text im Zentrum des Widgets platziert.

Die Größe des Anwendungsfensters richtet sich nach den enthaltenen Widgets, unter Beachtung des Abstands.

11.2 Widget-Typen

In diesem Abschnitt lernen Sie weitere Widgets kennen: Entry, Text, Listbox, Spinbox, Radiobutton, Checkbutton und Scale.

11.2.1 Einzeiliges Eingabefeld

Ein Entry-Widget ist ein einzeiliges Eingabefeld und dient zur Eingabe von kurzen Texten oder Zahlen durch den Benutzer. Das folgende Beispiel verdeutlicht die Verarbeitung von eingegebenen Daten. Der Benutzer trägt eine Zahl in ein Entry-Widget ein. Sie wird verdoppelt und ausgegeben:

```
import tkinter

def verdoppeln():
    try:
        zahl = float(etEingabe.get())
        lbAusgabe[ "text"] = zahl * 2
    except:
        lbAusgabe[ "text"] = "Keine Zahl"

def ende():
    fenster.destroy()

fenster = tkinter.Tk()
fenster.title("Einzeilig")
fenster.resizable(0, 0)

lbEingabe = tkinter.Label(fenster, text="Ihre Eingabe:")
lbEingabe.grid(row=0, column=0, sticky="w", padx=5, pady=5)

etEingabe = tkinter.Entry(fenster)
etEingabe.grid(row=1, column=0, padx=5, pady=5)

buVerdoppeln = tkinter.Button(fenster, text="Verdoppeln",
    command=verdoppeln, width=10)
buVerdoppeln.grid(row=2, column=0, sticky="w", padx=5, pady=5)

lbAusgabe = tkinter.Label(fenster, text="(leer)")
lbAusgabe.grid(row=3, column=0, sticky="w", padx=5, pady=5)

buEnde = tkinter.Button(fenster, text="Ende", command=ende, width=10)
buEnde.grid(row=4, column=0, padx=5, pady=5)

fenster.mainloop()
```

Listing 11.3 Datei »gui_eingabe.py«

Nach der Eingabe einer Zahl und der Betätigung des Buttons VERDOPPELN wird das Ergebnis angezeigt, siehe [Abbildung 11.5](#).

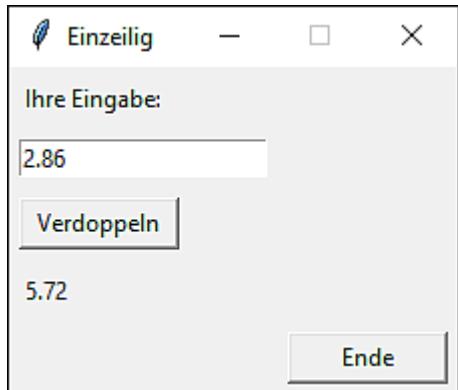


Abbildung 11.5 Einzeiliges Eingabefeld

Für die beiden Label wird keine Breite festgelegt. Daher sind sie nur so breit wie ihr Inhalt. Die Breite der linken Spalte wird von der Breite des Eingabefelds bestimmt. Die beiden Label und der Button werden mithilfe des Parameters `sticky` am linken Rand ihrer Zelle angeordnet.

Ein einzeiliges Eingabefeld ist ein Objekt der Klasse `Entry`. Es wird der Variablen `etEingabe` zugeordnet.

Bei Betätigung des Buttons VERDOPPELN wird die Funktion `verdoppeln()` aufgerufen. Die Methode `get()` liefert eine Zeichenkette mit der Eingabe des Benutzers. Kann die Eingabe nicht mithilfe der Funktion `float()` in eine Zahl umgewandelt werden, tritt eine Ausnahme auf, und es erscheint eine Fehlermeldung. Ansonsten wird der Wert der Zahl verdoppelt und ausgegeben.

11.2.2 Versteckte Eingabe, Widget deaktivieren

Ein Entry-Widget kann auch die Eingabe von versteckten Informationen, zum Beispiel Passwörtern, unterstützen. Dazu dient der Parameter `show`, in dem festgelegt wird, welches Zeichen

in einem Entry-Widget nach der Eingabe des Benutzers angezeigt wird, siehe [Abbildung 11.6](#).

Gibt der Benutzer im nachfolgenden Beispielprogramm das richtige Passwort ein, erhält er die Rückmeldung *Zugang erlaubt*, anderenfalls die Meldung *Zugang verweigert*.

Unabhängig davon können Sie zur Verbesserung der Benutzerführung ein Widget in Abhängigkeit vom Programmzustand aktivieren oder deaktivieren. Im nachfolgenden Programm ist der Button `ENDE` zunächst deaktiviert. Erst nach einer Prüfung des Passworts wird der Button aktiviert:

```
import tkinter

def pruefen():
    pw = etPasswort.get()
    if pw == "Bingo":
        lbAusgabe[ "text"] = "Zugang erlaubt"
    else:
        lbAusgabe[ "text"] = "Zugang nicht erlaubt"
    etPasswort.delete(0, "end")
    buEnde[ "state"] = "normal"

def ende():
    fenster.destroy()

fenster = tkinter.Tk()
fenster.title("Passwort")
fenster.resizable(0, 0)

lbPasswort = tkinter.Label(fenster, text="Ihr Passwort:")
lbPasswort.grid(row=0, column=0, sticky="w", padx=5, pady=5)

etPasswort = tkinter.Entry(fenster, show="*")
etPasswort.grid(row=1, column=0, padx=5, pady=5)

buPruefen = tkinter.Button(fenster, text="Prüfen",
                           command=pruefen, width=10)
buPruefen.grid(row=2, column=0, sticky="w", padx=5, pady=5)

lbAusgabe = tkinter.Label(fenster, text="(leer)")
lbAusgabe.grid(row=3, column=0, sticky="w", padx=5, pady=5)

buEnde = tkinter.Button(fenster, text="Ende", command=ende,
                        width=10, state="disabled")
buEnde.grid(row=4, column=0, padx=5, pady=5)

fenster.mainloop()
```

Listing 11.4 Datei »gui_passwort.py«

Der Parameter `show` des Entry-Widgets erhält den Wert `*`. Daher wird für jedes eingegebene Zeichen ein Stern angezeigt, wie bei Passwörtern üblich.

Bei Betätigung des Buttons **PRÜFEN** wird die Funktion `pruefen()` aufgerufen. Die Methode `get()` liefert die Eingabe. Es wird geprüft, ob diese dem richtigen Passwort entspricht.

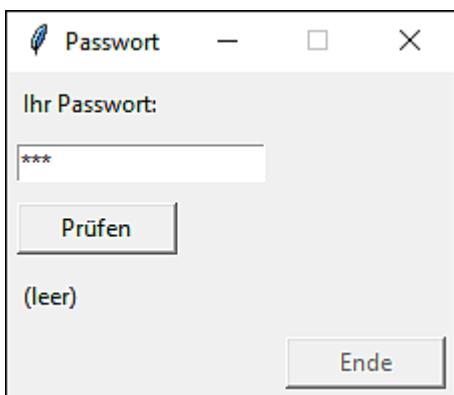


Abbildung 11.6 Versteckte Eingabe, Widget deaktivieren

Die Methode `delete()` dient zum Löschen der Zeichenkette, die in einem Entry-Widget steht. Der zu löschen Bereich der Zeichenkette beginnt mit dem Index, der im ersten Parameter steht, und endet vor dem Index, der im zweiten Parameter steht. Wird als zweiter Parameter der Wert »end« angegeben, wird bis zum letzten Zeichen einschließlich gelöscht.

Der Button wird mit dem Wert »disabled« für den Parameter `state` erstellt. Damit ist er deaktiviert und kann nicht bedient werden. Weisen Sie der Eigenschaft `state` den Standardwert »normal« zu, wird der Button aktiviert.

11.2.3 Mehrzeiliges Eingabefeld

Ein Text-Widget ist ein mehrzeiliges Eingabefeld und dient zur Eingabe und Darstellung größerer Textmengen. Gegenüber dem Entry-Widget bietet es mehr Möglichkeiten zur Verarbeitung von Daten.

Das Modul `tkinter.scrolledtext` bietet mit dem ScrolledText-Widget eine Weiterentwicklung eines einfachen Text-Widgets. Es ermöglicht mit einem Scrollbalken das vertikale Scrollen des mehrzeiligen Eingabefelds.

Im folgenden Programm hat der Benutzer die Möglichkeit, den Inhalt einer Textdatei in ein ScrolledText-Widget zu laden bzw. den Inhalt des ScrolledText-Widgets in einer Textdatei zu speichern:

```
import tkinter, tkinter.scrolledtext

def laden():
    txText.delete(1.0, "end")
    try:
        d = open("gui_mehrzeilig.txt")
        txText.insert(1.0, d.read())
        d.close()
    except:
        txText.insert(1.0, "Datei nicht geöffnet")

def speichern():
    try:
        d = open("gui_mehrzeilig.txt", "w")
        d.write(txText.get(1.0, "end"))
        d.close()
    except:
        txText.insert(1.0, "Datei nicht geöffnet")

def loeschen():
    txText.delete(1.0, "end")

def ende():
    fenster.destroy()

fenster = tkinter.Tk()
fenster.title("Mehrzeilig")
fenster.resizable(0, 0)

lbText = tkinter.Label(fenster, text="Text:")
lbText.grid(row=0, column=0, sticky="w", padx=5, pady=5)

txText = tkinter.scrolledtext.ScrolledText(fenster, width=50, height=4)
txText.grid(row=1, column=0, columnspan=3, padx=5, pady=5)

buLaden = tkinter.Button(
    fenster, text="Aus Datei laden", command=laden, width=15)
buLaden.grid(row=2, column=0, sticky="w", padx=5, pady=5)

buSpeichern = tkinter.Button(
    fenster, text="In Datei speichern", command=speichern, width=15)
buSpeichern.grid(row=2, column=1, padx=5, pady=5)
```

```

buLoeschen = tkinter.Button(
    fenster, text="Text löschen", command=loeschen, width=15)
buLoeschen.grid(row=2, column=2, sticky="e", padx=5, pady=5)

buEnde = tkinter.Button(fenster, text="Ende", command=ende, width=10)
buEnde.grid(row=3, column=3, padx=5, pady=5)

fenster.mainloop()

```

Listing 11.5 Datei »gui_mehrzeilig.py«

Abbildung 11.7 zeigt die Darstellung nach Aufruf des Programms und Betätigung des Buttons **AUS DATEI LADEN**.

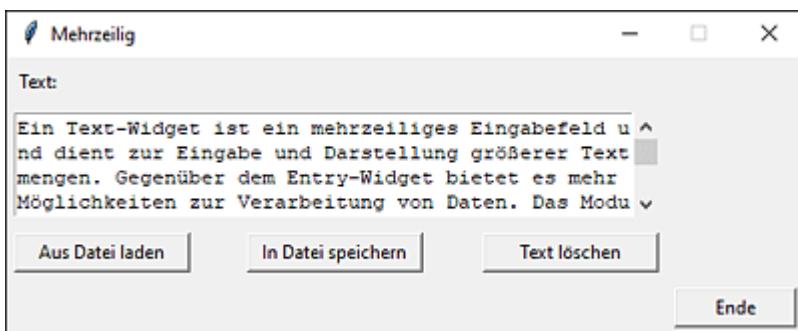


Abbildung 11.7 ScrolledText-Widget mit Inhalt

Die Betätigung des Buttons **TEXT LÖSCHEN** führt zum Löschen der Zeichenkette im ScrolledText-Widget mithilfe der Methode `delete()`. Die Zeichenkette wird hier mithilfe der Parameter `1.0` und »end« vom ersten bis zum letzten Zeichen gelöscht.

Das ScrolledText-Widget erstreckt sich dank des Parameters `columnspan`, der hier den Wert 3 hat, über drei Spalten des Rasters, das mithilfe des Geometrie-Managers `pack` erstellt wird. Die beiden Buttons **AUS DATEI LADEN** und **TEXT LÖSCHEN** sind mithilfe des Parameters `STICKY` zum linken bzw. rechten Rand ihrer jeweiligen Zelle orientiert.

Nach Betätigung des Buttons **AUS DATEI LADEN** wird die Funktion `laden()` aufgerufen. Zunächst wird die Zeichenkette im ScrolledText-Widget gelöscht.

Anschließend wird versucht, die Textdatei zu öffnen. Gelingt das, wird ihr gesamter Inhalt mithilfe der Methode `read()` gelesen und mithilfe der Methode `insert()` im ScrolledText-Widget eingefügt. Der erste Parameter gibt den Einfügepunkt an. Mit dem Wert `1.0`

wird hier wiederum der Beginn des ScrolledText-Widgets gewählt. Gelingt das Öffnen nicht, wird eine Fehlermeldung im ScrolledText-Widget angezeigt. Beachten Sie, dass die Textdatei, aus der gelesen wird, in ANSI-Kodierung vorliegen muss.

Nach Betätigung des Buttons **IN DATEI SPEICHERN** wird die Funktion `speichern()` aufgerufen. Die Textdatei wird zum Schreiben geöffnet. Der Inhalt des ScrolledText-Widgets wird mithilfe der Methode `get()` und der Parameter `1.0` und »end« vom ersten bis zum letzten Zeichen ermittelt und mithilfe der Methode `write()` in die Textdatei geschrieben.

So können Sie eine Textdatei unter Windows mithilfe des Standardeditors in ANSI-Kodierung speichern: Wählen Sie den Menüpunkt **DATEI • SPEICHERN UNTER**, klappen Sie die Liste **CODIERUNG** auf, wählen Sie den Eintrag **ANSI**, und betätigen Sie den Button **SPEICHERN**, siehe [Abbildung 11.8](#).

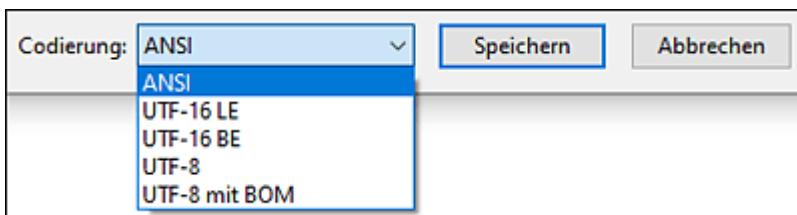


Abbildung 11.8 In ANSI-Kodierung speichern

11.2.4 Listbox mit einfacher Auswahl

Ein Listbox-Widget dient als Auswahlmenü für eine Reihe von Begriffen, aus denen der Benutzer einen oder mehrere auswählen kann.

Im folgenden Programm wird eine Listbox mit insgesamt zehn Einträgen erstellt. Sie ist mit einer Scrollbar verbunden, mit der die Auswahl eines Eintrags erleichtert wird. Listbox und Scrollbar werden in einem Frame-Widget zusammengefasst, das eine gemeinsame Anordnung ermöglicht:

```
import tkinter
```

```

def ausgabe():
    lbAusgabe[ "text"] = liAuswahl.get("active")

def ende():
    fenster.destroy()

fenster = tkinter.Tk()
fenster.title("Einfach")
fenster.resizable(0, 0)

lbAuswahl = tkinter.Label(fenster, text="Ihre Auswahl:")
lbAuswahl.grid(row=0, column=0, sticky="w", padx=5, pady=5)

frAuswahl = tkinter.Frame(fenster)
frAuswahl.grid(row=1, column=0, padx=5, pady=5)

sbAuswahl = tkinter.Scrollbar(frAuswahl, orient="vertical")
liAuswahl = tkinter.Listbox(frAuswahl, height=4,
    yscrollcommand=sbAuswahl.set)
sbAuswahl[ "command"] = liAuswahl.yview

stadt = [ "Hamburg", "Stuttgart", "Berlin", "Dortmund", "Trier",
    "Duisburg", "Potsdam", "Halle", "Flensburg", "Augsburg"]
for s in stadt:
    liAuswahl.insert("end", s)
liAuswahl.grid(row=0, column=0)
sbAuswahl.grid(row=0, column=1, sticky="sn")

buAusgabe = tkinter.Button(fenster, text="Ausgabe",
    command=ausgabe, width=10)
buAusgabe.grid(row=2, column=0, sticky="w", padx=5, pady=5)

lbAusgabe = tkinter.Label(fenster, text="(leer)")
lbAusgabe.grid(row=3, column=0, sticky="w", padx=5, pady=5)

buEnde = tkinter.Button(fenster, text="Ende", command=ende, width=10)
buEnde.grid(row=4, column=0, padx=5, pady=5)

fenster.mainloop()

```

Listing 11.6 Datei »gui_liste.py«

Nach der Auswahl eines Eintrags und der Betätigung des Buttons AUSGABE ergibt sich die in Abbildung 11.9 gezeigte Darstellung.

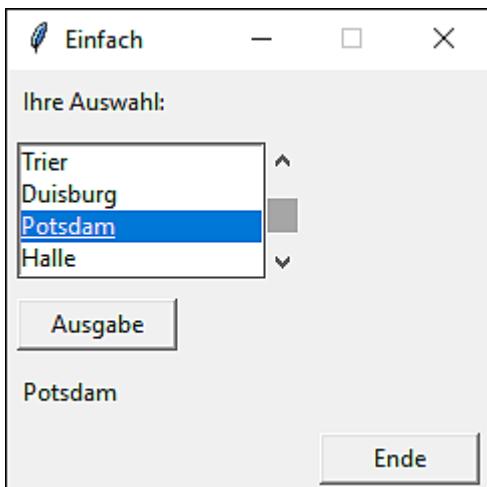


Abbildung 11.9 Liste mit einfacher Auswahl

Es wird ein Frame-Widget erzeugt und der Variablen `frAuswahl` zugewiesen. Es ist nicht sichtbar und dient nur zur Anordnung anderer Widgets.

Das Scrollbar-Widget wird dem Frame-Widget zugeordnet und der Variablen `sbAuswahl` zugewiesen. Mit dem Wert `vertical` für den Parameter `orient` handelt es sich um einen vertikalen Scrollbalken. Eine Scrollbar kann unterschiedlichen Widget-Typen zugeordnet werden.

Das Listbox-Widget wird ebenfalls dem Frame-Widget zugeordnet und der Variablen `liAuswahl` zugewiesen. Die Listbox hat eine Höhe von 4.

Es sind zwei Maßnahmen notwendig, um die Verbindung zwischen Listbox und Scrollbalken herzustellen:

1. Der Parameter `yscrollcommand` der Listbox erhält den Wert `sbAuswahl.set`.
2. Der Eigenschaft `command` der Scrollbar wird der Wert `liAuswahl.yview` zugewiesen.

Anschließend gilt: Wird der Scrollbalken betätigt, scrollt die Listbox.

Die Methode `insert()` fügt einem Listbox-Widget ein Element hinzu. Mit dem Wert »end« wird eine Einfügeposition am Ende

der Liste gewählt. Die Listbox wird mit den zehn Elementen der Liste `stadt` gefüllt.

Für das Listbox-Widget und das Scrollbar-Widget wird die Methode `grid()` aufgerufen. Die Angaben für die Parameter `row` und `column` beziehen sich auf das umgebende Element, also das Frame-Widget. Mit dem Wert »sn« für den Parameter `sticky` wird erreicht, dass sich das Scrollbar-Widget zum oberen und zum unteren Rand des Frame-Widgets erstreckt, also über die gesamte Höhe der Listbox, die wiederum die Höhe des Frame-Widgets bestimmt.

Das Frame-Widget bestimmt die Breite der linken Spalte. Die beiden Label und der Button werden mithilfe des Parameters `sticky` am linken Rand ihrer Zelle angeordnet.

In der Funktion `ausgabe()` wird die Methode `get()` aufgerufen. Sie liefert das Listenelement zur angegebenen Nummer. Mit der Angabe »active« wird das vom Benutzer ausgewählte Element geliefert.

Hinweis

Sie können einer Listbox mithilfe einer Zahl für die Einfügeposition Einträge an beliebiger Stelle einfügen. Die Nummerierung beginnt bei 0. Ein Beispiel: Würden Sie bei allen Aufrufen der Methode `insert()` den Wert 0 angeben, so würde jedes Element der Liste zu Beginn der Listbox eingefügt werden. Die Einträge würden also in umgekehrter Reihenfolge erscheinen.

11.2.5 Listbox mit mehrfacher Auswahl

In einem Listbox-Widget können auch mehrere Elemente ausgewählt werden. Im nachfolgenden Programmausschnitt werden nur die Unterschiede zum vorherigen Programm gezeigt:

```

...
def ausgabe():
    ausgabe = ""
    for index in liAuswahl.curselection():
        ausgabe += f"[liAuswahl.get(index)}\n"
    lbAusgabe[ "text" ] = ausgabe
...
liAuswahl = tkinter.Listbox(frAuswahl,
    height=4, yscrollcommand=sbAuswahl.set, selectmode="multiple")
...
lbAusgabe = tkinter.Label(fenster, text="(leer)", height=10, anchor="n")
lbAusgabe.grid(row=3, column=0, sticky="w", padx=5, pady=5)
...

```

Listing 11.7 Datei »gui_liste_mehrfach.py«

Nach der Auswahl mehrerer Einträge und der Betätigung des Buttons AUSGABE ergibt sich die in [Abbildung 11.10](#) gezeigte Darstellung.

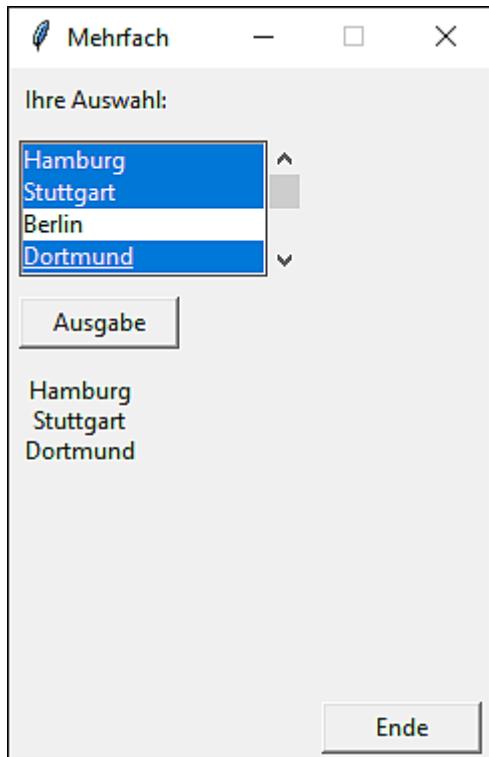


Abbildung 11.10 Liste mit mehrfacher Auswahl

Die Einträge werden zur Auswahl nacheinander angeklickt, ohne die gewohnte Betätigung der Sondertasten **Strg** oder **←**. Ein erneuter Klick auf einen ausgewählten Eintrag macht dessen Auswahl wieder rückgängig.

Die Listbox wird zusätzlich mit der Eigenschaft `selectmode` und dem Eigenschaftswert `multiple` versehen. Der Standardwert ist `single` und muss daher beim vorherigen Beispiel nicht angegeben werden.

Die Methode `cursellection()` liefert die Indizes der vom Benutzer ausgewählten Einträge in einem Tupel zurück. Die zugehörigen Einträge aus der Listbox werden ausgegeben.

Das Label-Widget hat eine Höhe, die zur Ausgabe aller Elemente ausreichen würde. Mithilfe des Parameters `anchor` wird der Text am oberen Rand des Label-Widgets angeordnet.

11.2.6 Spinbox

Seit Python 3.7 gibt es die Klasse `tkinter.ttk.Spinbox` zur Erzeugung eines Spinbox-Widgets. Es entspricht einer Combobox, also einem Kombinationsfeld, wie es von anderen Benutzeroberflächen bekannt ist.

In einer Spinbox können Sie:

- einen Wert eintragen wie in einem Entry-Widget
- einen Wert aus einem Zahlenbereich auswählen
- einen Eintrag aus einer Liste auswählen

Nachfolgend sehen Sie ein Programm, in dem die genannten Möglichkeiten umgesetzt werden:

```
import tkinter.ttk

def auswahlStadt():
    lbAusgabe[ "text"] = "Auswahl: " + spStadt.get()

def auswahlZahl():
    lbAusgabe[ "text"] = "Auswahl: " + spZahl.get()

def ausgabe():
    lbAusgabe[ "text"] = f"Ausgabe: {spStadt.get()}/{spZahl.get()}"

def ende():
    fenster.destroy()
```

```

fenster = tkinter.Tk()
fenster.title("Spinbox")
fenster.resizable(0, 0)

lbAuswahl = tkinter.Label(fenster, text="Ihre Eingabe oder Auswahl:")
lbAuswahl.grid(row=0, column=0, padx=5, pady=5)

stadt = [ "Hamburg", "Stuttgart", "Berlin", "Dortmund", "Trier",
          "Duisburg", "Potsdam", "Halle", "Flensburg", "Augsburg"]
spStadt = tkinter.ttk.Spinbox(fenster, values=stadt,
                               command=auswahlStadt)
spStadt.set("Hamburg")
spStadt.grid(row=1, column=0, sticky="w", padx=5, pady=5)

spZahl = tkinter.ttk.Spinbox(fenster,
                            from_=10, to=20, width=15, command=auswahlZahl)
spZahl.set(12)
spZahl.grid(row=2, column=0, sticky="w", padx=5, pady=5)

buAusgabe = tkinter.Button(fenster, text="Ausgabe",
                           command=ausgabe, width=10)
buAusgabe.grid(row=3, column=0, sticky="w", padx=5, pady=5)

lbAusgabe = tkinter.Label(fenster, text="(leer)")
lbAusgabe.grid(row=4, column=0, sticky="w", padx=5, pady=5)

buEnde = tkinter.Button(fenster, text="Ende", command=ende, width=10)
buEnde.grid(row=5, column=0, sticky="w", padx=5, pady=5)

fenster.mainloop()

```

Listing 11.8 Datei »gui_spinbox.py«

Nach der Auswahl eines Eintrags in der oberen Spinbox sieht es so aus wie in [Abbildung 11.11](#).

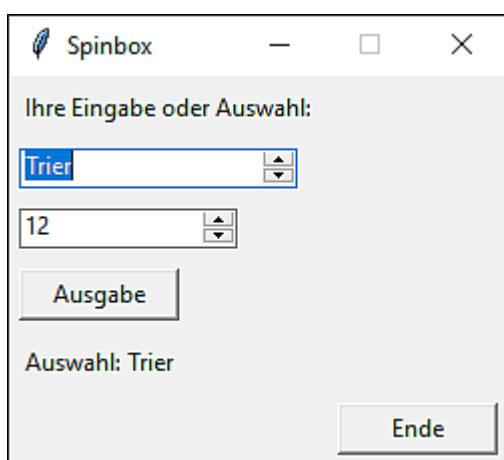


Abbildung 11.11 Auswahl eines Eintrags in der oberen Spinbox

Nach der Auswahl eines Eintrags in der unteren Spinbox sieht es so aus wie in [Abbildung 11.12](#).

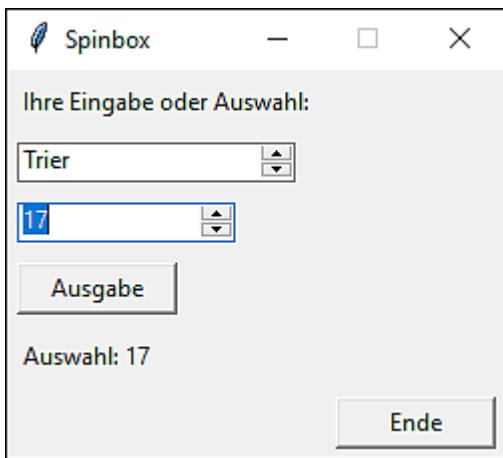


Abbildung 11.12 Auswahl eines Eintrags in der unteren Spinbox

Nach der Eingabe einer Zeichenkette in der oberen Spinbox und einer Zahl in der unteren Spinbox sowie der anschließenden Betätigung des Buttons **AUSGABE** sieht es aus wie in [Abbildung 11.13.](#)

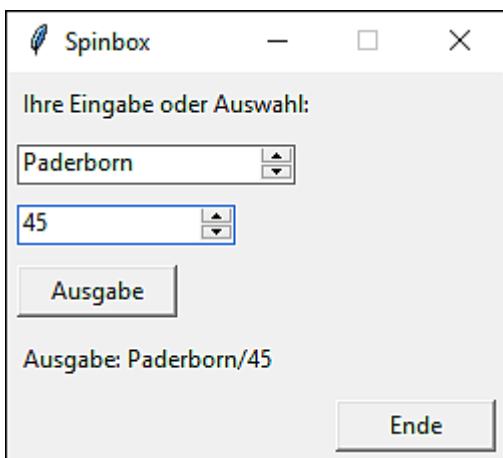


Abbildung 11.13 Eingabe von neuen Einträgen

In der oberen Spinbox `spStadt` werden die Elemente der Liste `stadt` angezeigt. Sie wurden der Spinbox mithilfe des Parameters `values` hinzugefügt. In der unteren Spinbox werden die Zahlen von 10 bis 20 angezeigt. Sie wurden der Spinbox mithilfe der beiden Parameter `from_` und `to` hinzugefügt. Die Methode `set()` dient zum Setzen des Startwerts einer Spinbox.

Eine Spinbox kann mit einer Callback-Funktion verbunden werden. Diese Funktion wird aufgerufen, sobald der Benutzer eine Auswahl in der Spinbox vornimmt. Diese Auswahl wird

mithilfe der Methode `get()` ermittelt. Die Callback-Funktion wird allerdings nicht nach einer Eingabe in der Spinbox aufgerufen.

Nach Betätigung des Buttons `AUSGABE` werden die aktuellen Werte der beiden Spinboxen angezeigt. Das gilt sowohl für ausgewählte als auch für eingegebene Werte.

11.2.7 Radiobutton, Widget-Variable

Radiobuttons ermöglichen eine einfache Auswahl durch den Benutzer, ähnlich wie eine Listbox mit einfacher Auswahl. Radiobuttons sind einzelne Widgets, die zu einer Gruppe zusammengefügt werden, damit sie gemeinsam reagieren. Dazu müssen sie mit einer Widget-Variablen verbunden werden.

Hinweis

Sie können mehrere Widget-Typen mit Widget-Variablen verbinden. Eine Änderung des Widget-Zustands führt zu einer Änderung des Werts der Widget-Variablen und umgekehrt.

Im folgenden Beispiel kann der Benutzer mithilfe von drei Radiobuttons eine Farbe auswählen:

```
import tkinter

def auswahl():
    lbAusgabe[ "text" ] = "Auswahl: " + farbe.get()

def ausgabe():
    lbAusgabe[ "text" ] = "Ausgabe: " + farbe.get()

def ende():
    fenster.destroy()

fenster = tkinter.Tk()
fenster.title("Radiobuttons")
fenster.resizable(0, 0)

lbAuswahl = tkinter.Label(fenster, text="Farbe:", anchor="w", width=20)
lbAuswahl.grid(row=0, column=0, sticky="w", padx=5, pady=5)

farbe = tkinter.StringVar()
farbe.set("Rot")

frFarbe = tkinter.Frame(fenster)
```

```

frFarbe.grid(row=1, column=0, padx=5, pady=5)

rbRot = tkinter.Radiobutton(frFarbe, text="Rot", variable=farbe,
    value="Rot", command=auswahl)
rbRot.grid(row=0, column=0, padx=5, pady=5)

rbGelb = tkinter.Radiobutton(frFarbe, text="Gelb", variable=farbe,
    value="Gelb", command=auswahl)
rbGelb.grid(row=0, column=1, padx=5, pady=5)

rbBlau = tkinter.Radiobutton(frFarbe, text="Blau", variable=farbe,
    value="Blau", command=auswahl)
rbBlau.grid(row=0, column=2, padx=5, pady=5)

buAusgabe = tkinter.Button(fenster, text="Ausgabe",
    command=ausgabe, width=10)
buAusgabe.grid(row=2, column=0, sticky="w", padx=5, pady=5)

lbAusgabe = tkinter.Label(fenster, text="(leer)")
lbAusgabe.grid(row=3, column=0, sticky="w", padx=5, pady=5)

buEnde = tkinter.Button(fenster, text="Ende", command=ende, width=10)
buEnde.grid(row=4, column=1, padx=5, pady=5)

fenster.mainloop()

```

Listing 11.9 Datei »gui_radio.py«

Nach der Auswahl der Option GELB sieht es aus wie in [Abbildung 11.14](#).

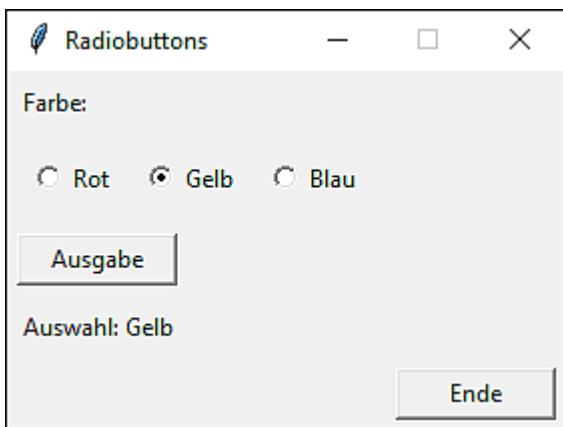


Abbildung 11.14 Gruppe von Radiobuttons

Eine Widget-Variable muss ein Objekt aus einer der folgenden Klassen sein:

- StringVar (für Zeichenketten)
- IntVar (für ganze Zahlen)
- DoubleVar (für Zahlen mit Nachkommastellen)

Hier wird die Widget-Variable `farbe` als Objekt der Klasse `StringVar` erzeugt. Mithilfe der Methode `set()` wird ihr der Startwert `Rot` zugewiesen.

Die drei Radiobuttons werden nebeneinander in ein Frame-Widget eingebettet. Sie besitzen den Wert `farbe` für die Eigenschaft `variable`. Damit sind sie miteinander und mit der Widget-Variablen `farbe` verbunden. Wählt der Benutzer einen Radiobutton aus, wird der Widget-Variablen der zugehörige Wert der Eigenschaft `value` des Radiobuttons zugewiesen.

Die Widget-Variable `farbe` besitzt den Startwert `Rot`. Damit ist der zugehörige Radiobutton zu Beginn des Programms bereits ausgewählt. Eine solche Vorbelegung gehört zum guten Programmierstil und verhindert, dass der Benutzer gar keine Auswahl trifft.

Die Funktion `auswahl()` wird aufgerufen, sobald der Benutzer einen der Radiobuttons auswählt. Die Funktion `ausgabe()` wird nach der Betätigung des Buttons `AUSGABE` aufgerufen. In beiden Funktionen wird mithilfe der Methode `get()` der Wert der Widget-Variablen und damit die vom Benutzer getroffene Auswahl ermittelt.

Meist wird in einer Anwendung nur eine der beiden Funktionen benötigt: Entweder möchten Sie, dass die Benutzerin durch die Auswahl eines Radiobuttons unmittelbar eine weitere Aktion auslöst, oder Sie möchten den Zustand der Gruppe von Radiobuttons erst nach der Bedienung aller Elemente einer Anwendung erfahren.

11.2.8 Checkbutton

Checkbuttons ermöglichen eine mehrfache Auswahl durch den Benutzer, ähnlich wie eine Listbox mit mehrfacher Auswahl. Den Zustand eines Checkbuttons können Sie, wie den Zustand einer

Gruppe von Radiobuttons, entweder unmittelbar oder erst nach der Bedienung aller Elemente erfahren.

Nachfolgend wird eine Reservierung für ein Hotelzimmer durchgeführt. Mithilfe von zwei Checkbuttons entscheidet der Benutzer, ob er ein Zimmer mit oder ohne Dusche sowie mit oder ohne Minibar möchte:

```
import tkinter

def auswahl():
    lbAusgabe[ "text"] = f"Auswahl: {dusche.get()}, {minibar.get() }"

def ausgabe():
    lbAusgabe[ "text"] = f"Ausgabe: {dusche.get()}, {minibar.get() }"

def ende():
    fenster.destroy()

fenster = tkinter.Tk()
fenster.title("Checkbuttons")
fenster.resizable(0, 0)

lbAuswahl = tkinter.Label(fenster, text="Zimmer:", anchor="w", width=30)
lbAuswahl.grid(row=0, column=0, padx=5, pady=5)

dusche = tkinter.StringVar()
dusche.set("ohne Dusche")

minibar = tkinter.StringVar()
minibar.set("ohne Minibar")

cbDusche = tkinter.Checkbutton(fenster, text="Dusche", variable=dusche,
    onvalue="mit Dusche", offvalue="ohne Dusche", command=auswahl)
cbDusche.grid(row=1, column=0, sticky="w", padx=5, pady=5)

cbMinibar = tkinter.Checkbutton(fenster, text="Minibar", variable=minibar,
    onvalue="mit Minibar", offvalue="ohne Minibar", command=auswahl)
cbMinibar.grid(row=2, column=0, sticky="w", padx=5, pady=5)

buAusgabe = tkinter.Button(fenster, text="Ausgabe",
    command=ausgabe, width=10)
buAusgabe.grid(row=3, column=0, sticky="w", padx=5, pady=5)

lbAusgabe = tkinter.Label(fenster, text="(leer)")
lbAusgabe.grid(row=4, column=0, sticky="w", padx=5, pady=5)

buEnde = tkinter.Button(fenster, text="Ende", command=ende, width=10)
buEnde.grid(row=5, column=0, padx=5, pady=5)

fenster.mainloop()
```

Listing 11.10 Datei »gui_check.py«

Nachdem der Benutzer beide Optionen ausgewählt hat, sieht es so aus wie in [Abbildung 11.15](#).

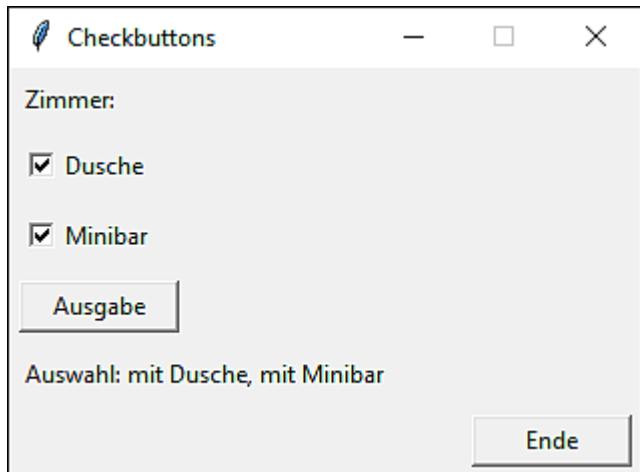


Abbildung 11.15 Nach der Auswahl beider Optionen

Die beiden Widget-Variablen `dusche` und `minibar` erhalten mithilfe der Methode `set()` die Startwerte »ohne Dusche« und »ohne Minibar«.

Die beiden Checkbuttons sind jeweils über die Eigenschaft `variable` mit der zugehörigen Variablen verbunden. Die beiden Eigenschaften `onvalue` und `offvalue` repräsentieren die beiden Zustände »Checkbutton ist markiert« bzw. »Checkbutton ist nicht markiert«.

Die Funktion `auswahl()` wird aufgerufen, sobald der Benutzer die Markierung eines Checkbuttons setzt oder entfernt. Die Funktion `ausgabe()` wird nach der Betätigung des Buttons `AUSGABE` aufgerufen. In beiden Funktionen wird mithilfe der Methode `get()` der Wert der beiden Widget-Variablen und damit die vom Benutzer getroffene Auswahl ermittelt.

11.2.9 Schieberegler, Scale

Ein Scale-Widget entspricht einem Schieberegler, also der visuellen Anzeige eines Werts, der mithilfe der Maus verändert werden kann.

Im folgenden Beispiel wird ein Geschwindigkeitsmesser simuliert. Es können Geschwindigkeiten zwischen 0 und 200 km/h in Schritten von 5 km/h angezeigt und eingestellt werden. Der eingestellte Wert wird zusätzlich in einem Label angezeigt:

```
import tkinter

def aktion(self):
    lbWert[ "text"] = f"Aktion: {scWert.get()} km/h"

def ausgabe():
    lbWert[ "text"] = f"Ausgabe: {scWert.get()} km/h"

def ende():
    fenster.destroy()

fenster = tkinter.Tk()
fenster.title("Scale")
fenster.resizable(0, 0)

wert = tkinter.IntVar()
wert.set(100)

scWert = tkinter.Scale(fenster, length=300, orient="horizontal",
    from_=0, to=200, resolution=5, tickinterval=20, label="km/h",
    command=aktion, variable=wert)
scWert.grid(row=0, column=0, padx=5, pady=5)

buAusgabe = tkinter.Button(fenster, text="Ausgabe", command=ausgabe, width=10)
buAusgabe.grid(row=1, column=0, sticky="w", padx=5, pady=5)

lbWert = tkinter.Label(fenster, text="Start: 100 km/h")
lbWert.grid(row=2, column=0, sticky="w", padx=5, pady=5)

buEnde = tkinter.Button(fenster, text="Ende", command=ende, width=10)
buEnde.grid(row=3, column=1, padx=5, pady=5)

fenster.mainloop()
```

Listing 11.11 Datei »gui_scale.py«

Nachdem die Benutzerin den Schieberegler betätigt hat, sieht es so aus wie in [Abbildung 11.16](#).

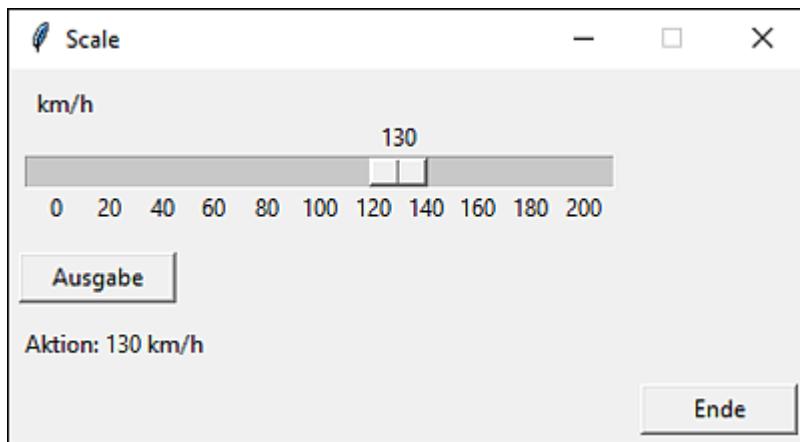


Abbildung 11.16 Nach dem Schieben auf 130

Die Widget-Variable `wert` wird mit dem Wert 100 vorbelegt.

Das Scale-Widget wird mithilfe der Eigenschaft `orient` horizontal ausgerichtet, besitzt dank der Eigenschaft `label` ein Bezeichnungsfeld mit dem Wert »km/h« und erhält über die Eigenschaft `length` eine Länge von 300. Es zeigt dank der Eigenschaften `from_` und `to` Werte von 0 bis 200 an.

Der Wert für die Eigenschaft `resolution` (deutsch: Auflösung) kennzeichnet die Schrittweite beim Schieben. Mithilfe der Eigenschaft `tickinterval` wird der Abstand der angezeigten Werte festgelegt. Über die Eigenschaft `variable` ist der Schieberegler mit der Widget-Variablen `wert` verbunden.

Beim Betätigen des Schiebereglers wird die Funktion `aktion()` aufgerufen, nach der Betätigung des Buttons `AUSGABE` die Funktion `ausgabe()`. In beiden Funktionen wird mithilfe der Methode `get()` der aktuelle Wert des Schiebereglers ermittelt und angezeigt.

11.3 Bilder und Mausereignisse

In diesem Abschnitt erläutere ich, wie Bilder im Anwendungsfenster eingebettet und unterschiedliche Mausereignisse verarbeitet werden.

11.3.1 Bild einbetten und ändern

Auf einem Label kann der Inhalt einer Bilddatei dargestellt werden. Dazu wird ein Objekt der Klasse `PhotoImage` benötigt. Die Eigenschaften eines solchen Objekts können ermittelt und verändert werden.

Im nachfolgenden Beispiel werden die Breite und die Höhe eines Bilds ermittelt. Außerdem werden die Farbe und die Transparenz von einzelnen Bildpunkten ermittelt und geändert:

```
import tkinter

def schrift():
    for x in range(5, 94):
        for y in range(69, 80):
            imBild.put("#00549D", (x, y))

def transparent():
    for x in range(5, 94):
        for y in range(38, 46):
            imBild.transparency_set(x, y, True)
    if imBild.transparency_get(50, 40):
        print("Punkt ist transparent")

def ende():
    fenster.destroy()

fenster = tkinter.Tk()
fenster.title("Bild")
fenster.resizable(0, 0)

imBild = tkinter.PhotoImage(file="rheinwerk.png")
lbBild = tkinter.Label(fenster)
lbBild[ "image"] = imBild
lbBild.grid(row=0, column=0, padx=5, pady=5)

print("Breite:", imBild.width())
print("Höhe:", imBild.height())
```

```

print("Farbe x,y:", imBild.get(0, 0))
if not imBild.transparency_get(50, 40):
    print("Punkt ist nicht transparent")

buSchrift = tkinter.Button(fenster,
    text="Schrift blau", command=schrift, width=20)
buSchrift.grid(row=1, column=0, padx=5, pady=5)

buTransparent = tkinter.Button(fenster,
    text="Bereich transparent", command=transparent, width=20)
buTransparent.grid(row=2, column=0, padx=5, pady=5)

buEnde = tkinter.Button(fenster, text="Ende", command=ende, width=10)
buEnde.grid(row=3, column=0, padx=5, pady=5)

fenster.mainloop()

```

Listing 11.12 Datei »gui_bild.py«

Nach dem Aufruf des Programms sieht es aus wie in [Abbildung 11.17](#).



Abbildung 11.17 Nach dem Aufruf des Programms

Nach der Betätigung des Buttons **SCHRIFT IN BLAU** wird der Schriftzug gelöscht, indem die Bildpunkte in diesem Bereich in Blau umgefärbt werden, siehe [Abbildung 11.18](#).



Abbildung 11.18 Nach dem Löschen des Schriftzugs

Nach der Betätigung des Buttons BEREICH TRANSPARENT werden die Bildpunkte in einem Bereich transparent, siehe [Abbildung 11.19](#).

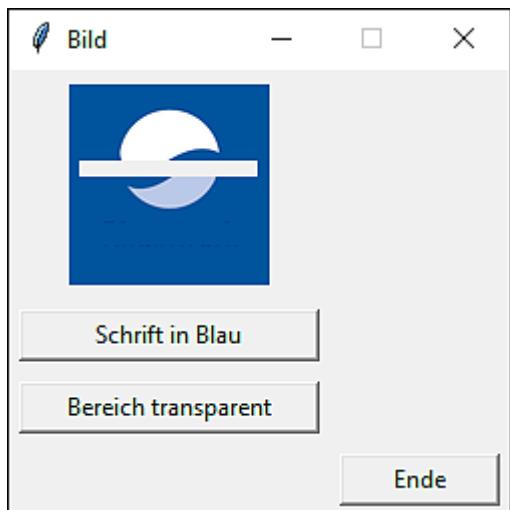


Abbildung 11.19 Nach dem Ändern der Transparenz

Zusätzlich erscheinen in der IDLE SHELL einige Informationen:

```
Breite: 100
Höhe: 100
Farbe x,y: (0, 84, 157)
Punkt ist nicht transparent
```

Das Bildobjekt wird als Objekt der Klasse `PhotoImage` erstellt. Der Parameter `file` enthält den Namen der Bilddatei, deren Inhalt dargestellt wird. Die Darstellung erfolgt in einem Label. Dessen Eigenschaft `image` wird die Referenz auf das Bildobjekt zugewiesen.

Die Methoden `width()` und `height()` liefern die Breite und die Höhe des Bilds in Pixeln, die Methode `get()` die Farbe des angegebenen Bildpunkts als Tupel aus drei Werten zwischen 0 und 255 für die Farbanteile Rot, Grün und Blau. Als Parameter werden die beiden Koordinaten (`x` und `y`) des Bildpunkts übergeben. Die Koordinaten 0, 0 befinden sich oben links.

Seit Python 3.8 gibt es die Methode `transparency_get()`. Sie liefert die Information, ob der betreffende Bildpunkt transparent ist.

Nach der Betätigung des Buttons **SCHRIFT IN BLAU** wird in der Funktion `schrift()` für alle Bildpunkte eines Bereichs die Methode `put()` aufgerufen. Sie dient zum Setzen der Farbe des Bildpunkts. Als erster Parameter wird die neue Farbe mithilfe einer HTML-Farbangabe im Format `#RRGGBB` notiert.

Dabei folgen nach dem Zeichen `#` jeweils zwei hexadezimale Ziffern für die Farbanteile Rot, Grün und Blau. Als zweiter Parameter folgt ein Tupel mit den beiden Koordinaten für `x` und `y`. Hexadezimale Ziffern sind (in aufsteigender Reihenfolge): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A (entspricht dem Dezimalwert 10), B (= 11), C (= 12), D (= 13), E (= 14), F (= 15), siehe auch [Abschnitt 4.1.1](#), »Ganze Zahlen«.

Im vorliegenden Beispiel wird mithilfe einer doppelten Schleife ein breiter Streifen von Bildpunkten in der Farbe `#00549D` eingefärbt, dem Standard-Dunkelblau des Rheinwerk-Logos. Der Wert entspricht in dezimaler Form der Angabe 0, 84, 157. Anschließend ist der Schriftzug »Rheinwerk« nicht mehr sichtbar.

Nach der Betätigung des Buttons **BEREICH TRANSPARENT** wird in der Funktion `transparent()` die Methode `transparency_set()` aufgerufen, die es ebenfalls seit Python 3.8 gibt. Sie dient zum Setzen der Transparenz eines Bildpunkts. Die ersten beiden Parameter kennzeichnen die beiden Koordinaten für `x` und `y`. Der dritte Parameter gibt an, ob der Punkt transparent sein soll (`True`)

oder nicht (`False`). Im vorliegenden Beispiel wird wiederum ein breiter Streifen von Bildpunkten, der mitten im Bild liegt, transparent gemacht. Für einen Bildpunkt in diesem Streifen wird die entsprechende Information ausgegeben.

11.3.2 Mausereignisse

Die Widgets, die in den bisherigen Programmen zu Aktionen geführt haben, enthalten die Eigenschaft `command`. Über diese Eigenschaft wird das Widget mit einer Funktion verbunden, die nach dem meistgenutzten Ereignis des betreffenden Widgets aufgerufen wird. Häufig handelt es sich dabei um das Klickereignis.

Sie können Widgets auch mit anderen Mausereignissen verbinden. In der nachfolgenden Liste sehen Sie einige dieser Ereignisse. In Klammern dahinter steht der Name des Ereignisses, wie er für Ihr Programm benötigt wird:

- Herunterdrücken der linken oder rechten Maustaste (`<Button1>` bzw. `<Button3>`)
- Loslassen der linken oder rechten Maustaste (`<ButtonRelease 1>` bzw. `<ButtonRelease 3>`)
- Bewegung der Maus in ein Widget hinein (`<Enter>`)
- Bewegung der Maus innerhalb eines Widgets (Ereignis `<Motion>`)
- Bewegung der Maus aus einem Widget heraus (`<Leave>`)

Nachfolgend werden die genannten Möglichkeiten vorgeführt:

```
import tkinter

def bewegt(e):
    lbAusgabe[ "text"] = f"x={e.x}, y={e.y}"

def links(e):
    lbAusgabe[ "text"] = "Links gedrückt"
```

```

def linkslos(e):
    lbAusgabe[ "text"] = "Links losgelassen"

def rechts(e):
    lbAusgabe[ "text"] = "Rechts gedrückt"

def rechtslos(e):
    lbAusgabe[ "text"] = "Rechts losgelassen"

def verlassen(e):
    lbAusgabe[ "text"] = "Verlassen"

def betreten(e):
    lbAusgabe[ "text"] = "Betreten"

def ende():
    fenster.destroy()

fenster = tkinter.Tk()
fenster.title("Maus")
fenster.resizable(0, 0)

imBild = tkinter.PhotoImage(file="rheinwerk.png")
lbBild = tkinter.Label(fenster, image=imBild)
lbBild.bind("<Motion>", bewegt)
lbBild.bind("<Button 1>", links)
lbBild.bind("<ButtonRelease 1>", linkslos)
lbBild.bind("<Button 3>", rechts)
lbBild.bind("<ButtonRelease 3>", rechtslos)
lbBild.bind("<Leave>", verlassen)
lbBild.grid(row=0, column=0, padx=5, pady=5)

lbAusgabe = tkinter.Label(fenster, text="(leer)")
lbAusgabe.grid(row=1, column=0, sticky="w", padx=5, pady=5)

buEnde = tkinter.Button(fenster, text="Ende", command=ende, width=10)
buEnde.grid(row=2, column=0, padx=5, pady=5)
buEnde.bind("<Enter>", betreten)

fenster.mainloop()

```

Listing 11.13 Datei »gui_maus.py«

In Abbildung 11.20 befindet sich die Maus im oberen linken Bereich des Labels mit dem Bild.

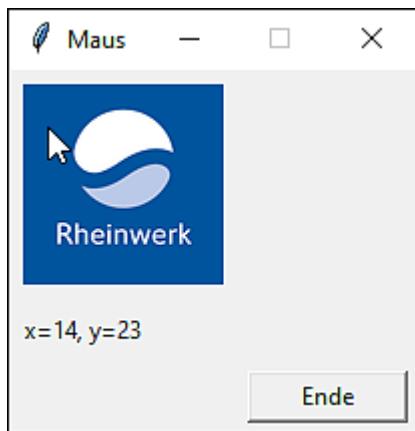


Abbildung 11.20 Maus im Bereich links oben

Wird die rechte Maustaste im Widget losgelassen, sieht es aus wie in [Abbildung 11.21](#).

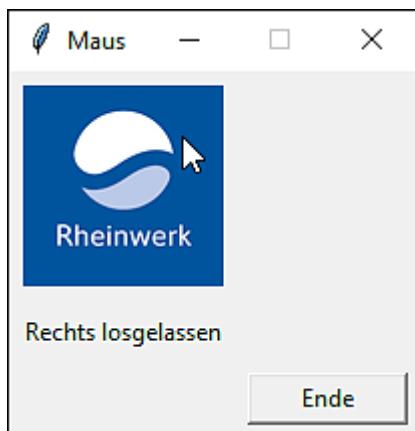


Abbildung 11.21 Rechte Maustaste losgelassen

Es werden insgesamt sieben Funktionen definiert, die mit bestimmten Ereignissen und Widgets verbunden sind. Hier ist das Ereignis »Enter« mit dem Button ENDE verbunden, die übrigen Ereignisse mit dem Label, in dem ein Bild dargestellt wird. In einem weiteren Label werden Informationen zu dem jeweiligen Ereignis angezeigt.

Zur Verbindung dient die Methode `bind()`. Der erste Parameter der Methode ist eine Zeichenkette mit dem Namen des Ereignisses. Der zweite Parameter ist der Name der Funktion, die aufgerufen wird, wenn das Ereignis eintritt.

Jede Ereignisfunktion übermittelt Informationen über das Ereignis mithilfe eines Event-Objekts (hier: `e`). Bei einem

Mausereignisse sind dabei die Objekteigenschaften `x` und `y` interessant. Sie liefern die aktuellen Koordinaten des Mauszeigers innerhalb des Widgets.

Unterschiede unter macOS

Unter macOS wird die rechte Maustaste anders abgefangen.

11.4 Geometrie-Manager »place«

Der Geometrie-Manager *place* ermöglicht die Anordnung von Widgets über Koordinaten. Die Größe des Anwendungsfensters richtet sich in diesem Fall nicht nach den enthaltenen Widgets und muss daher eingestellt werden.

11.4.1 Fenstergröße und absolute Position

Im nachfolgenden Beispiel wird die GUI aus [Abschnitt 11.1.2, »Anordnung von Widgets«](#), mithilfe des Geometrie-Managers *place* erstellt:

```
import tkinter

def hallo():
    lbAusgabe[ "text"] = "Hallo"

def ende():
    fenster.destroy()

fenster = tkinter.Tk()
fenster.title("GUI")
fenster.geometry("245x105+200+50")
fenster.resizable(0, 0)

buHallo = tkinter.Button(fenster, text="Hallo", command=hallo, width=10)
buHallo.place(x=5, y=5)

lbAusgabe = tkinter.Label(fenster, text="(leer)",
                           anchor="w", relief="sunken", width=20)

buEnde = tkinter.Button(fenster, text="Ende", command=ende, width=10)
buEnde.place(x=160, y=75)

fenster.mainloop()
```

Listing 11.14 Datei »gui_place.py«

Die Anwendung sieht nach der Betätigung des Buttons HALLO so aus wie in [Abbildung 11.22](#).

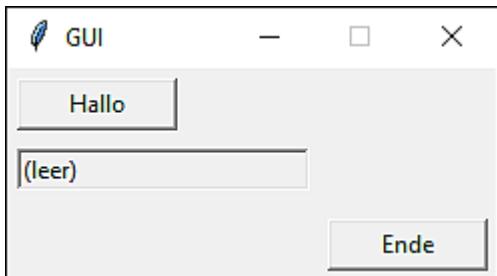


Abbildung 11.22 Fenstergröße und absolute Position

Mithilfe der Methode `geometry()` können Sie die Größe und die Position des Anwendungsfensters festlegen. Die ersten beiden Zahlen werden mit dem Zeichen »x« verbunden und geben die Breite und die Höhe des Fensters an. Die nächsten beiden Zahlen folgen jeweils nach dem Zeichen »+« und geben den Abstand der linken oberen Ecke des Fensters zur linken oberen Ecke des Bildschirms an. Geben Sie nur die Größe an, zum Beispiel mit `fenster.geometry("245x105")`, ist der Ort beliebig.

Beim Aufruf der Methode `place()` müssen Werte für die benannten Parameter `x` und `y` übergeben werden. Sie bestimmen den Abstand der linken oberen Ecke des Widgets zur linken oberen Ecke des Bereichs des Anwendungsfensters unterhalb der Titelzeile.

Da die Größe des Anwendungsfensters sich nicht nach den enthaltenen Widgets richtet, werden seine endgültige Größe und die Position der Widgets meist erst nach mehrfachem Ausprobieren erreicht.

11.4.2 Relative Position

Mit dem Geometrie-Manager `place` kann die Position eines Widgets auch in Relation zur Größe des Anwendungsfensters gewählt werden. Das hat Vorteile, falls diese verändert werden kann.

Im nachfolgenden Beispiel werden insgesamt neun Buttons mit den Aufschriften 1 bis 9 in den Ecken oder in der Mitte der Seiten

des Anwendungsfensters positioniert. Die Buttons sind nicht mit einer Aktion verbunden und dienen nur als Beispiele für beliebige Widgets. Das Fenster kann innerhalb definierter Werte vergrößert oder verkleinert werden:

```
import tkinter

fenster = tkinter.Tk()
fenster.title("GUI")
fenster.geometry("300x225")
fenster.minsize(200, 150)
fenster.maxsize(400, 300)

bu1 = tkinter.Button(fenster, text="1", width=5)
bu1.place(relx=0.05, rely=0.05)

bu2 = tkinter.Button(fenster, text="2", width=5)
bu2.place(relx=0.5, rely=0.05, anchor="n")

bu3 = tkinter.Button(fenster, text="3", width=5)
bu3.place(relx=0.95, rely=0.05, anchor="ne")

bu4 = tkinter.Button(fenster, text="4", width=5)
bu4.place(relx=0.05, rely=0.5, anchor="w")

bu5 = tkinter.Button(fenster, text="5", width=5)
bu5.place(relx=0.5, rely=0.5, anchor="center")

bu6 = tkinter.Button(fenster, text="6", width=5)
bu6.place(relx=0.95, rely=0.5, anchor="e")

bu7 = tkinter.Button(fenster, text="7", width=5)
bu7.place(relx=0.05, rely=0.95, anchor="sw")

bu8 = tkinter.Button(fenster, text="8", width=5)
bu8.place(relx=0.5, rely=0.95, anchor="s")

bu9 = tkinter.Button(fenster, text="9", width=5)
bu9.place(relx=0.95, rely=0.95, anchor="se")

fenster.mainloop()
```

Listing 11.15 Datei »gui_relativ.py«

Nach dem Start des Programms sieht es aus wie in Abbildung 11.23.

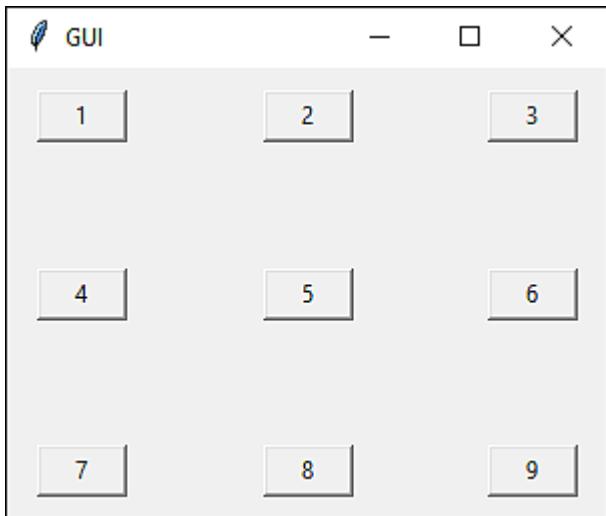


Abbildung 11.23 Relative Position nach dem Start

Das Fenster sieht in der kleinstmöglichen Größe aus wie in [Abbildung 11.24](#).



Abbildung 11.24 Relative Position nach Verkleinerung

Die Methoden `minsize()` und `maxsize()` dienen zur Festlegung der minimalen bzw. maximalen Größe des Anwendungsfensters.

Für die beiden Parameter `relx` und `rely` können Werte zwischen 0 (ganz links bzw. ganz oben) und 1 (ganz rechts bzw. ganz unten) gewählt werden. Ohne Angabe des Parameters `anchor` wird als Bezugspunkt für die relative Position die linke obere Ecke des Widgets gewählt. Weitere Bezugspunkte sind zum Beispiel:

- `n`: Mitte der oberen Seite des Widgets
- `ne`: obere rechte Ecke des Widgets
- `w`: Mitte der linken Seite des Widgets
- `center`: Zentrum des Widgets

- e: Mitte der rechten Seite des Widgets
- sw: untere linke Ecke des Widgets
- s: Mitte der unteren Seite des Widgets
- se: untere rechte Ecke des Widgets

11.4.3 Position ändern

Ein Aufruf eines der Geometrie-Manager zur Laufzeit der Anwendung ermöglicht die Änderung der Position eines Widgets.

Im nachfolgenden Beispiel kann ein Widget mithilfe von zwei Buttons und der Methode `place()` nach links oder rechts verschoben werden:

```
import tkinter, time

posx = 205

def links():
    global posx
    if posx > 80:
        posx -= 20
    buVerschieben.place(x=posx, y=5)

def rechts():
    global posx
    if posx < 340:
        posx += 20
    buVerschieben.place(x=posx, y=5)

fenster = tkinter.Tk()
fenster.title("Verschieben")
fenster.geometry("455x35")
fenster.resizable(0, 0)

buLinks = tkinter.Button(fenster, text="<<", command=links, width=5)
buLinks.place(x=5, y=5)

buVerschieben = tkinter.Button(fenster, text="xxx", width=5)
buVerschieben.place(x=205, y=5)

buRechts = tkinter.Button(fenster, text=">>", command=rechts, width=5)
buRechts.place(x=405, y=5)

fenster.mainloop()
```

Listing 11.16 Datei »gui_verschieben.py«

Nach mehrmaliger Betätigung des Buttons >> sieht es aus wie in Abbildung 11.25.



Abbildung 11.25 Position ändern

Die Variable `posx` dient zur Speicherung der aktuellen Position des Widgets mit der Aufschrift `xxx` in X-Richtung. Zu Beginn wird die Startposition 205 gespeichert.

Die Betätigung der beiden Buttons führt zu den Funktionen `links()` bzw. `rechts()`. Hier wird zunächst die Variable `posx` als global übernommen. Es wird geprüft, ob sich das Widget bereits ganz links bzw. ganz rechts befindet. Ist das nicht der Fall, wird es mithilfe der Methode `place()` auf eine neue Position gesetzt, die sich ein Stück weiter links bzw. rechts von der aktuellen Position befindet.

11.5 Menüs, Messageboxen und Dialogfelder

Viele Benutzeroberflächen verfügen zur Bedienung neben den Widgets innerhalb des Anwendungsfenster über:

- eine Menüleiste, die permanent am oberen Rand sichtbar ist
- Kontextmenüs, die einzelnen Widgets zugeordnet sind und nur bei Bedarf eingeblendet werden

In Python stehen zu diesem Zweck `Menu`-Widgets zur Verfügung. In diesem Abschnitt stelle ich die beiden genannten Menüarten, vorgefertigte Dialogfelder (Messageboxen) und eigene Dialogfelder vor.

11.5.1 Menüleisten

Die Erzeugung einer Menüleiste mit einzelnen Menüs und darin enthaltenen Menübefehlen erfordert grundsätzlich die folgenden Schritte:

- Erzeugung eines Objekts der Klasse `Menu` als Menüleiste
- Erzeugung weiterer Objekte der Klasse `Menu` als einzelne Menüs innerhalb der Menüleiste
- Erzeugung der Menüpunkte innerhalb eines Menüs
- Hinzufügen der Menüs zur Menüleiste und Bezeichnen der Menüs
- Hinzufügen der Menüleiste zum Fenster

Das folgende Programm enthält eine Menüleiste mit zwei Menüs, die jeweils mehrere Menüpunkte unterschiedlichen Typs enthalten:

```

import tkinter

def farbwechsel():
    lbHallo[ "bg"] = farbe.get()

def randwechsel():
    lbHallo[ "relief"] = rand.get()

def ende():
    fenster.destroy()

fenster = tkinter.Tk()
fenster.title("Menü")
fenster.resizable(0, 0)

lbHallo = tkinter.Label(fenster, text="Hallo", width=30, bg="#FAAC58")
lbHallo.grid(row=0, column=0, padx=20, pady=40)

mLeiste = tkinter.Menu(fenster)

mDatei = tkinter.Menu(mLeiste)
mDatei.add_command(label="Neu")
mDatei.add_command(label="Öffnen")
mDatei.add_separator()
mDatei.add_command(label="Beenden", command=ende)

mAnsicht = tkinter.Menu(mLeiste)
mAnsicht[ "tearoff"] = 0

farbe = tkinter.StringVar()
farbe.set("#FAAC58")
mAnsicht.add_radiobutton(label="Braun", variable=farbe,
    value="#FAAC58", underline=0, command=farbwechsel)
mAnsicht.add_radiobutton(label="Grün", variable=farbe,
    value="#ACFA58", underline=0, command=farbwechsel)
mAnsicht.add_separator()

rand = tkinter.StringVar()
rand.set("flat")
mAnsicht.add_checkbutton(label="Rand", variable=rand,
    onvalue="solid", offvalue="flat", underline=0, command=randwechsel)

mLeiste.add_cascade(label="Datei", menu=mDatei)
mLeiste.add_cascade(label="Ansicht", menu=mAnsicht)
fenster[ "menu"] = mLeiste

fenster.mainloop()

```

Listing 11.17 Datei »gui_menu.py«

Das Menü DATEI sieht aus wie in [Abbildung 11.26](#). Nur der Menüpunkt BEENDEN ist mit einer Funktion belegt, die anderen Menüpunkte dienen ausschließlich der Darstellung.

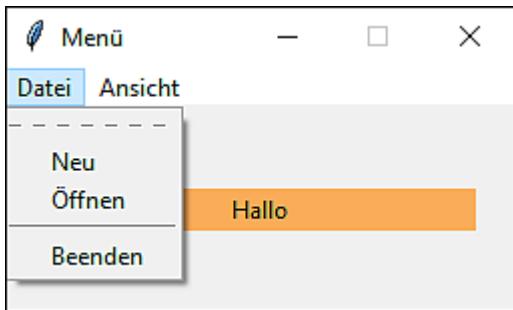


Abbildung 11.26 Menü »Datei«

Das Menü **ANSICHT** sieht aus wie in [Abbildung 11.27](#). Die Betätigung eines der beiden Menüpunkte führt zu einer Änderung der Hintergrundfarbe bzw. des Rands des Labels.

Oberhalb jedes Menüs wird standardmäßig eine gestrichelte Linie angezeigt, wie im Menü **DATEI**. Ein Klick auf diese Linie bewirkt die Auslagerung (*Tear-off*) des betreffenden Menüs als eigenständiges Fenster. Beim Menü **ANSICHT** wird diese Möglichkeit unterdrückt.

Das Label erhält über die Eigenschaft `bg` (kurz für *background*) die Hintergrundfarbe `#FAAC58` für Hellbraun.

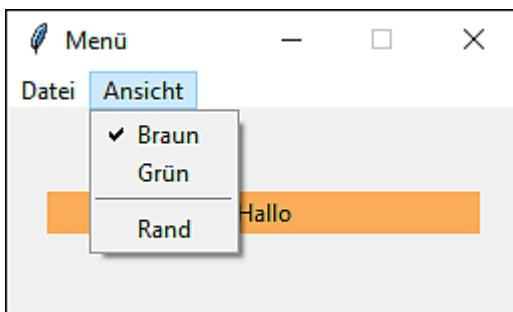


Abbildung 11.27 Menü »Ansicht«

Anschließend wird ein Objekt der Klasse `Menu` als Menüleiste für das Fenster erzeugt. Über die Variable `mLeiste` kann darauf zugegriffen werden. Innerhalb der Menüleiste wird das Menü **DATEI** erzeugt, auf das über die Variable `mDatei` zugegriffen werden kann.

Die Methode `add_command()` erstellt einen Standard-Menüpunkt, der zur Ausführung eines Befehls führt. Die Eigenschaft `label` dient der sichtbaren Darstellung des Menüpunkts. Die Methode

`add_separator()` erzeugt eine optische Trennlinie zwischen thematisch getrennten Menüpunkten.

Innerhalb der Menüleiste wird das zweite Menü mit dem Namen `ANSICHT` erzeugt, auf das über die Variable `mAnsicht` zugegriffen werden kann. Die Eigenschaft `tearoff` wird mit 0 belegt, damit das Menü nicht abgetrennt werden kann.

Es wird die Widget-Variablen `farbe` für die Auswahl der Hintergrundfarbe des Labels erstellt, mit dem passenden Startwert `#FAAC58`.

Mithilfe der Methode `add_radiobutton()` werden zwei Menüpunkte als zusammengehörige Gruppe erzeugt, die ähnlich wie Radiobuttons im Anwendungsfenster reagieren. Über den Parameter `variable` wird die gemeinsame Verbindung zur Widget-Variablen `farbe` erstellt. Der Wert der Eigenschaft `value` steht für die jeweilige Farbe. Nach Betätigung eines der Menüpunkte wird die Funktion `farbwechsel()` aufgerufen, in der der aktuell ausgewählte Farbwert mithilfe der Methode `get()` ermittelt und zur Einstellung der Hintergrundfarbe genutzt wird.

Die Eigenschaft `underline` legt den Index des sichtbar unterstrichenen Buchstabens eines Menüpunkts fest. Nach Auswahl des Menüs genügt die Eingabe des betreffenden Buchstabens zur Auswahl des Menüpunkts.

Es wird die Widget-Variablen `rand` für die Auswahl des Rands des Labels erstellt, mit dem passenden Startwert »flat«.

Mithilfe der Methode `add_checkbutton()` wird ein Menüpunkt erzeugt, der ähnlich wie ein Checkbutton im Anwendungsfenster reagiert. Über den Parameter `variable` wird die Verbindung zur Widget-Variablen `rand` erstellt. Die Werte der beiden Eigenschaften `onvalue` und `offvalue` stehen für die jeweilige Einstellung des Rands. Nach Betätigung des Menüpunkts wird die Funktion `randwechsel()` aufgerufen, in der mithilfe der Methode

`get()` ermittelt wird, ob der Menüpunkt ausgewählt ist oder nicht. Entsprechend wird der Rand eingestellt.

Die beiden zuvor erstellten Menüs `DATEI` und `ANSICHT` werden der Menüleiste mithilfe der Methode `add_cascade()` hinzugefügt. Die Eigenschaft `menu` enthält eine Referenz auf das jeweilige Menü-Objekt. Die Menüleiste wird dem Anwendungsfenster ebenfalls über die Eigenschaft `menu` hinzugefügt.

Unterschied unter macOS

Die Menüs erscheinen, wie beim Mac üblich, in der Mac-Menüleiste.

11.5.2 Kontextmenüs

Ein Kontextmenü erzeugen Sie auf ähnliche Weise wie eine Menüleiste. Folgende Schritte sind erforderlich:

- Erzeugung eines Objekts der Klasse `Menu` als Kontextmenü
- Erzeugung der Menüpunkte innerhalb des Kontextmenüs
- Verbindung eines Ereignisses mit einem Widget, meist: Herunterdrücken der rechten Maustaste
- Aufruf des Kontextmenüs als Folge dieses Ereignisses in der Nähe des betreffenden Widgets

Im nachfolgenden Programm kann die Hintergrundfarbe eines Labels über ein Kontextmenü geändert werden:

```
import tkinter

def zeigeKontext(e):
    mKontext.tk_popup(e.x_root, e.y_root)

def farbwechsel():
    lbHallo[ "bg"] = farbe.get()

def ende():
    fenster.destroy()

fenster = tkinter.Tk()
```

```

fenster.title("Kontextmenü")
fenster.resizable(0, 0)

lbHallo = tkinter.Label(fenster, text="Hallo", width=20, bg="#FAAC58")
lbHallo.bind("<Button 3>", zeigeKontext)
lbHallo.grid(row=0, column=0, padx=20, pady=40)

farbe = tkinter.StringVar()
farbe.set("#FAAC58")

mKontext = tkinter.Menu(fenster)
mKontext["tearoff"] = 0
mKontext.add_radiobutton(label="Braun", variable=farbe,
    value="#FAAC58", command=farbwechsel)
mKontext.add_radiobutton(label="Grün", variable=farbe,
    value="#ACFA58", command=farbwechsel)

buEnde = tkinter.Button(fenster, text="Ende", command=ende, width=10)
buEnde.grid(row=0, column=1, padx=20, pady=20)

fenster.mainloop()

```

Listing 11.18 Datei »gui_kontext.py«

Nach Betätigung der rechten Maustaste innerhalb des Labels sieht es aus wie in [Abbildung 11.28](#).

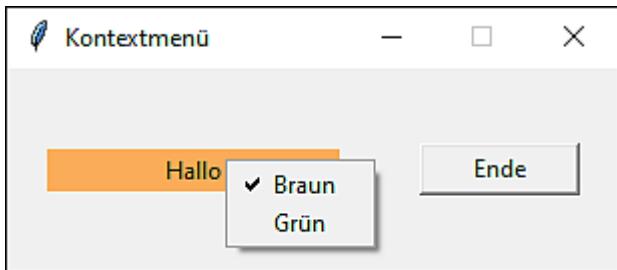


Abbildung 11.28 Kontextmenü für das Label

Das Kontextmenü wird wie ein Standard-Menü auf die bekannte Art und Weise erzeugt, aber nicht angezeigt.

Nach dem Herunterdrücken der rechten Maustaste im Label wird die Funktion `zeigeKontext()` aufgerufen. Dabei wird ein Event-Objekt übergeben, das unter anderem in den beiden Eigenschaften `x_root` und `y_root` die Koordinaten des Mausklicks enthält, bezogen auf das Anwendungsfenster.

In der Funktion `zeigeKontext()` wird für das Menü die Methode `tk_popup()` aufgerufen, die zur Anzeige des Menüs führt. Dabei werden die Koordinaten des Mausklicks übergeben. An dieser

Position befindet sich anschließend die linke obere Ecke des angezeigten Menüs.

Wird einer der beiden Menüpunkte des Kontextmenüs aufgerufen, wird die Farbe im Label mithilfe der Funktion `farbwechsel()` eingestellt.

Unterschiede unter macOS

Da die rechte Maustaste anders abgefangen wird, soll das Kontextmenü nach Betätigung der linken Maustaste erscheinen. Die Anweisung zur Bindung wird entsprechend geändert

```
lb.bind("<Button 1>", lbpop).
```

11.5.3 Messageboxen

Bei einer Messagebox handelt es sich um ein kleines vorgefertigtes Dialogfeld, das die Übermittlung einer einfachen Nachricht zwischen dem Benutzer und dem Programm vereinfacht. Sie rufen es mithilfe einer der Funktionen des Moduls `tkinter.messagebox` auf. Die verschiedenen Typen von Messageboxen haben die folgenden Aufgaben:

- `showinfo()`: eine Information an den Benutzer übermitteln
- `showwarning()`: den Benutzer warnen
- `showerror()`: einen Fehler an den Benutzer melden
- `askyesno()`: die Antwort »Ja« oder »Nein« vom Benutzer erfragen
- `askokcancel()`: die Antwort »OK« oder »Abbrechen« vom Benutzer erfragen
- `askretrycancel()`: die Antwort »Wiederholen« oder »Abbrechen« vom Benutzer erfragen
- `show()`: mit einer eigenen Auswahl von Buttons eine Antwort vom Benutzer erfragen

Im folgenden Programm werden alle genannten Funktionen jeweils über einen eigenen Button aufgerufen. Ergibt die Bedienung der Messagebox eine Antwort, so wird sie in einem Label des Anwendungsfensters angezeigt:

```
import tkinter, tkinter.messagebox as mb

def info():
    mb.showinfo("Box", "Info")

def warning():
    mb.showwarning("Box", "Warnung")

def error():
    mb.showerror("Box", "Fehler")

def yesno():
    antwort = mb.askyesno("Box", "Ja oder Nein")
    lbAusgabe[ "text"] = "Ja" if antwort == 1 else "Nein"

def okcancel():
    antwort = mb.askokcancel("Box", "OK oder Abbrechen")
    lbAusgabe[ "text"] = "OK" if antwort == 1 else "Abbrechen"

def retrycancel():
    antwort = mb.askretrycancel("Box", "Wiederholen oder Abbrechen")
    lbAusgabe[ "text"] = "Wiederholen" if antwort == 1 else "Abbrechen"

def frage():
    msg_box = mb.Message(fenster, type=mb.ABORTRETRYIGNORE,
                         icon=mb.QUESTION, title="Box",
                         message="Abbrechen, Wiederholen oder Ignorieren")
    antwort = msg_box.show()
    if antwort == "abort":
        lbAusgabe[ "text"] = "Abbrechen"
    elif antwort == "retry":
        lbAusgabe[ "text"] = "Wiederholen"
    else:
        lbAusgabe[ "text"] = "Ignorieren"

def ende():
    fenster.destroy()

fenster = tkinter.Tk()

buInfo = tkinter.Button(fenster, text="Info", width=20, command=info)
buInfo.grid(row=0, column=0, padx=5, pady=5)

buWarning = tkinter.Button(fenster, text="Warnung", width=20, command=warning)
buWarning.grid(row=0, column=1, padx=5, pady=5)

buError = tkinter.Button(fenster, text="Fehler", width=20, command=error)
buError.grid(row=0, column=2, padx=5, pady=5)

buYesNo = tkinter.Button(fenster, text="Ja/Nein", width=20, command=yesno)
```

```

buYesNo.grid(row=1, column=0, padx=5, pady=5)

buOkCancel = tkinter.Button(fenster, text="OK/Abbrechen",
    width=20, command=okcancel)
buOkCancel.grid(row=1, column=1, padx=5, pady=5)

buRetryCancel = tkinter.Button(fenster, text="Wiederholen/Abbrechen",
    width=20, command=retrycancel)
buRetryCancel.grid(row=1, column=2, padx=5, pady=5)

buFrage = tkinter.Button(fenster, text="Allgemeine Frage",
    width=20, command=frage)
buFrage.grid(row=2, column=0, padx=5, pady=5)

buEnde = tkinter.Button(fenster, text="Ende", width=20, command=ende)
buEnde.grid(row=2, column=2, padx=5, pady=5)

lbAusgabe = tkinter.Label(fenster, text="(leer)")
lbAusgabe.grid(row=3, column=1, padx=5, pady=5)

fenster.mainloop()

```

Listing 11.19 Datei »gui_nachricht.py«

Abbildung 11.29 bis Abbildung 11.35. geben jeweils die Darstellung der verschiedenen Messageboxen unter Windows wieder.

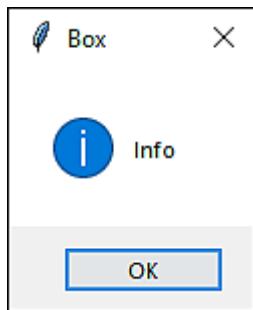


Abbildung 11.29 Information, mit Bestätigung

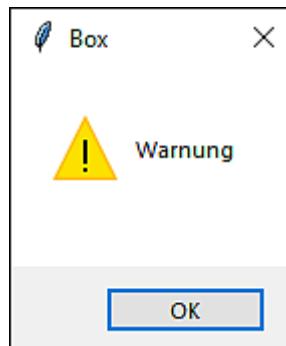


Abbildung 11.30 Warnung, mit Bestätigung

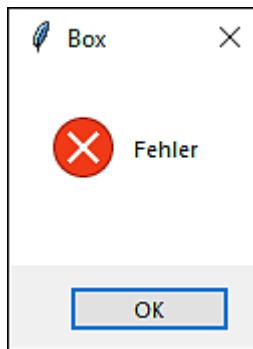


Abbildung 11.31 Fehler, mit Bestätigung

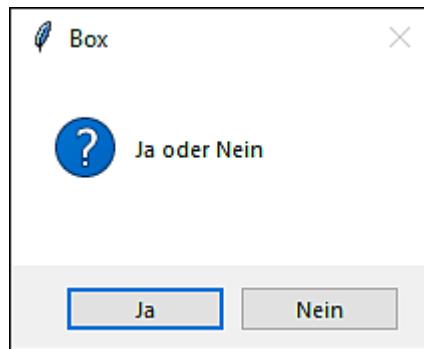


Abbildung 11.32 Mit »Ja« oder »Nein« antworten

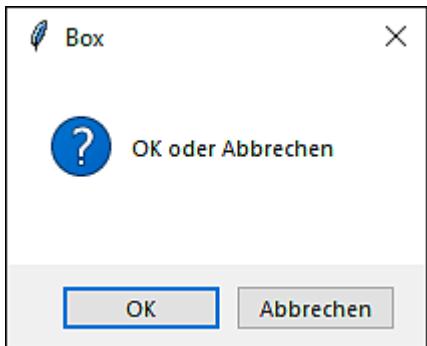


Abbildung 11.33 Mit »OK« oder »Abbrechen« antworten

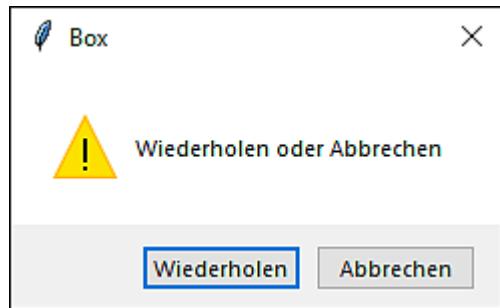


Abbildung 11.34 Mit »Wiederholen« oder »Abbrechen« antworten

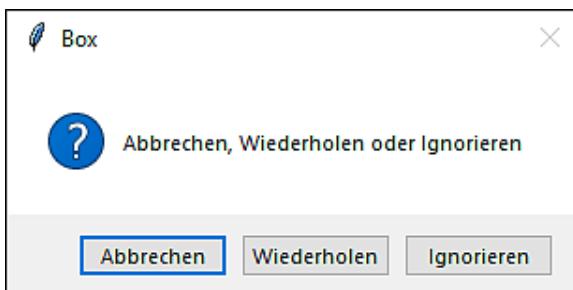


Abbildung 11.35 Eine allgemeine Frage beantworten

Die `show-` und `ask-`Funktionen haben jeweils zwei Parameter. Der erste Parameter enthält den Text für den Titel der Box. Der zweite Parameter enthält den Informationstext für die Benutzerin bzw. die Frage neben den Buttons. Zusätzlich werden passende Symbole (Icons) dargestellt, und gegebenenfalls wird der jeweilige Systemton ausgegeben.

Die drei `ask`-Funktionen, die eine Frage an die Benutzerin enthalten, liefern einen Rückgabewert: Nach dem Betätigen des linken Buttons den Wert 1, nach dem Betätigen des rechten Buttons den Wert 0.

Die allgemeine Funktion `show()` ist nicht so komfortabel zu benutzen wie die anderen Funktionen, bietet aber mehr Möglichkeiten bei der Gestaltung. Es wird zunächst ein Objekt der Klasse `Message` erzeugt, dem verschiedene Eigenschaften gegeben werden: `type` (welche Buttons), `icon` (dargestelltes Symbol), `title` (Titel) und `message` (Informationstext neben den Buttons). Anschließend wird die Funktion `show()` für dieses Objekt

aufgerufen. Sie liefert eine Zeichenkette, die den Typ des gedrückten Buttons enthält.

11.5.4 Eigene Dialogfelder

Sie können auch zusätzliche Dialogfelder erzeugen und damit Ihre Anwendung für den Benutzer übersichtlicher gestalten. Dabei sollten Sie darauf achten, dass der Benutzer nur die gewünschten Fenster und Elemente bedienen kann.

Im nachfolgenden Programm kann der Benutzer aus dem Anwendungsfenster ein zweites Dialogfeld öffnen. Die Bedienung des Anwendungsfensters hat erst wieder Auswirkungen, nachdem er das zweite Dialogfeld geschlossen hat:

```
import tkinter

def erzeugeZwei():
    global status, fensterZwei
    if status != "fenster":
        return
    status = "fensterZwei"
    fensterZwei = tkinter.Toplevel(fenster)
    fensterZwei.title("Zwei")
    lbHallo = tkinter.Label(fensterZwei, text="Hallo", width=10)
    lbHallo.grid(row=0, column=0, padx=20, pady=10)
    buEndeZwei = tkinter.Button(fensterZwei, text="Ende Zwei",
                                 width=10, command=endeZwei)
    buEndeZwei.grid(row=0, column=1, padx=20, pady=10)

def endeZwei():
    global status
    fensterZwei.destroy()
    status = "fenster"

def ende():
    if status == "fenster":
        fenster.destroy()

fenster = tkinter.Tk()
fenster.title("Fenster")
fenster.resizable(0, 0)
status = "fenster"

buZwei = tkinter.Button(fenster, text="Zwei", width=10, command=erzeugeZwei)
buZwei.grid(row=0, column=0, padx=20, pady=10)

buEnde = tkinter.Button(fenster, text="Ende", width=10, command=ende)
buEnde.grid(row=0, column=1, padx=20, pady=10)
```

```
fenster.mainloop()
```

Listing 11.20 Datei »gui_fenster.py«

In Abbildung 11.36 sehen Sie das Anwendungsfenster und das zweite Dialogfeld.

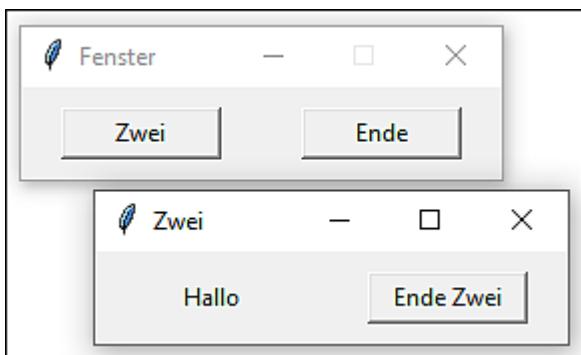


Abbildung 11.36 Anwendungsfenster und zweites Dialogfeld

Die globale Variable `status` legt fest, welches Fenster vom Benutzer bedient werden kann: Sie hat den Wert »fenster«, falls das Anwendungsfenster geöffnet ist, und den Wert »fensterZwei«, falls zusätzlich das zweite Dialogfeld geöffnet ist.

Nach der Betätigung des Buttons `ZWEI` wird die Funktion `erzeugeZwei()` aufgerufen. In dieser Funktion wird die globale Variable `status` geprüft. Hat Sie den Wert »fenster«, ist das zweite Dialogfeld noch nicht geöffnet und kann jetzt geöffnet werden. Hat sie nicht den Wert »fenster«, ist das zweite Dialogfeld bereits geöffnet und kann nicht noch einmal geöffnet werden.

Kann das zweite Dialogfeld geöffnet werden, wird die globale Variable auf den Wert »fensterZwei« gesetzt. Mithilfe der Funktion `Toplevel()` wird das zweite Dialogfeld erzeugt. Eine Referenz darauf wird in der globalen Variablen `fensterZwei` gespeichert. Im zweiten Dialogfeld werden ein Label und ein Button erstellt. Nach der Betätigung des Buttons wird die Funktion `endeZwei()` aufgerufen, in der das zweite Dialogfeld geschlossen und die globale Variable `status` wieder zurück auf den Wert »fenster« gesetzt wird.

11.6 Zeichnungen und Animationen

Zeichnungen können mithilfe eines Canvas-Objekts (deutsch: Leinwand) erstellt werden. Die einzelnen Objekte einer Zeichnung können auf dieser Leinwand abgebildet, bewegt und animiert werden.

11.6.1 Verschiedene Zeichnungsobjekte

Im nachfolgenden Programm werden verschiedene Objekte in einer Zeichnung abgebildet, siehe [Abbildung 11.37](#):

```
import tkinter

fenster = tkinter.Tk()
fenster.title("Canvas, Objekte")
fenster.geometry("400x200")
fenster.resizable(0, 0)

cv = tkinter.Canvas(fenster, bg="#E0E0E0")
cv.pack(fill="both", expand=True)
cv.create_line(20, 50, 70, 50, 70, 150, fill="#A0A0A0", width=3, arrow="last")
cv.create_rectangle(90, 50, 140, 150, fill="#A0A0A0", outline="#FFFFFF", width=3)
cv.create_oval(160, 50, 210, 150, fill="#A0A0A0", width=0)
cv.create_polygon(230, 50, 280, 50, 280, 150, fill="#A0A0A0")
cv.create_text(300, 50, text="Hallo", anchor="nw")

fenster.mainloop()
```

Listing 11.21 Datei »canvas_objekte.py«

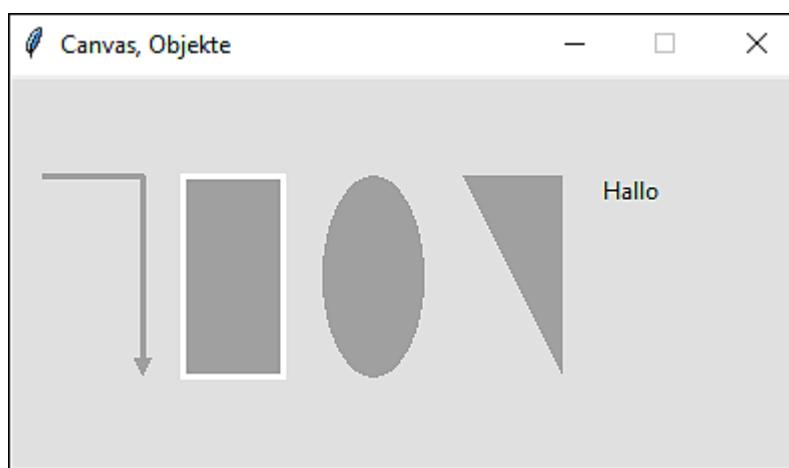


Abbildung 11.37 Zeichnungsobjekte

Der Variablen `cv` wird eine Referenz auf das neu erstellte Canvas-Objekt übergeben. Das Objekt hat als Hintergrund eine hellgraue Zeichnungsfläche.

Das Canvas-Objekt soll sich über die gesamte Benutzeroberfläche erstrecken. Für den Parameter `fill` gibt es die folgenden Möglichkeiten:

- Mit dem Standardwert `none` wird ein Widget nicht ausgedehnt.
- Mit dem Wert `x` füllt es das umgebende Element horizontal aus, mit dem Wert `y` vertikal, mit dem Wert `both` wie hier in beiden Dimensionen.

Mit dem Wert `True` für den Parameter `expand` belegt ein Widget den gesamten Platz, der nicht von anderen Widgets belegt wird. Der Standardwert ist `False`.

Die Methode `create_line()` dient zur Erzeugung einer Linie. Sie besteht aus geraden Verbindungen zwischen einer beliebigen Anzahl von Punkten. Jeder Punkt wird mithilfe von zwei Koordinaten angegeben, dem X-Wert und dem Y-Wert. Der X-Wert wird vom linken Rand des Canvas-Objekts gemessen, der Y-Wert vom oberen Rand.

Es gibt unter anderem die folgenden optionalen Parameter:

- Mithilfe der Parameter `fill` und `width` werden Farbe und Dicke der Linie eingestellt. Die Standardwerte sind »Schwarz« bzw. »1«.
- Der Parameter `arrow` wird benötigt, falls eine Linie einen Pfeilkopf zu Beginn oder am Ende haben soll. Es gibt die Werte `first`, `last` und `both` für den Beginn der Linie, für das Ende der Linie oder für beide. Wird dieser Parameter weggelassen, besitzt die Linie keinen Pfeilkopf.

Mithilfe der Methode `create_rectangle()` wird ein Rechteck gezeichnet. Haben die Seiten des Rechtecks dieselbe Länge,

handelt es sich um ein Quadrat. Zur Erstellung des Rechtecks werden die Koordinaten der linken oberen Ecke und der rechten unteren Ecke angegeben.

Es gibt unter anderem die folgenden optionalen Parameter:

- Der Parameter `fill` dient zur Einstellung der Füllfarbe des Rechtecks. Es ist standardmäßig nicht mit einer Farbe gefüllt.
- Mit den Parametern `outline` und `width` werden Farbe und Dicke der Rahmenlinie eingestellt. Die Standardwerte sind, wie bei einer Linie, »Schwarz« bzw. »1«. Mit `width=0` sorgen Sie dafür, dass das Rechteck keine Rahmenlinie besitzt.

Die Methode `create_oval()` erzeugt ein Oval innerhalb der Koordinaten eines umgebenden Rechtecks. Haben die Seiten des umgebenden Rechtecks dieselbe Länge, handelt es sich um einen Kreis. Die optionalen Parameter entsprechen denjenigen eines Rechtecks.

Mithilfe der Methode `create_polygon()` wird ein Vieleck gezeichnet, zum Beispiel ein Dreieck. Die Kanten des Polygons werden wie eine Linie mithilfe einer beliebigen Anzahl von Punkten aus X- und Y-Koordinaten angegeben. Die optionalen Parameter entsprechen denjenigen eines Rechtecks. Allerdings ist ein Polygon standardmäßig mit einer Farbe gefüllt.

Die Methode `create_text()` erzeugt ein Textfeld an einem Punkt, der mithilfe von Koordinaten für X und Y angegeben wird. Der Parameter `text` enthält den dargestellten Text. Der Bezugspunkt für die Koordinaten ist standardmäßig das Zentrum des Textfelds. Mit dem optionalen Parameter `anchor` stellen Sie einen anderen Bezugspunkt ein. Hier wird mit dem Wert »nw« die linke obere Ecke des Textfelds zum Bezugspunkt.

Die Methoden liefern jeweils eine Referenz auf das neu erstellte Objekt zurück. Diese wird hier nicht benötigt und daher nicht gespeichert.

11.6.2 Zeichnungsobjekte steuern

Die Methode `move()` dient zum Verschieben eines Zeichnungsobjekts, die Methode `coords()` liefert die aktuelle Position eines Zeichnungsobjekts. Die Methode `bind_all()` verbindet ein Tastaturereignis mit einem Widget.

Im nachfolgenden Programm kann ein Rechteck in einem Canvas mithilfe der WASD-Tasten (Tasten **W**, **A**, **S** und **D**) verschoben werden. Die aktuelle Position des Rechtecks wird zu Beginn und nach jeder Tastenbetätigung in einem Textfeld ausgegeben, siehe [Abbildung 11.38](#):

```
import tkinter

def position():
    x0, y0, x1, y1 = cv.coords(spieler)
    tx = f"{int(x0)} {int(y0)} {int(x1)} {int(y1)}"
    cv.itemconfigure(ausgabe, text=tx)

def taste(e):
    if e.char == "w":
        cv.move(spieler, 0, -10)
    elif e.char == "a":
        cv.move(spieler, -10, 0)
    elif e.char == "s":
        cv.move(spieler, 0, 10)
    elif e.char == "d":
        cv.move(spieler, 10, 0)
    position()

fenster = tkinter.Tk()
fenster.title("Canvas, Tasten")
fenster.geometry("400x200")
fenster.resizable(0, 0)

cv = tkinter.Canvas(fenster, bg="#E0E0E0")
cv.bind_all("<Key>", taste)
cv.pack(fill="both", expand=True)
spieler = cv.create_rectangle(175, 75, 225, 125, fill="#A0A0A0",
                             outline="#A0A0A0")
ausgabe = cv.create_text(20, 20, text="", anchor="nw")
position()

fenster.mainloop()
```

Listing 11.22 Datei »canvas_tasten.py«

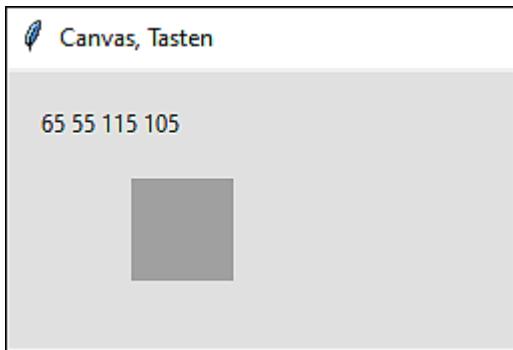


Abbildung 11.38 Zeichnungsobjekte bewegen, Position ausgeben

Das Ereignis »Benutzer betätigt Taste« wird, bezogen auf das Canvas-Objekt, mithilfe der Methode `bind_all()` mit der Callback-Funktion `taste()` verbunden. Anschließend wird im Canvas ein Rechteck erstellt, das zum Beispiel den Benutzer in einem Spiel repräsentieren soll. Eine Referenz auf das Rechteck wird in der Variablen `spieler` gespeichert. Es folgt die Erstellung eines Textfelds im Canvas. Eine Referenz darauf wird in der Variablen `ausgabe` gespeichert. Als Letztes wird die Startposition des Rechtecks mithilfe der Methode `position()` ausgegeben.

Der Funktion `taste()` wird ein Ereignis-Objekt übergeben, das Informationen über das Ereignis enthält. Die Eigenschaft `char` des Ereignis-Objekts enthält das Zeichen, das auf der Tastatur betätigt wurde.

In Abhängigkeit dieses Zeichens wird die Methode `move()` für das Canvas-Objekt auf verschiedene Arten aufgerufen. Als erster Parameter wird die Referenz des Zeichnungsobjekts notiert, das verschoben wird. Es folgen zwei Werte für die Verschiebung in X-Richtung und in Y-Richtung. Beachten Sie, dass der Y-Wert für eine Verschiebung nach oben vermindert werden muss.

Als Letztes wird die neue Position des Rechtecks mithilfe der Funktion `position()` ausgegeben.

In der Funktion `position()` wird die Methode `coords()` für das Canvas-Objekt aufgerufen. Sie erwartet als Parameter eine Referenz auf das Zeichnungsobjekt, dessen Position ermittelt wird. Als Rückgabewert wird die aktuelle Position des

Zeichnungsobjekts geliefert. Im Fall eines Rechtecks handelt es sich dabei um ein Tupel aus vier `float`-Werten. Die vier Werte entsprechen den Werten, die bei Erstellung eines Rechtecks benötigt werden. Sie werden hier in ganze Zahlen umgewandelt und zu einem Text verbunden.

Mithilfe des Aufrufs der Methode `itemconfigure()` für das Canvas-Objekt können Eigenschaften eines Zeichnungsobjekts geändert werden. Hier erhält die Eigenschaft `text` des Textfelds einen neuen Inhalt.

11.6.3 Zeichnungsobjekte animieren

Die Methode `after()` des Anwendungsfensters wird für den verzögerten Aufruf einer Funktion benötigt.

Im nachfolgenden Programm findet ein solcher Aufruf mehrmals statt. Damit wird die animierte Verschiebung eines Zeichnungsobjekts von einem Startpunkt bis zu einem Endpunkt ermöglicht. Zusätzlich wird die aktuelle Position des Rechtecks in einem Textfeld ausgegeben, siehe [Abbildung 11.39](#):

```
import tkinter

def animieren():
    x0, y0, x1, y1 = cv.coords(spieler)
    tx = f"{int(x0)}"
    cv.itemconfigure(ausgabe, text=tx)
    if x0 < 330:
        cv.move(spieler, 2, 0)
        fenster.after(10, animieren)

fenster = tkinter.Tk()
fenster.title("Canvas, Animation")
fenster.geometry("400x150")
fenster.resizable(0, 0)

cv = tkinter.Canvas(fenster, bg="#E0E0E0")
cv.pack(fill="both", expand=True)

spieler = cv.create_rectangle(20, 50, 70, 100, fill="#A0A0A0", outline="#A0A0A0")
ausgabe = cv.create_text(20, 20, text="", anchor="nw")

fenster.after(100, animieren)
fenster.mainloop()
```

Listing 11.23 Datei »canvas_animation.py«

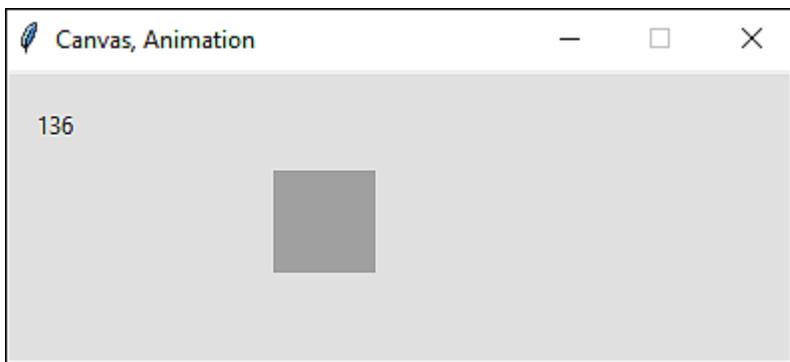


Abbildung 11.39 Animierte Verschiebung

Nach der Erstellung eines Rechtecks und eines Textfelds wird die Methode `after()` erstmals aufgerufen. Als erster Parameter wird die Verzögerung in Millisekunden notiert, als zweiter Parameter der Name der Funktion (`animieren`), deren Aufruf verzögert erfolgt.

In der Funktion `animieren()` wird mithilfe der Methode `coords()` die Position des Rechtecks ermittelt. Das zurückgelieferte Tupel wird in vier Variablen entpackt. Im Programm findet nur eine Verschiebung nach rechts statt, es ändern sich also nur die X-Werte der Position. Der X-Wert der linken oberen Ecke wird ausgegeben.

Ist der Endpunkt der Verschiebung noch nicht erreicht, wird das Rechteck um ein kleines Stück nach rechts verschoben.

Anschließend wird die Funktion `animieren()` mithilfe der Methode `after()` nach 10 Millisekunden erneut aufgerufen.

11.6.4 Kollision von Zeichnungsobjekten

Mehrere Zeichnungsobjekte können teilweise oder ganz übereinanderliegen. Mithilfe von Animationen können sich Zeichnungsobjekte relativ zueinander bewegen. Bei Spielen kann es wichtig sein, die Kollision von zwei Zeichnungsobjekten, also den Beginn einer Überlappung, zu erkennen.

Ein Beispiel: Ein Zeichnungsobjekt bewegt sich auf ein anderes Objekt zu. In dem Moment, in dem sich die Flächen der beiden Objekte überlappen, findet eine Kollision statt und es soll eine weitere Aktion stattfinden. Zur Erkennung einer solchen Kollision kann die Methode `find_overlapping()` des Canvas-Objekts genutzt werden.

Im nachfolgenden Programm wird ein Rechteck in Richtung eines zweiten Rechtecks animiert verschoben. Die Animation endet, sobald das animierte Rechteck mit dem zweiten Rechteck kollidiert, siehe [Abbildung 11.40](#):

```
import tkinter

def animieren():
    x0, y0, x1, y1 = cv.coords(spieler)
    treffer = cv.find_overlapping(x0, y0, x1, y1)
    if len(treffer) < 2:
        cv.move(spieler, 2, 0)
        fenster.after(10, animieren)

fenster = tkinter.Tk()
fenster.title("Canvas, Treffer")
fenster.geometry("400x150")
fenster.resizable(0, 0)

cv = tkinter.Canvas(fenster, bg="#E0E0E0")
cv.pack(fill="both", expand=True)

spieler = cv.create_rectangle(20, 50, 70, 100, fill="#A0A0A0", outline="#A0A0A0")
cv.create_rectangle(220, 50, 270, 100, fill="#606060", outline="#606060")

fenster.after(100, animieren)
fenster.mainloop()
```

Listing 11.24 Datei »canvas_kollision.py«



Abbildung 11.40 Kollision von zwei Zeichnungsobjekten

In der Funktion `animieren()` wird wiederum mithilfe der Methode `coords()` die Position des animierten Rechtecks ermittelt. Das zurückgelieferte Tupel wird in vier Variablen entpackt.

Die Methode `find_overlapping()` liefert ein Tupel, das Referenzen aller Zeichnungsobjekte enthält, die sich innerhalb eines Such-Rechtecks befinden. Das Such-Rechteck wird mithilfe der vier Aufruf-Parameter der Methode erstellt. Werden die vier Variablen übergeben, in denen die Position des animierten Rechtecks gespeichert ist, enthält das zurückgelieferte Tupel die Referenzen aller Zeichnungsobjekte, die das animierte Rechteck überlappen, also mit ihm kollidieren.

Ein Element des Tupels ist die Referenz auf das animierte Rechteck selbst. Ist also die Anzahl der Referenzen kleiner als 2, hat noch keine Kollision stattgefunden und das Rechteck kann weiter verschoben werden. Ist die Anzahl der Referenzen 2 oder höher, hat mindestens eine Kollision stattgefunden und das Rechteck wird nicht weiter verschoben.

11.7 Spiel, GUI-Version

Das bekannte Kopfrechenspiel wird in diesem Abschnitt mithilfe einer Benutzeroberfläche umgesetzt. Nach dem Start des Programms erscheint das Anwendungsfenster, siehe [Abbildung 11.41](#). Der Benutzer trägt seinen Namen ein und betätigt den Button **START**.

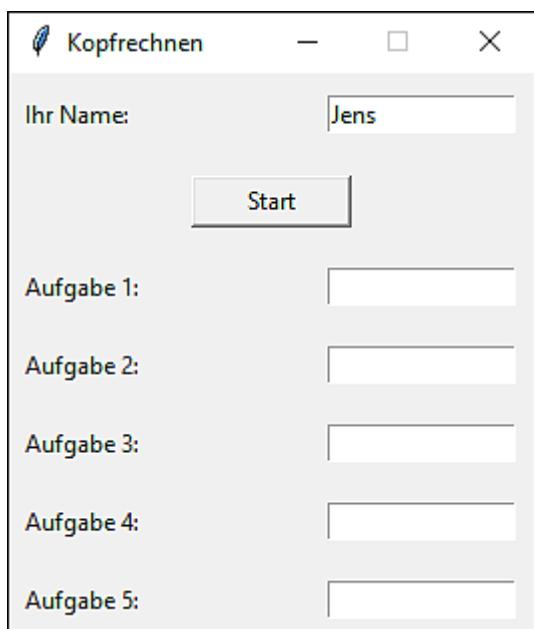


Abbildung 11.41 Nach dem Start

Anschließend erscheinen fünf Aufgaben, siehe [Abbildung 11.42](#). Die Aufschrift des Buttons wechselt zu **STOP**. Der Benutzer ermittelt die fünf Lösungen, trägt sie ein und betätigt den Button **STOP**.

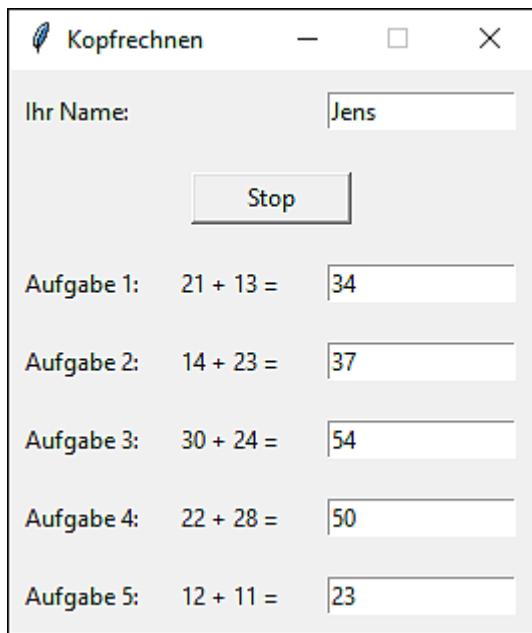


Abbildung 11.42 Aufgaben und Ergebnisse

Es folgt die Auswertung der Ergebnisse. Hat der Benutzer alle Aufgaben richtig gelöst, wird eine Highscore-Liste angezeigt, siehe [Abbildung 11.43](#). Die Speicherung der Namen und Zeiten erfolgt in einer SQLite-Datenbank.

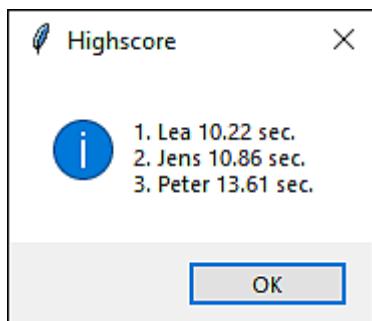


Abbildung 11.43 Highscore-Liste

Zunächst der Rahmen des Programms mit der Benutzeroberfläche:

```
import time, random, glob, sqlite3, tkinter, tkinter.messagebox as mb
def auswertung():
    ...
def start():
    ...
def aktion():
    ...

# Programm
fenster = tkinter.Tk()
fenster.title("Kopfrechnen")
fenster.resizable(0, 0)
```

```

status = "Start"

lbName = tkinter.Label(fenster, text="Ihr Name:")
lbName.grid(row=0, column=0, sticky="w", padx=5, pady=10)
enName = tkinter.Entry(fenster, width=15)
enName.grid(row=0, column=2, pady=10)
buAktion = tkinter.Button(fenster, text="Start", width=10, command=aktion)
buAktion.grid(row=1, column=0, columnspan=3, pady=10)

aufgabeListe = []
eingabeListe = []
for i in range(5):
    lbTemp = tkinter.Label(fenster, text=f"Aufgabe {i+1}:")
    lbTemp.grid(row=i+2, column=0, padx=5, pady=5)
    lbTemp = tkinter.Label(fenster, width=10, pady=5)
    lbTemp.grid(row=i+2, column=1, pady=5)
    aufgabeListe.append(lbTemp)
    enTemp = tkinter.Entry(fenster, width=15)
    enTemp.grid(row=i+2, column=2, padx=10, pady=5)
    eingabeListe.append(enTemp)

fenster.mainloop()

```

Listing 11.25 Datei »spiel_gui.py«, Rahmen des Programms

Die drei Funktionen werden weiter unten erläutert. Zunächst erscheinen ein Label, das Eingabefeld für den Namen und der Button.

Der Button hat zunächst die Aufschrift »Start«. Erst nach Eingabe des Namens und der Betätigung des Buttons erhält dieser die Aufschrift »Stop«. Auf diese Weise wird gesteuert, ob nach der Betätigung des Buttons die Aufgaben erscheinen oder die Auswertung erfolgt.

Anschließend werden zwei leere Listen angelegt. In der `for`-Schleife passiert bei jedem Durchlauf Folgendes:

- Es wird ein Label mit der Aufgabennummer erstellt. Die Referenz auf das Label wird nur temporär zur Erstellung und Platzierung benötigt.
- Es wird ein leeres Label erstellt, in dem später die Aufgabenstellung erscheint. Auch die Referenz auf dieses Label wird nur temporär benötigt. Er wird der Liste `aufgabeListe`

zugewiesen, die nach der `for`-Schleife die Referenzen auf die fünf leeren Label enthält.

- Es wird ein Eingabefeld erstellt, in dem der Benutzer seine Lösung für die Aufgabe eingeben soll. Auch die Referenz auf das Eingabefeld wird nur temporär benötigt. Er wird der Liste `eingabeListe` zugewiesen, die nach der `for`-Schleife die Referenzen auf die fünf Eingabefelder enthält.

Die Funktion `aktion()` wird nach der Betätigung des Buttons aufgerufen. Abhängig von der Aufschrift werden unterschiedliche Aktionen durchgeführt:

```
def aktion():
    if buAktion[ "text" ] == "Start":
        if enName.get() == "":
            mb.showinfo("Name", "Bitte einen Namen eintragen")
        else:
            start()
    else:
        auswertung()
```

Listing 11.26 Datei »spiel_gui.py«, Funktion »aktion()«

Hat der Button die Aufschrift »Start«, wird geprüft, ob ein Name eingegeben wurde. Ist das der Fall, wird die Funktion `start()` aufgerufen, ansonsten erscheint eine Fehlermeldung. Hat der Button nicht die Aufschrift »Start«, wird die Funktion `auswertung()` aufgerufen.

In der Funktion `start()` werden fünf Aufgaben erzeugt und angezeigt:

```
def start():
    global ergebnisListe, startzeit
    ergebnisListe = []
    for i in range(5):
        a = random.randint(10,30)
        b = random.randint(10,30)
        ergebnisListe.append(a + b)
        aufgabeListe[ i ][ "text" ] = f"{a} + {b} ="
    startzeit = time.time()
    buAktion[ "text" ] = "Stop"
```

Listing 11.27 Datei »spiel_gui.py«, Funktion »start()«

Es wird eine weitere leere Liste erstellt. In der `for`-Schleife werden zwei zufällige Werte für die Aufgabe ermittelt. Das richtige Ergebnis wird berechnet und der Liste hinzugefügt. Die Aufgabenstellung wird angezeigt, die Startzeit wird ermittelt, und die Aufschrift des Buttons wechselt zu »Stop«.

In der Funktion `auswertung()` werden die Eingaben des Benutzers ausgewertet. Hat der Benutzer alle fünf Aufgaben richtig gelöst, werden sein Name und die benötigte Zeit in der SQLite-Datenbank eingetragen:

```
def auswertung():
    global ergebnisListe, startzeit

    differenz = time.time() - startzeit
    richtig = 0
    for i in range(5):
        try:
            if int(eingabeListe[ i ].get()) == ergebnisListe[ i ]:
                richtig += 1
        except:
            pass

    if richtig < 5:
        mb.showinfo("Kein Highscore", "Leider kein Highscore")
        fenster.destroy()
        return

    if not glob.glob("spiel_gui_highscore.db"):
        con = sqlite3.connect("spiel_gui_highscore.db")
        cursor = con.cursor()
        sql = "CREATE TABLE daten(name TEXT, zeit REAL)"
        cursor.execute(sql)
        con.close()

    con = sqlite3.connect("spiel_gui_highscore.db")
    cursor = con.cursor()
    sql = f"INSERT INTO daten VALUES(?, {differenz})"
    cursor.execute(sql, (enName.get(), ))
    con.commit()
    con.close()

    con = sqlite3.connect("spiel_gui_highscore.db")
    cursor = con.cursor()
    sql = "SELECT * FROM daten ORDER BY zeit LIMIT 10"
    cursor.execute(sql)
    ausgabe = ""
    i = 1
    for dsatz in cursor:
        ausgabe += f"{i}. {dsatz[ 0 ]} {round(dsatz[ 1 ], 2)} sec.\n"
        i += 1
    mb.showinfo("Highscore", ausgabe)
```

```
con.close()  
fenster.destroy()
```

Listing 11.28 Datei »spiel_gui.py«, Funktion »auswertung()«

Als Erstes wird die benötigte Zeit ermittelt. Anschließend werden die fünf Eingaben des Benutzers mit den fünf richtigen Ergebnissen verglichen. Sind nicht alle Ergebnisse richtig, erscheint ein Nachrichtenfenster mit einer entsprechenden Meldung und das Programm endet.

Gibt es noch keine Highscore-Datenbank, wird sie neu erstellt. In der Datenbank wird ein Datensatz mit Namen und Zeit eingetragen.

Mithilfe einer `SELECT`-Anweisung und des Schlüsselworts `LIMIT` werden die zehn Datensätze mit den geringsten Zeiten ermittelt und in einem Nachrichtenfenster angezeigt, auf zwei Stellen nach dem Komma gerundet. Anschließend endet das Programm.

12 Benutzeroberflächen mit PyQt

Die Bibliothek *Qt* ermöglicht wie die Bibliothek *Tk* die einheitliche Programmierung grafischer Benutzeroberflächen (GUIs) für verschiedene Betriebssysteme. Für Python gibt es *PyQt* als frei verfügbare Schnittstelle zu Qt. In diesem Kapitel folgt ein Einstieg in die aktuelle Version 6 von PyQt. Die Schnittstelle lässt sich wie folgt installieren:

```
pip install PyQt6
```

12.1 Ein erstes Programm

Mithilfe eines ersten Programms wird eine Anwendung mit einer PyQt-GUI erstellt. Sie enthält ein einzelnes Widget, und zwar den Button ENDE, siehe [Abbildung 12.1](#). Nach der Betätigung dieses Buttons wird die GUI geschlossen und die Anwendung beendet.

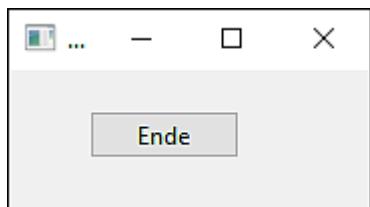


Abbildung 12.1 Erstes Programm mit PyQt

Zunächst das Programm:

```
import PyQt6.QtWidgets as wi
import sys

class Fenster(wi.QWidget):
    def __init__(self):
        super().__init__()
```

```

buEnde = wi.QPushButton("Ende", self)
buEnde.move(40, 20)
buEnde.clicked.connect(wi.QApplication.instance().quit)

self.setWindowTitle("...")  

self.show()

anwendung = wi.QApplication(sys.argv)
fenster = Fenster()
sys.exit(anwendung.exec())

```

Listing 12.1 Datei »pyqt_erstes.py«

Das Modul `PyQt6.QtWidgets` enthält die verschiedenen Klassen für die Widgets, also die Elemente des Anwendungsfensters. Es wird hier mit dem Alias `wi` importiert. Zum Start und zum Beenden der Anwendung wird das Modul `sys` benötigt.

Im Hauptprogramm wird ein Objekt der Klasse `QApplication` zur Steuerung der GUI-Anwendung erstellt. Die Parameter der Kommandozeile werden als Parameter für den Konstruktor weitergegeben.

Als Nächstes wird ein Objekt der eigenen Klasse `Fenster` erstellt, das zur Gestaltung des Anwendungsfensters dient. Diese Klasse erbt von der Klasse `QWidget`, der Basisklasse der verschiedenen Widget-Klassen.

Für das Objekt der Klasse `QApplication` wird die Methode `exec()` aufgerufen. Damit wird eine Endlosschleife gestartet, in der auf Ereignisse gewartet wird. Wird das Anwendungsfenster geschlossen, wird anschließend mithilfe der Methode `exit()` auch die Anwendung beendet.

Im Konstruktor der Klasse `Fenster` muss zunächst der Konstruktor der Basisklasse `QWidget` aufgerufen werden. Anschließend werden die Elemente des Anwendungsfensters erstellt.

Ein Objekt der Klasse `QPushButton` entspricht einer Standardschaltfläche, also einem Button. Hier erhält er die Aufschrift »Ende« und wird mithilfe von `self` dem Fenster-Objekt zugeordnet. Die Methode `move()` verschiebt den Button von der

Standardposition 0, 0 auf die gewünschte Position, jeweils gemessen von der linken oberen Ecke des Fensters.

Zum Bearbeiten von Ereignissen dienen in Qt Signale und Slots. Beim Eintritt eines Ereignisses, hier des Ereignisses `clicked` des Buttons, wird ein Signal gesendet. Dieses Signal wird mithilfe der Methode `connect()` mit einem Slot verbunden, standardmäßig einer Methode.

In diesem Fall wird beim Klick auf den Button die Methode `quit()` für die Instanz, also das Objekt der Klasse `QApplication`, aufgerufen. Damit wird die Endlosschleife verlassen, die GUI geschlossen und die Anwendung beendet

Die Methode `setWindowTitle()` sorgt für die Anzeige eines Texts in der Titelzeile. Die Methode `show()` zeigt das Anwendungsfenster an.

12.2 Layout und Größe eines Anwendungsfensters

PyQt bietet die Möglichkeit, die Widgets im Anwendungsfenster komfortabel in einem *Grid-Layout*, also einem Raster aus Zeilen und Spalten, anzuordnen. Für die Größe des Anwendungsfensters gelten bestimmte Regeln.

12.2.1 Grid-Layout

Im folgenden Beispiel werden zwei Buttons und ein Label in einem Grid-Layout angeordnet:

```
import PyQt6.QtWidgets as wi
import sys

class Fenster(wi.QWidget):
    def __init__(self):
        super().__init__()
        gr = wi.QGridLayout()

        buHallo = wi.QPushButton("Hallo")
        buHallo.clicked.connect(self.hallo)
        gr.addWidget(buHallo, 0, 0)

        self.lbAusgabe = wi.QLabel("(leer)")
        gr.addWidget(self.lbAusgabe, 1, 0)

        buEnde = wi.QPushButton("Ende")
        buEnde.clicked.connect(wi.QApplication.instance().quit)
        gr.addWidget(buEnde, 2, 1)

        self.setWindowTitle("...")
        self.setLayout(gr)
        self.show()

    def hallo(self):
        self.lbAusgabe.setText("Hallo")

anwendung = wi.QApplication(sys.argv)
fenster = Fenster()
sys.exit(anwendung.exec())
```

Listing 12.2 Datei »pyqt_anordnung.py«

Nach der Betätigung des Buttons HALLO erscheint der Text »Hallo« in einem Label, siehe [Abbildung 12.2](#).

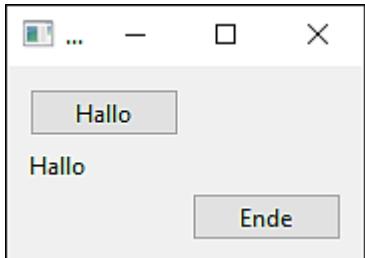


Abbildung 12.2 Anordnung der Widgets in einem Raster

Das Raster wird mithilfe eines Objekts der Klasse `QGridLayout` erstellt. Die einzelnen Widgets werden nicht mehr dem Fenster zugeordnet, sondern dem Raster. Die Zuordnung erfolgt durch den Aufruf der Methode `addWidget()` für das `QGridLayout`-Objekt. Als Parameter wird eine Referenz auf das jeweilige Widget angegeben, gefolgt von den Nummern für die Zeile und für die Spalte des Rasters. Die Nummerierung beginnt jeweils bei 0.

Beim Klick auf den Button `buHallo` wird für das Objekt der Klasse `Fenster` die Methode `hallo()` aufgerufen. Der Button wird mit den Parametern 0, 0 in der linken oberen Zelle des Rasters angezeigt.

In einem Label können Texte oder Bilder dargestellt werden. Es wird mithilfe der Klasse `QLabel` erzeugt. In der Methode `hallo()` soll der Inhalt des Labels geändert werden. Daher wird die Referenz auf das Label mithilfe von `self` als Eigenschaft der Klasse `Fenster` deklariert. Ansonsten würde es sich bei der Referenz nur um eine lokale Variable der Methode `__init__()` handeln, die in anderen Methoden nicht bekannt ist.

Durch den Aufruf der Methode `setLayout()` für das Objekt der Klasse `Fenster` wird das Layout der GUI zugeordnet.

In der Methode `hallo()` wird für das Label-Objekt die Methode `setText()` zur Änderung des Texts aufgerufen.

12.2.2 Größe des Anwendungsfensters

Die Größe des Anwendungsfensters richtet sich nach der Größe der beinhalteten Elemente. Die Größe eines Labels richtet sich wiederum nach dem Inhalt und bei einem Textinhalt nach der Schriftgröße. In den verschiedenen Betriebssystemen werden unterschiedliche Standardschriftgrößen genutzt. Daher kann es bei der Größe des Anwendungsfensters für dasselbe PyQt-Programm in den verschiedenen Betriebssystemen kleine Unterschiede geben.

Sie könnten die Größe des Anwendungsfensters mithilfe der Methode `setFixedSize()` unveränderlich festlegen. Das hätte aber aus den oben genannten Gründen die folgenden Nachteile:

- Wird während des Ablaufs des Programms in einem Label ein Text angezeigt, der zu lang ist, wird er abgeschnitten.
- Werden die Texte in einem anderen Betriebssystem standardmäßig mit einer höheren Schriftgröße angezeigt, werden sie gegebenenfalls bereits beim Start des Programms abgeschnitten.

12.3 Widget-Typen

In diesem Abschnitt lernen Sie weitere Widgets kennen:

`QLineEdit`, `QTextEdit`, `QListWidget`, `QComboBox`, `QSpinBox`, `QRadioButton`, `QButtonGroup`, `QCheckBox` und `QSlider`. Zudem wird das `QLabel`-Widget zur Darstellung von Bildern, Farben und Hyperlinks eingesetzt.

12.3.1 Einzeiliges Eingabefeld

Die Inhalte vieler Widgets können sowohl während ihrer Bedienung als auch im Nachhinein ausgewertet werden.

Ein Objekt der Klasse `QLineEdit` dient als einzeiliges Eingabefeld. Das Signal `textChanged` kann genutzt werden, um unmittelbar bei der Änderung des Inhalts des Eingabefelds, also beim Hinzufügen oder Entfernen einzelner Zeichen, zu reagieren. Meist wird der Inhalt des Eingabefelds aber erst nach der Betätigung eines Buttons geprüft und weiterverarbeitet.

Im nachfolgenden Programm werden beide Ereignisse genutzt:

```
import PyQt6.QtWidgets as wi
import sys

class Fenster(wi.QWidget):
    def __init__(self):
        super().__init__()
        gr = wi.QGridLayout()

        lbEingabe = wi.QLabel("Ihre Eingabe:")
        gr.addWidget(lbEingabe, 0, 0)

        self.leEingabe = wi.QLineEdit()
        self.leEingabe.textChanged.connect(self.eingabe)
        gr.addWidget(self.leEingabe, 1, 0)

        buVerdoppeln = wi.QPushButton("Verdoppeln")
        buVerdoppeln.clicked.connect(self.verdoppeln)
        gr.addWidget(buVerdoppeln, 2, 0)

        self.lbAusgabe = wi.QLabel("(leer)")
        gr.addWidget(self.lbAusgabe, 3, 0)
```

```

buEnde = wi.QPushButton("Ende")
buEnde.clicked.connect(wi.QApplication.instance().quit)
gr.addWidget(buEnde, 4, 1)

self.setWindowTitle("Eingabe")
self.setLayout(gr)
self.show()

def eingabe(self, tx):
    self.lbAusgabe.setText(f"Eingabe: {tx}")

def verdoppeln(self):
    try:
        zahl = float(self.leEingabe.text())
        self.lbAusgabe.setText(f"{zahl * 2}")
    except:
        self.lbAusgabe.setText("Keine Zahl")

anwendung = wi.QApplication(sys.argv)
fenster = Fenster()
sys.exit(anwendung.exec())

```

Listing 12.3 Datei »pyqt_eingabe.py«

Die Referenz `leEingabe` verweist auf das `QLineEdit`-Objekt. Das Signal `textChanged` dieses Objekts ist mit der Methode `eingabe()` verbunden. Dieser Methode wird automatisch der aktuelle Inhalt des Eingabefelds übergeben. Hier wird dieser Inhalt in einem Label ausgegeben, siehe [Abbildung 12.3](#).

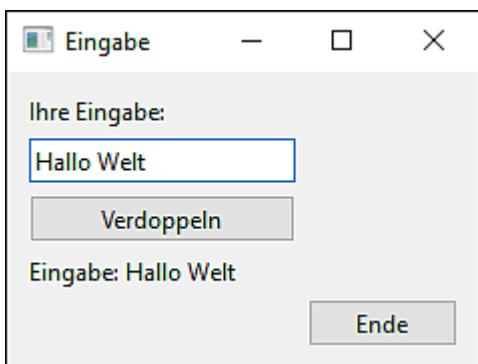


Abbildung 12.3 Ereignis »Änderung des Inhalts«

Die Betätigung des Buttons `VERDOPPELN` ist mit der Methode `verdoppeln()` verbunden. In dieser Methode wird der Inhalt des Eingabefelds mithilfe der Methode `text()` des `QLineEdit`-Objekts ermittelt. Kann dieser Inhalt mithilfe der Funktion `float()` in eine gültige Zahl umgewandelt werden, wird diese Zahl

verdoppelt und in einem Label ausgegeben, siehe [Abbildung 12.4](#). Ansonsten erscheint eine Fehlermeldung.

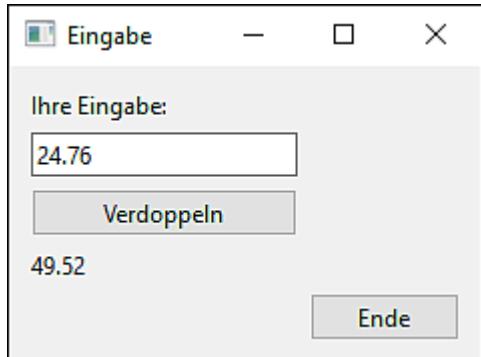


Abbildung 12.4 Betätigung des Buttons »Verdoppeln«

12.3.2 Versteckte Eingabe, Deaktivieren von Widgets

Ein `QLineEdit`-Objekt ermöglicht auch eine versteckte Eingabe, zum Beispiel für ein Passwort. Widgets können zur besseren Benutzersteuerung aktiviert oder deaktiviert werden. Beides wird im folgenden Programm gezeigt:

```
import PyQt6.QtWidgets as wi
import sys

class Fenster(wi.QWidget):
    def __init__(self):
        super().__init__()
        gr = wi.QGridLayout()

        lbPasswort = wi.QLabel("Ihr Passwort:")
        gr.addWidget(lbPasswort, 0, 0)

        self.lePasswort = wi.QLineEdit()
        self.lePasswort.setEchoMode(wi.QLineEdit.EchoMode.Password)
        gr.addWidget(self.lePasswort, 1, 0)

        buPruefen = wi.QPushButton("Prüfen")
        buPruefen.clicked.connect(self.pruefen)
        gr.addWidget(buPruefen, 2, 0)

        self.lbAusgabe = wi.QLabel("(leer)")
        gr.addWidget(self.lbAusgabe, 3, 0)

        self.buEnde = wi.QPushButton("Ende")
        self.buEnde.setEnabled(False)
        self.buEnde.clicked.connect(wi.QApplication.instance().quit)
        gr.addWidget(self.buEnde, 4, 0)

    self.setWindowTitle("Passwort")
```

```

        self.setLayout(gr)
        self.show()

    def pruefen(self):
        pw = self.lePasswort.text()
        if pw == "Bingo":
            self.lbAusgabe.setText("Zugang erlaubt")
        else:
            self.lbAusgabe.setText("Zugang nicht erlaubt")
        self.lePasswort.setText("")
        self.buEnde.setEnabled(True)

anwendung = wi.QApplication(sys.argv)
fenster = Fenster()
sys.exit(anwendung.exec())

```

Listing 12.4 Datei »pyqt_passwort.py«

Die Referenz `lePasswort` verweist auf ein `QLineEdit`-Objekt. Für dieses Objekt wird die Methode `setEchoMode()` aufgerufen. Als Parameter muss ein Element der Enumeration `EchoMode` der Klasse `QLineEdit` übergeben werden. Wird das Element `Password` genutzt, erscheint für jedes eingegebene Zeichen ein neutrales Bullet-Zeichen, siehe [Abbildung 12.5](#).

Der Button `ENDE` wurde zu Beginn durch den Aufruf der Methode `setEnabled()` mit dem Parameter `False` deaktiviert und ist nicht bedienbar.

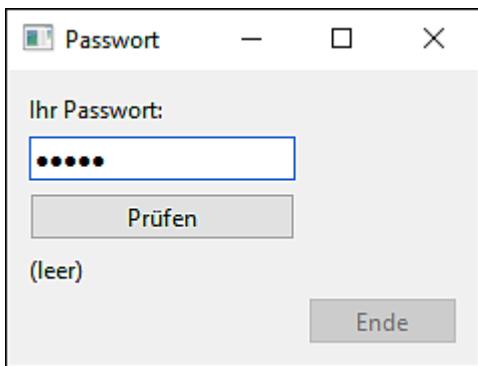


Abbildung 12.5 Versteckte Eingabe

Nach der Betätigung des Buttons `PRÜFEN` wird die Methode `pruefen()` aufgerufen. Der Inhalt des Eingabefelds wird ermittelt. Abhängig vom Inhalt erscheint eine von zwei verschiedenen Meldungen. Anschließend wird der Inhalt entfernt. Der Button `ENDE` wird durch den Aufruf der Methode `setEnabled()` mit dem

Parameter `True` aktiviert und kann jetzt bedient werden, siehe Abbildung 12.6.

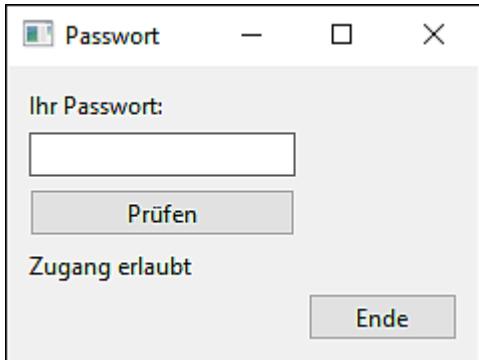


Abbildung 12.6 Aktivierung eines Widgets

12.3.3 Mehrzeiliges Eingabefeld

Ein Objekt der Klasse `QTextEdit` entspricht einem mehrzeiligen Eingabefeld zur Darstellung und Bearbeitung großer Datenmengen. Mithilfe von HTML bietet es vielfältige Möglichkeiten zur Formatierung und kann als WYSIWYG-Editor (*What you see is what you get*) dienen. Ist die Größe des Widgets nicht ausreichend, wird ein vertikaler Scrollbalken eingeblendet.

Im nachfolgenden Beispiel wird nur ein kleiner Ausschnitt der Fähigkeiten gezeigt. Ein `QTextEdit`-Objekt wird zur Darstellung von geladenen Daten aus einer Textdatei genutzt. Zudem besteht die Möglichkeit, den Inhalt des Eingabefelds in der Textdatei zu speichern:

```
import PyQt6.QtWidgets as wi
import sys

class Fenster(wi.QWidget):
    def __init__(self):
        super().__init__()
        gr = wi.QGridLayout()

        lbText = wi.QLabel("Text:")
        gr.addWidget(lbText, 0, 0)

        self.teText = wi.QTextEdit()
        self.teText.setFixedHeight(80)
        gr.addWidget(self.teText, 1, 0, 1, 3)
```

```

buLaden = wi.QPushButton("Aus Datei laden")
buLaden.clicked.connect(self.laden)
gr.addWidget(buLaden, 2, 0)

buSpeichern = wi.QPushButton("In Datei speichern")
buSpeichern.clicked.connect(self.speichern)
gr.addWidget(buSpeichern, 2, 1)

buLoeschen = wi.QPushButton("Text löschen")
buLoeschen.clicked.connect(self.loeschen)
gr.addWidget(buLoeschen, 2, 2)

self.lbAusgabe = wi.QLabel("(leer)")
gr.addWidget(self.lbAusgabe, 3, 0)

self.buEnde = wi.QPushButton("Ende")
self.buEnde.clicked.connect(wi.QApplication.instance().quit)
gr.addWidget(self.buEnde, 3, 3)

self.setWindowTitle("TextEdit")
self.setLayout(gr)
self.show()

def laden(self):
    self.teText.setText("")
    try:
        d = open("pyqt_mehrzeilig.txt")
        self.teText.setText(d.read())
        d.close()
    except:
        self.teText.setText("Datei nicht geöffnet")

def speichern(self):
    try:
        d = open("pyqt_mehrzeilig.txt", "w")
        d.write(self.teText.toPlainText())
        d.close()
    except:
        self.lbAusgabe.setText("Datei nicht geöffnet")

def loeschen(self):
    self.teText.setText("")
```

anwendung = wi.QApplication(sys.argv)
fenster = Fenster()
sys.exit(anwendung.exec())

Listing 12.5 Datei »pyqt_mehrzeilig.py«

Die Referenz `teText` verweist auf ein `QTextEdit`-Objekt, siehe [Abbildung 12.7](#). Die Höhe eines Widgets kann mithilfe der Methode `setFixedHeight()` unveränderlich festgelegt werden.

Das Widget soll sich innerhalb des Rasters über drei Spalten erstrecken. Zu diesem Zweck werden beim Aufruf der Methode

`addWidget()` zwei weitere Parameter mit den Werten 1 (Zeile) und 3 (Spalten) angehängt. Die beiden Parameter haben die Standardwerte 1 und 1.

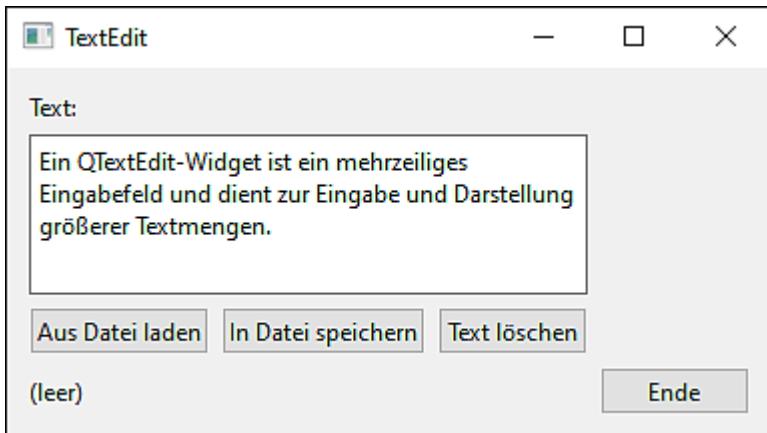


Abbildung 12.7 Mehrzeiliges Eingabefeld

Nach der Betätigung des Buttons `AUS DATEI LADEN` wird die Methode `laden()` aufgerufen. Darin wird das Textfeld mithilfe der Methode `setText()` geleert. Anschließend wird eine Textdatei geöffnet. Ihr Inhalt wird vollständig gelesen und im Textfeld dargestellt. Tritt dabei ein Fehler auf, erscheint stattdessen eine Fehlermeldung im Textfeld.

Nach der Betätigung des Buttons `IN DATEI SPEICHERN` wird die Methode `speichern()` aufgerufen. Darin wird der reine Textinhalt des Textfelds mithilfe der Methode `toPlainText()` ermittelt und in die Datei geschrieben. Tritt dabei ein Fehler auf, wird ebenfalls eine Fehlermeldung ausgegeben. Damit sie nicht den Inhalt des Textfelds überschreibt, erscheint sie in dem Label unter dem Textfeld. Die Methode `toHTML()` würde den HTML-Inhalt eines Textfelds ermitteln.

Der Button `TEXT LÖSCHEN` führt zum Aufruf der Methode `loeschen()`, in der das Textfeld gelöscht wird. Der Button `ENDE` befindet sich in Spalte 3 des Rasters, da sich die drei anderen Buttons in den Spalten 0, 1 und 2 befinden und sich das Textfeld über diese Spalten 0, 1 und 2 erstreckt.

12.3.4 Liste mit einfacher Auswahl

Sie können ein Objekt der Klasse `QListWidget` einsetzen, damit der Benutzer eine Auswahl aus einer Liste treffen kann. In diesem Abschnitt wird die Auswahl eines einzelnen Elements behandelt, im nächsten Abschnitt die gleichzeitige Auswahl von mehreren Elementen.

Das `QListWidget`-Objekt kann unmittelbar auf einen Wechsel der Auswahl reagieren. Zudem können Sie die getroffene Auswahl auch im Nachhinein auswerten. Beide Möglichkeiten werden im nachfolgenden Beispiel gezeigt:

```
import PyQt6.QtWidgets as wi
import sys

class Fenster(wi.QWidget):
    def __init__(self):
        super().__init__()
        gr = wi.QGridLayout()

        lbAuswahl = wi.QLabel("Ihre Auswahl:")
        gr.addWidget(lbAuswahl, 0, 0)

        self.liAuswahl = wi.QListWidget()
        self.liAuswahl.setFixedHeight(80)
        stadt = ["Hamburg", "Stuttgart", "Berlin", "Dortmund", "Trier",
                 "Duisburg", "Potsdam", "Halle", "Flensburg", "Augsburg"]
        for s in stadt:
            self.liAuswahl.addItem(s)
        self.liAuswahl.setCurrentRow(0)
        self.liAuswahl.itemClicked.connect(self.auswahl)
        gr.addWidget(self.liAuswahl, 1, 0)

        buAusgabe = wi.QPushButton("Ausgabe")
        buAusgabe.clicked.connect(self.ausgabe)
        gr.addWidget(buAusgabe, 2, 0)

        self.lbAusgabe = wi.QLabel("(leer)")
        gr.addWidget(self.lbAusgabe, 3, 0)

        buEnde = wi.QPushButton("Ende")
        buEnde.clicked.connect(wi.QApplication.instance().quit)
        gr.addWidget(buEnde, 4, 1)

        self.setWindowTitle("ListView")
        self.setLayout(gr)
        self.show()

    def auswahl(self, it):
        self.lbAusgabe.setText(f"Auswahl: {it.text()}")
```

```

def ausgabe(self):
    it = self.liAuswahl.currentItem()
    self.lbAusgabe.setText(f"Ausgabe: {it.text()}")


anwendung = wi.QApplication(sys.argv)
fenster = Fenster()
sys.exit(anwendung.exec())

```

Listing 12.6 Datei »pyqt_liste.py«

Die Referenz `liAuswahl` verweist auf ein `QListWidget`-Objekt. Sind nicht alle Einträge gleichzeitig zu sehen, erhält das Objekt einen vertikalen Scrollbalken, siehe [Abbildung 12.8](#).

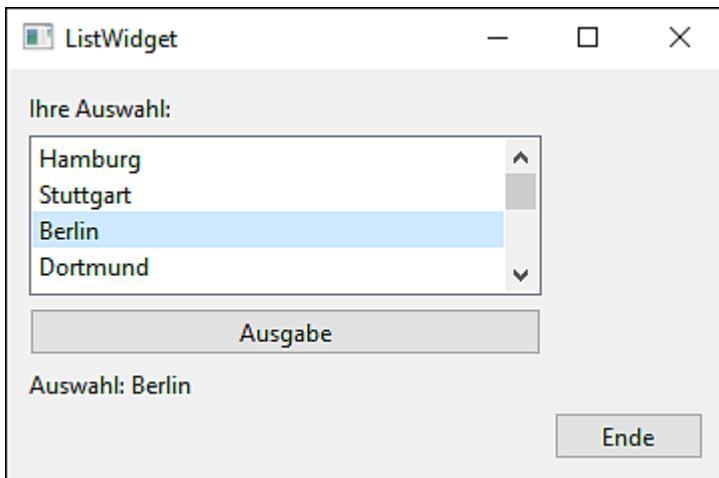


Abbildung 12.8 Liste mit einfacher Auswahl

Die dargestellten Einträge sind vom Typ `QListWidgetItem`. Sie werden dem `QListWidget` mithilfe der Methode `addItem()` hinzugefügt, hier aus einer Liste. Als Parameter wird der dargestellte Text des Eintrags übergeben.

Mithilfe der Methode `setCurrentRow()` wird ein Eintrag sichtbar markiert, damit es auch eine Auswahl gibt, falls die Benutzerin das `QListWidget`-Objekt nicht bedient hat. Als Parameter wird der Index des Eintrags übergeben. Der erste Eintrag hat den Index 0.

Die Markierung eines Eintrags entspricht dem Signal `itemClicked`. Es wird hier mit der Methode `auswahl()` verbunden. Dieser Methode wird automatisch eine Referenz auf das ausgewählte `QListWidgetItem`-Objekt übergeben. Die Methode `text()` des Objekts liefert den dargestellten Text des Eintrags.

Nach der Betätigung des Buttons AUSGABE wird wiederum die Methode ausgabe() aufgerufen. Darin wird der aktuell ausgewählte Eintrag mithilfe der Methode currentItem() ermittelt. Der zugehörige Text wird ausgegeben.

12.3.5 Liste mit mehrfacher Auswahl

Die Methode setSelectionMode() dient zum Einstellen des Auswahlmodus für ein QListWidget-Objekt. Im nachfolgenden Programm wird eine mehrfache Auswahl ermöglicht. Der Benutzer kann mithilfe der **[Strg]**-Taste mehrere Einträge auswählen, die nicht aufeinander folgen müssen. Mithilfe der **[Shift]**-Taste kann er zudem mehrere aufeinanderfolgende Einträge auswählen. Die aktuell getroffene Auswahl erscheint in einem zweiten QListWidget-Objekt, siehe Abbildung 12.9:

```
import PyQt6.QtWidgets as wi
import sys

class Fenster(wi.QWidget):
    def __init__(self):
        super().__init__()
        gr = wi.QGridLayout()

        lbAuswahl = wi.QLabel("Ihre Auswahl:")
        gr.addWidget(lbAuswahl, 0, 0)

        self.liAuswahl = wi.QListWidget()
        self.liAuswahl.setFixedHeight(80)
        self.liAuswahl.setSelectionMode(
            wi.QAbstractItemView.SelectionMode.ExtendedSelection)
        stadt = ["Hamburg", "Stuttgart", "Berlin", "Dortmund", "Trier",
                 "Duisburg", "Potsdam", "Halle", "Flensburg", "Augsburg"]
        for s in stadt:
            self.liAuswahl.addItem(s)
        self.liAuswahl.setCurrentRow(0)
        self.liAuswahl.itemClicked.connect(self.ausgabe)
        gr.addWidget(self.liAuswahl, 1, 0)

        buAusgabe = wi.QPushButton("Ausgabe")
        buAusgabe.clicked.connect(self.ausgabe)
        gr.addWidget(buAusgabe, 2, 0)

        self.liAusgabe = wi.QListWidget()
        self.liAusgabe.setFixedHeight(80)
        gr.addWidget(self.liAusgabe, 3, 0)
```

```

buEnde = wi.QPushButton("Ende")
buEnde.clicked.connect(wi.QApplication.instance().quit)
gr.addWidget(buEnde, 4, 1)

self.setWindowTitle("ListWidget")
self.setLayout(gr)
self.show()

def ausgabe(self):
    while self.liAusgabe.count() > 0:
        self.liAusgabe.takeItem(0)
    for it in self.liAuswahl.selectedItems():
        self.liAusgabe.addItem(it.text())

anwendung = wi.QApplication(sys.argv)
fenster = Fenster()
sys.exit(anwendung.exec())

```

Listing 12.7 Datei »pyqt_liste_mehrfach.py«

Als Parameter der Methode `setSelectionMode()` wird ein Element der Enumeration `SelectionMode` aus der Klasse `QAbstractItemView` erwartet. Diese Klasse dient als abstrakte Basisklasse für verschiedene Typen von Widgets, die über Einträge verfügen.

Sowohl eine Änderung der Auswahl als auch die Betätigung des Buttons `AUSGABE` führen zur Methode `ausgabe()`. Darin wird das zweite `QListWidget`-Objekt geleert und anschließend mit den ausgewählten Einträgen gefüllt.

Die Methode `count()` liefert die Anzahl der Einträge. Die Methode `takeItem()` entfernt den Eintrag mit dem angegebenen Index. Hier wird mithilfe einer `while`-Schleife mehrfach der erste Eintrag entfernt, bis die Liste leer ist.

Die Methode `selectedItems()` liefert eine Liste von `QListWidgetItem`-Objekten. Sie stehen in der Reihenfolge, in der sie ausgewählt wurden. Die Liste wird mithilfe einer Schleife durchlaufen. Der Text des jeweiligen Eintrags wird dem zweiten `QListWidget`-Objekt mithilfe der Methode `addItem()` hinzugefügt, siehe [Abbildung 12.9](#).

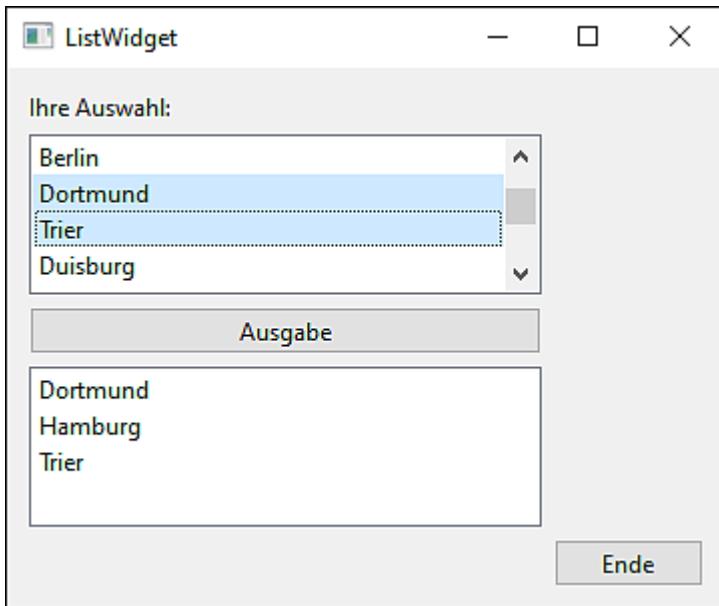


Abbildung 12.9 Liste mit mehrfacher Auswahl

12.3.6 Combobox

Mithilfe eines Objekts der Klasse `QComboBox` erstellen Sie eine Combobox, also ein Kombinationsfeld. Darin können Sie sowohl Einträge aus einer Liste auswählen als auch einen Text eintragen wie in einem Textfeld. Bei dem Objekt werden die Ereignisse »Auswahl ändert sich« und »Text ändert sich« bemerkt. Im nachfolgenden Programm wird das gezeigt:

```
import PyQt6.QtWidgets as wi
import sys

class Fenster(wi.QWidget):
    def __init__(self):
        super().__init__()
        gr = wi.QGridLayout()

        self.lbErgebnis = wi.QLabel("Ihre Auswahl oder Eingabe:")
        gr.addWidget(self.lbErgebnis, 0, 0)

        self.cbAuswahl = wi.QComboBox()
        self.cbAuswahl.setEditable(True)
        stadt = ["Hamburg", "Stuttgart", "Berlin", "Dortmund", "Trier",
                 "Duisburg", "Potsdam", "Halle", "Flensburg", "Augsburg"]
        for s in stadt:
            self.cbAuswahl.addItem(s)
        self.cbAuswahl.currentTextChanged.connect(self.eingabe)
        self.cbAuswahl.currentIndexChanged.connect(self.auswahl)
        gr.addWidget(self.cbAuswahl, 1, 0)
```

```

buAusgabe = wi.QPushButton("Ausgabe")
buAusgabe.clicked.connect(self.ausgabe)
gr.addWidget(buAusgabe, 2, 0)

buEnde = wi.QPushButton("Ende")
buEnde.clicked.connect(wi.QApplication.instance().quit)
gr.addWidget(buEnde, 3, 1)

self.setWindowTitle("ComboBox")
self.setLayout(gr)
self.show()

def auswahl(self, ix):
    tx = self.cbAuswahl.currentText()
    self.lbErgebnis.setText(f"Auswahl: {tx}, Index: {ix}")

def eingabe(self, tx):
    self.lbErgebnis.setText(f"Eingabe: {tx}")

def ausgabe(self):
    tx = self.cbAuswahl.currentText()
    ix = self.cbAuswahl.currentIndex()
    self.lbErgebnis.setText(f"Ausgabe: {tx}, Index: {ix}")

anwendung = wi.QApplication(sys.argv)
fenster = Fenster()
sys.exit(anwendung.exec())

```

Listing 12.8 Datei »pyqt_combo.py«

Standardmäßig kann kein Text im `QComboBox`-Objekt eingegeben werden. Mithilfe der Methode `setEditable()` und des Parameters `True` ändern Sie das. Die Einträge werden wie beim `QListWidget`-Objekt mithilfe der Methode `addItem()` hinzugefügt.

Bei einer Textänderung wird das Signal `currentTextChanged` gesetzt. Es führt zur Methode `eingabe()`, die automatisch den aktuellen Text im `QComboBox`-Objekt erhält. Er wird ausgegeben, siehe [Abbildung 12.10](#).

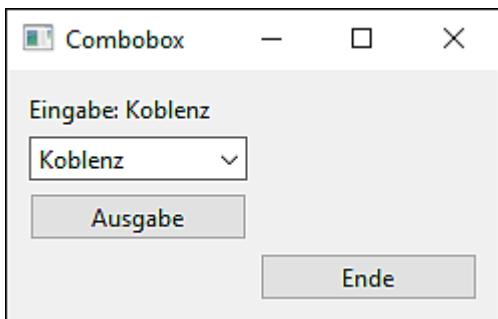


Abbildung 12.10 Eingabe eines Texts

Bei einem Wechsel zu einem anderen Eintrag wird das Signal `currentIndexChanged` gesetzt. Es führt zur Methode `auswahl()`, die automatisch den Index des aktuellen Eintrags erhält. Der Text des aktuellen Eintrags kann mithilfe der Methode `currentText()` ermittelt werden. Beide Informationen werden ausgegeben, siehe Abbildung 12.11.

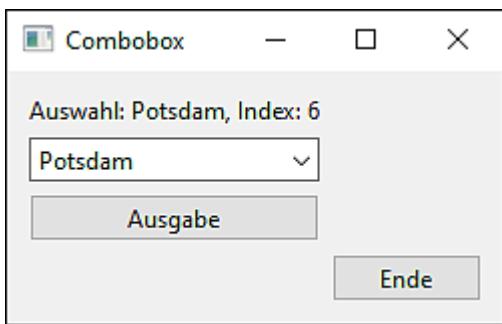


Abbildung 12.11 Auswahl eines Eintrags

Nach der Betätigung des Buttons `AUSGABE` wird die Methode `ausgabe()` aufgerufen. Die Methode `currentIndex()` liefert den Index des aktuellen Eintrags.

12.3.7 Spinbox

Eine Spinbox ermöglicht die komfortable Auswahl oder Eingabe eines Zahlenwerts aus einem begrenzten Zahlenbereich. Werte außerhalb des Zahlenbereichs können weder ausgewählt noch eingegeben werden. Bei Objekten der Klasse `QSpinBox` geht es um ganze Zahlen, bei Objekten der Klasse `QDoubleSpinBox` um Zahlen mit Nachkommastellen.

Im nachfolgenden Programm werden Objekte beider Klassen gezeigt:

```
import PyQt6.QtWidgets as wi
import sys

class Fenster(wi.QWidget):
    def __init__(self):
        super().__init__()
        gr = wi.QGridLayout()

        self.lbErgebnis = wi.QLabel("Ihre Auswahl oder Eingabe:")

        # ... (remaining code for the window class)
```

```

        self.lbErgebnis.setFixedWidth(150)
        gr.addWidget(self.lbErgebnis, 0, 0)

        self.spAuswahlInt = wi.QSpinBox()
        self.spAuswahlInt.setMinimum(10)
        self.spAuswahlInt.setMaximum(30)
        self.spAuswahlInt.setSingleStep(2)
        self.spAuswahlInt.setValue(16)
        self.spAuswahlInt.valueChanged.connect(self.aendern)
        gr.addWidget(self.spAuswahlInt, 1, 0)

        self.spAuswahlDbl = wi.QDoubleSpinBox()
        self.spAuswahlDbl.setMinimum(2.8)
        self.spAuswahlDbl.setMaximum(7.2)
        self.spAuswahlDbl.setSingleStep(0.2)
        self.spAuswahlDbl.setValue(3.4)
        self.spAuswahlDbl.setDecimals(1)
        self.spAuswahlDbl.valueChanged.connect(self.aendern)
        gr.addWidget(self.spAuswahlDbl, 2, 0)

        buAusgabe = wi.QPushButton("Ausgabe")
        buAusgabe.clicked.connect(self.ausgabe)
        gr.addWidget(buAusgabe, 3, 0)

        buEnde = wi.QPushButton("Ende")
        buEnde.clicked.connect(wi.QApplication.instance().quit)
        gr.addWidget(buEnde, 4, 1)

        self.setWindowTitle("Spinbox")
        self.setLayout(gr)
        self.show()

    def aendern(self, wert):
        if self.sender() is self.spAuswahlInt:
            self.lbErgebnis.setText(f"Auswahl: {wert}")
        else:
            self.lbErgebnis.setText(f"Auswahl: {round(wert,1)}")

    def ausgabe(self):
        wertInt = self.spAuswahlInt.value()
        wertDbl = round(self.spAuswahlDbl.value(), 1)
        self.lbErgebnis.setText(f"Ausgabe: {wertInt} und {wertDbl}")

anwendung = wi.QApplication(sys.argv)
fenster = Fenster()
sys.exit(anwendung.exec())

```

Listing 12.9 Datei »pyqt_spin.py«

Die Methoden `setMinimum()`, `setMaximum()` und `setValue()` dienen zum Setzen der Grenzen des Zahlenbereichs und des aktuellen Werts.

Mithilfe der Methode `setSingleStep()` können Sie die Schrittweite bei der Betätigung der Pfeile an der Spinbox

festlegen. Der Standardwert ist 1. Eine Spinbox kann durch eine Eingabe auch einen Wert erhalten, der zwischen zwei Schritten liegt, aber niemals einen Wert außerhalb des Zahlenbereichs.

Bei einer Spinbox für Zahlen mit Nachkommastellen wird mithilfe der Methode `setDecimals()` die Anzahl der Nachkommastellen für die Eingabe und die Darstellung in der Spinbox festgelegt.

Der Wert einer Spinbox kann durch eine Eingabe oder eine Auswahl geändert werden. Beide Aktionen setzen das Signal `valueChanged`. Hier führt es für beide Spinboxen zur Methode `aendern()`. In der Methode steht der aktuelle Wert derjenigen Spinbox, die das Signal gesetzt, also das Ereignis ausgelöst hat, automatisch zur Verfügung. Er wird ausgegeben, siehe [Abbildung 12.12](#).

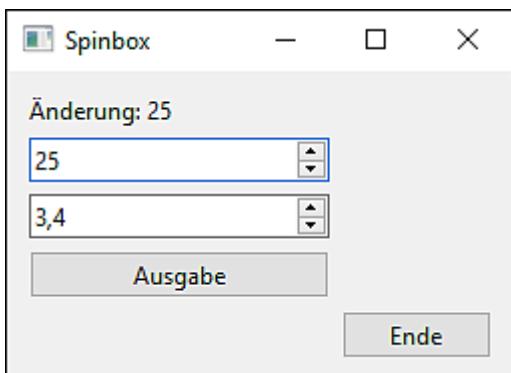


Abbildung 12.12 Wert der ersten Spinbox nach Eingabe

Die Festlegung der Anzahl der Nachkommastellen gilt nur für die Eingabe und die Darstellung in der Spinbox, nicht für den resultierenden Wert oder seine Darstellung im Ausgabefeld. Daher muss dieser Wert ebenfalls auf eine Nachkommastelle gerundet werden. Da die Methode `aendern()` auf beide Spinboxen reagiert, müssen wir den Sender des Signals ermitteln.

Die Methode `sender()` des Anwendungsfensters liefert im Fall eines Ereignisses eine Referenz auf dasjenige Objekt, das das Signal gesetzt hat. Hier wird diese Referenz mit der Referenz auf

die erste Spinbox verglichen. Auf diese Weise kann die Ausgabe passend gestaltet werden, siehe [Abbildung 12.13](#).

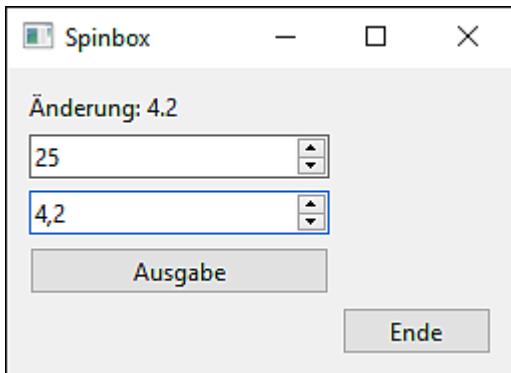


Abbildung 12.13 Wert der zweiten Spinbox nach Auswahl

Nach der Betätigung des Buttons AUSGABE wird die Methode `ausgabe()` aufgerufen. Die Methode `value()` liefert den aktuellen Wert einer Spinbox, siehe [Abbildung 12.14](#).

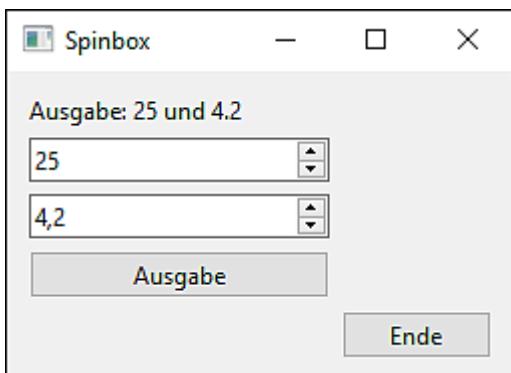


Abbildung 12.14 Werte der beiden Spinboxen

12.3.8 Radiobutton

Objekte der Klasse `QRadioButton` dienen als Radiobuttons. Sie ermöglichen eine einfache Auswahl zwischen mehreren Möglichkeiten.

Es folgt ein Programm zur Auswahl einer Farbe:

```
import PyQt6.QtWidgets as wi
import sys

class Fenster(wi.QWidget):
    def __init__(self):
        super().__init__()
        gr = wi.QGridLayout()
```

```

        self.lbAuswahl = wi.QLabel("Ihre Auswahl:")
        gr.addWidget(self.lbAuswahl, 0, 0, 1, 3)

        self.rbRot = wi.QRadioButton("Rot")
        self.rbRot.toggled.connect(self.auswahl)
        gr.addWidget(self.rbRot, 1, 0)

        self.rbGelb = wi.QRadioButton("Gelb")
        self.rbGelb.setChecked(True)
        self.rbGelb.toggled.connect(self.auswahl)
        gr.addWidget(self.rbGelb, 1, 1)

        self.rbBlau = wi.QRadioButton("Blau")
        self.rbBlau.toggled.connect(self.auswahl)
        gr.addWidget(self.rbBlau, 1, 2)

        buAusgabe = wi.QPushButton("Ausgabe")
        buAusgabe.clicked.connect(self.ausgabe)
        gr.addWidget(buAusgabe, 2, 0, 1, 3)

        buEnde = wi.QPushButton("Ende")
        buEnde.clicked.connect(wi.QApplication.instance().quit)
        gr.addWidget(buEnde, 3, 3)

        self.setWindowTitle("Radiobutton")
        self.setLayout(gr)
        self.show()

    def auswahl(self):
        rb = self.sender()
        if rb.isChecked():
            self.lbAuswahl.setText(f"Auswahl: {rb.text()}")

    def ausgabe(self):
        if self.rbRot.isChecked():
            rb = self.rbRot
        elif self.rbGelb.isChecked():
            rb = self.rbGelb
        else:
            rb = self.rbBlau
        self.lbAuswahl.setText(f"Ausgabe: {rb.text()}")

anwendung = wi.QApplication(sys.argv)
fenster = Fenster()
sys.exit(anwendung.exec())

```

Listing 12.10 Datei »pyqt_radio.py«

Das Label zur Ausgabe und der Button unter den drei Radiobuttons erstrecken sich jeweils über drei Spalten.

Mithilfe der Methode `setChecked()` sollte einer der Radiobuttons ausgewählt werden, damit es in jedem Fall eine Auswahl gibt. Das Signal `toggled` wird gesetzt, falls ein Radiobutton seinen Zustand

ändert. Das passiert sowohl bei der Markierung eines Radiobuttons als auch beim Verlust der Markierung durch die Markierung eines anderen Radiobuttons.

Hier führt das Signal `toggled` für alle Radiobuttons zur Methode `auswahl()`. Die Methode `sender()` liefert eine Referenz auf denjenigen Radiobutton, bei dem sich der Zustand geändert hat. Mithilfe der Methode `isChecked()` wird ermittelt, ob ein Radiobutton markiert wurde. In diesem Fall wird der Text des Radiobuttons angezeigt, siehe [Abbildung 12.15](#). Hat ein Radiobutton seine Markierung verloren, wird nichts ausgegeben.



Abbildung 12.15 Markierung eines Radiobuttons

Nach der Betätigung des Buttons `AUSGABE` wird die Methode `ausgabe()` aufgerufen. In einer mehrfachen Verzweigung wird die Referenz des aktuell markierten Radiobuttons ermittelt und übergeben.

12.3.9 Mehrere Gruppen von Radiobuttons

Objekte der Klasse `QButtonGroup` sind selbst nicht sichtbar, können aber zur logischen Trennung mehrerer Gruppen von Radiobuttons genutzt werden. Zugleich erleichtern sie die Ermittlung des aktuell markierten Radiobuttons innerhalb einer Gruppe. Im nachfolgenden Programm können auf diese Weise eine Farbe und eine Größe ausgewählt werden:

```
import PyQt6.QtWidgets as wi
import sys

class Fenster(wi.QWidget):
```

```

def __init__(self):
    super().__init__()
    gr = wi.QGridLayout()

    self.lbAuswahl = wi.QLabel("Ihre Auswahl:")
    gr.addWidget(self.lbAuswahl, 0, 0, 1, 3)

    rbRot = wi.QRadioButton("Rot")
    gr.addWidget(rbRot, 1, 0)

    rbGelb = wi.QRadioButton("Gelb")
    rbGelb.setChecked(True)
    gr.addWidget(rbGelb, 1, 1)

    rbBlau = wi.QRadioButton("Blau")
    gr.addWidget(rbBlau, 1, 2)

    self.gpFarbe = wi.QButtonGroup()
    self.gpFarbe.addButton(rbRot)
    self.gpFarbe.addButton(rbGelb)
    self.gpFarbe.addButton(rbBlau)
    self.gpFarbe.buttonClicked.connect(self.auswahl)

    rbKlein = wi.QRadioButton("Klein")
    gr.addWidget(rbKlein, 2, 0)

    rbMittel = wi.QRadioButton("Mittel")
    gr.addWidget(rbMittel, 2, 1)

    rbGross = wi.QRadioButton("Groß")
    rbGross.setChecked(True)
    gr.addWidget(rbGross, 2, 2)

    self.gpGroesse = wi.QButtonGroup()
    self.gpGroesse.addButton(rbKlein)
    self.gpGroesse.addButton(rbMittel)
    self.gpGroesse.addButton(rbGross)
    self.gpGroesse.buttonClicked.connect(self.auswahl)

    buAusgabe = wi.QPushButton("Ausgabe")
    buAusgabe.clicked.connect(self.ausgabe)
    gr.addWidget(buAusgabe, 3, 0, 1, 3)

    buEnde = wi.QPushButton("Ende")
    buEnde.clicked.connect(wi.QApplication.instance().quit)
    gr.addWidget(buEnde, 4, 3)

    self.setWindowTitle("Gruppe")
    self.setLayout(gr)
    self.show()

def ausgabe(self):
    rbFarbe = self.gpFarbe.checkedButton()
    rbGroesse = self.gpGroesse.checkedButton()
    self.lbAuswahl.setText(
        f"Ausgabe: {rbFarbe.text()} und {rbGroesse.text() }")

def auswahl(self, rb):

```

```

        self.lbAuswahl.setText(f"Auswahl: {rb.text()}")
    )
anwendung = wi.QApplication(sys.argv)
fenster = Fenster()
sys.exit(anwendung.exec())

```

Listing 12.11 Datei »pyqt_group.py«

Die Referenzen auf die Radiobuttons müssen keine Eigenschaften der Klasse sein, da die Informationen über die Gruppen ermittelt werden.

Die Referenz `gpFarbe` verweist auf das erste `QButtonGroup`-Objekt. Die Zugehörigkeit eines Radiobuttons zu einer Gruppe wird erreicht, indem für die Gruppe die Methode `addButton()` aufgerufen und dabei die Referenz auf den betreffenden Radiobutton übergeben wird.

Das Signal `buttonClicked` wird für eine Gruppe ausgelöst, falls die Benutzerin einen der Radiobuttons dieser Gruppe markiert. Das Signal ist mit der Methode `auswahl()` verbunden. Beim Aufruf der Methode wird automatisch die Referenz auf den markierten Radiobutton zur Verfügung gestellt. Der zugehörige Text wird ausgegeben.

Nach der Betätigung des Buttons `AUSGABE` wird die Methode `ausgabe()` aufgerufen. Die Methode `checkedButton()` liefert für jede Gruppe eine Referenz auf den markierten Radiobutton. Die Texte der markierten Buttons der beiden Gruppen werden ausgegeben, siehe [Abbildung 12.16](#).

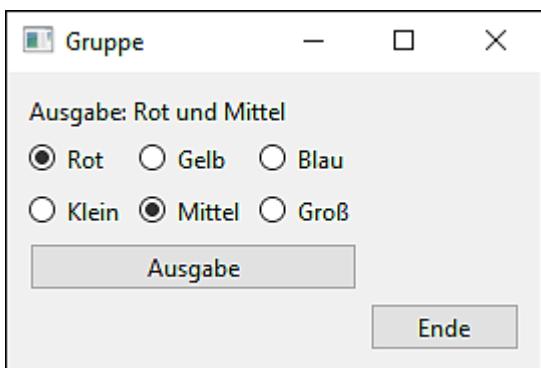


Abbildung 12.16 Mehrere Gruppen von Radiobuttons

12.3.10 Checkbox

Eine Checkbox kann ein- oder ausgeschaltet werden. Sie dient zur Darstellung eines Zustands, bei dem es zwei Möglichkeiten gibt. Kommen mehrere Checkboxen vor, sind sie voneinander unabhängig.

Im nachfolgenden Programm können die Eigenschaften eines Zeichnungsobjekts eingestellt werden: mit Rahmen oder ohne Rahmen, mit Farbe gefüllt oder nicht mit Farbe gefüllt.

```
import PyQt6.QtWidgets as wi
import sys

class Fenster(wi.QWidget):
    def __init__(self):
        super().__init__()
        gr = wi.QGridLayout()

        self.lbAuswahl = wi.QLabel("Ihre Auswahl:")
        gr.addWidget(self.lbAuswahl, 0, 0, 1, 2)

        self.cbRahmen = wi.QCheckBox("Rahmen")
        self.cbRahmen.stateChanged.connect(self.aendern)
        gr.addWidget(self.cbRahmen, 1, 0)

        self.cbFuellung = wi.QCheckBox("Füllung")
        self.cbFuellung.setChecked(True)
        self.cbFuellung.stateChanged.connect(self.aendern)
        gr.addWidget(self.cbFuellung, 1, 1)

        buAusgabe = wi.QPushButton("Ausgabe")
        buAusgabe.clicked.connect(self.ausgabe)
        gr.addWidget(buAusgabe, 2, 0, 1, 2)

        buEnde = wi.QPushButton("Ende")
        buEnde.clicked.connect(wi.QApplication.instance().quit)
        gr.addWidget(buEnde, 3, 2)

        self.setWindowTitle("Checkbox")
        self.setLayout(gr)
        self.show()

    def aendern(self):
        cb = self.sender()
        tx = cb.text()
        if cb.isChecked():
            self.lbAuswahl.setText(f"{tx}: Ja")
        else:
            self.lbAuswahl.setText(f"{tx}: Nein")

    def ausgabe(self):
        if self.cbRahmen.isChecked():
```

```

        tx = "Rahmen: Ja, "
else:
    tx = "Rahmen: Nein, "
if self.cbFuellung.isChecked():
    tx += "Füllung: Ja"
else:
    tx += "Füllung: Nein"
self.lbAuswahl.setText(tx)

anwendung = wi.QApplication(sys.argv)
fenster = Fenster()
sys.exit(anwendung.exec())

```

Listing 12.12 Datei »pyqt_check.py«

Die Methode `setChecked()` dient zum Einstellen der Checkbox per Programmcode. Mit dem Parameter `True` wird sie eingeschaltet, mit dem Parameter `False` ausgeschaltet. Das Signal `stateChanged` wird beim Wechsel des Zustands gesetzt. Hier führt das Signal für beide Objekte der Klasse `QCheckBox` zur Methode `aendern()`.

In dieser Methode wird zunächst ermittelt, welches Widget das Signal gesetzt hat. Anschließend wird mithilfe der Methode `isChecked()` geprüft, ob die Checkbox eingeschaltet ist, und es folgt eine entsprechende Ausgabe, siehe [Abbildung 12.17](#).

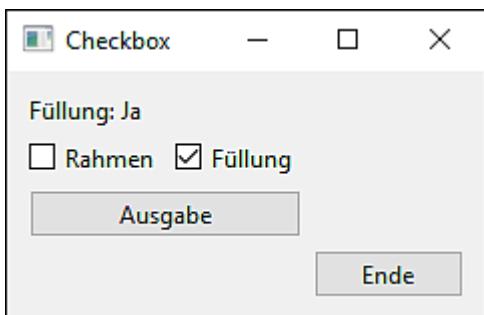


Abbildung 12.17 Nach dem Einschalten der Checkbox »Füllung«

Nach der Betätigung des Buttons `AUSGABE` wird die Methode `ausgabe()` aufgerufen. Darin wird der Zustand der beiden Checkboxen ermittelt und ausgegeben, siehe [Abbildung 12.18](#).

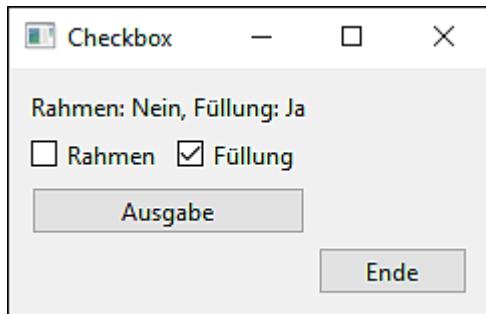


Abbildung 12.18 Ausgabe des Zustands beider Checkboxen

12.3.11 Slider

Mithilfe eines `QSlider`-Objekts können Sie einen ganzzahligen Wert innerhalb eines bestimmten Zahlenbereichs komfortabel einstellen und zugleich anschaulich darstellen. Ein Slider kann vertikal oder horizontal angeordnet werden und mithilfe der Maus, der Pfeiltasten oder der `Bild ↑ -` und der `Bild ↓ -`-Taste betätigt werden. Zur besseren Bedienung können Sie dem Slider eine Leiste mit Markierungen hinzufügen, sogenannten *Tickmarks*.

Im nachfolgenden Programm werden diese Elemente gezeigt:

```
import PyQt6.QtWidgets as wi
import PyQt6.QtCore as co
import sys

class Fenster(wi.QWidget):
    def __init__(self):
        super().__init__()
        gr = wi.QGridLayout()

        self.lbAuswahl = wi.QLabel("Ihr Wert:")
        gr.addWidget(self.lbAuswahl, 0, 0)

        self.slWert = wi.QSlider()
        self.slWert.setFixedWidth(200)
        self.slWert.setOrientation(co.Qt.Orientation.Horizontal)
        self.slWert.setMinimum(10)
        self.slWert.setMaximum(50)
        self.slWert.setValue(24)
        self.slWert.setSingleStep(2)
        self.slWert.setPageStep(10)
        self.slWert.setTickPosition(wi.QSlider.TickPosition.TicksBelow)
        self.slWert.setTickInterval(5)
        self.slWert.valueChanged.connect(self.aendern)
        gr.addWidget(self.slWert, 1, 0)
```

```

buAusgabe = wi.QPushButton("Ausgabe")
buAusgabe.clicked.connect(self.ausgabe)
gr.addWidget(buAusgabe, 2, 0)

buEnde = wi.QPushButton("Ende")
buEnde.clicked.connect(wi.QApplication.instance().quit)
gr.addWidget(buEnde, 3, 1)

self.setWindowTitle("Slider")
self.setLayout(gr)
self.show()

def andern(self, wert):
    self.lbAuswahl.setText(f"Ändern: {wert}")

def ausgabe(self):
    wert = self.slWert.value()
    self.lbAuswahl.setText(f"Ausgabe: {wert}")

anwendung = wi.QApplication(sys.argv)
fenster = Fenster()
sys.exit(anwendung.exec())

```

Listing 12.13 Datei »pyqt_slider.py«

Standardmäßig wird ein Slider vertikal angeordnet. Zur Einstellung der Anordnung rufen Sie die Methode

`setOrientation()` auf. Mithilfe des Elements `Horizontal` der Enumeration `Orientation` aus dem Modul `PyQt6.QtCore.Qt` als Parameter erfolgt eine horizontale Anordnung.

Die Methoden `setFixedWidth()`, `setMinimum()`, `setMaximum()` und `setValue()` dienen zur Einstellung einer unveränderlichen Breite, des Zahlenbereichs und des aktuellen Werts.

Standardmäßig wird der Slider mithilfe der Maus verschoben. Die Einstellung der Schrittweite für kleine Schritte, die mithilfe der Pfeiltasten ausgeführt werden, nehmen Sie mit der Methode `setSingleStep()` vor. Die Methode `setPageStep()` dient zum Einstellen der Schrittweite für große Schritte, die mithilfe der **Bild ↑**- und der **Bild ↓**-Taste oder mit einem Mausklick auf dem Slider vorgenommen werden.

Zur Anzeige der Tickmarks rufen Sie die Methode `setTickPosition()` auf. Mithilfe des Elements `TicksBelow` aus der Enumeration `TickPosition` der Klasse `QSlider` als Parameter

erfolgt die Anordnung unterhalb des Sliders. Mit der Methode `setTickInterval()` stellen Sie den Abstand der Tickmarks ein.

Die Bedienung des Sliders setzt das Signal `valueChanged`, das hier zur Methode `aendern()` führt. Diese Methode erhält automatisch den aktuellen Wert des Sliders. Dieser wird ausgegeben, siehe [Abbildung 12.19](#).

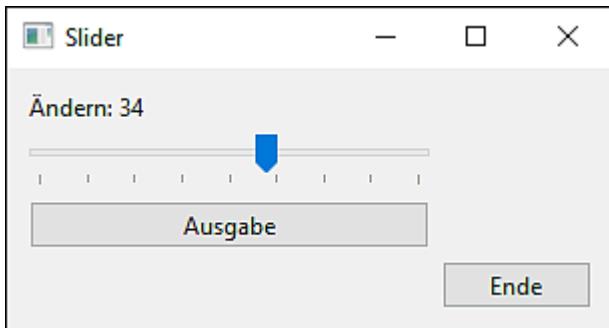


Abbildung 12.19 Nach Änderung des Werts

Nach der Betätigung des Buttons `AUSGABE` wird die Methode `ausgabe()` aufgerufen. Der aktuelle Wert des Sliders wird mithilfe der Methode `value()` ermittelt.

12.3.12 Bilder, Formate und Hyperlinks

In einem Objekt der Klasse `QLabel` kann nicht nur auf einfache Weise Text dargestellt werden, sondern er kann auch ausgerichtet werden oder einen Hyperlink bilden. Zudem kann in einem Label ein Bild angezeigt werden.

Mit bestimmten HTML- oder CSS-Angaben lassen sich die Funktion und die Darstellung eines Labels weiter gestalten. *HTML* (= Hypertext Markup Language) und *CSS* (= Cascading Style Sheets) bieten umfangreiche Möglichkeiten zur Formatierung von Internetdokumenten.

Das nachfolgende Programm zeigt einige Möglichkeiten:

```
import PyQt6.QtWidgets as wi
import PyQt6.QtGui as gu
import PyQt6.QtCore as co
import sys
```

```

class Fenster(wi.QWidget):
    def __init__(self):
        super().__init__()
        gr = wi.QGridLayout()

        lbBild = wi.QLabel()
        pm = gu.QPixmap("rheinwerk.png")
        lbBild.setPixmap(pm)
        gr.addWidget(lkBild, 0, 0)

        lbFarbe = wi.QLabel("Text")
        lbFarbe.setFixedHeight(50)
        lbFarbe.setStyleSheet(
            "background-color:#A0A0A0; border:2px solid #000000;")
        lbFarbe.setAlignment(
            co.Qt.AlignmentFlag.AlignBottom | co.Qt.AlignmentFlag.AlignRight)
        gr.addWidget(lbFarbe, 1, 0)

        lbLink = wi.QLabel(
            "<a href='https://www.rheinwerk-verlag.de/'>Rheinwerk-Verlag</a>")
        lbLink.setOpenExternalLinks(True)
        gr.addWidget(lbLink, 2, 0)

        buEnde = wi.QPushButton("Ende")
        buEnde.clicked.connect(wi.QApplication.instance().quit)
        gr.addWidget(buEnde, 3, 0)

    self.setWindowTitle("Label")
    self.setLayout(gr)
    self.show()

anwendung = wi.QApplication(sys.argv)
fenster = Fenster()
sys.exit(anwendung.exec())

```

Listing 12.14 Datei »pyqt_label.py«

Ein Objekt der Klasse `QPixmap` aus dem Modul `PyQt6.QtGui` repräsentiert ein Bild aus einer Bilddatei. Für ein `QLabel`-Objekt kann die Methode `setPixmap()` aufgerufen werden. Der Methode wird eine Referenz auf ein `QPixmap`-Objekt übergeben. Das repräsentierte Bild wird im Label dargestellt, siehe [Abbildung 12.20](#).



Abbildung 12.20 Bilder, Formate und Hyperlinks

Mithilfe der Methode `setStyleSheet()` formatieren Sie ein Label. Die CSS-Eigenschaft `background-color` dient zur Angabe der Hintergrundfarbe, die CSS-Eigenschaft `border` zur Gestaltung des Rahmens, mit Werten für die Dicke, die Linienart und die Farbe.

Die Methode `setAlignment()` wird zur Einstellung der horizontalen und vertikalen Ausrichtung des Texts innerhalb des Labels benötigt. Standardmäßig erscheint der Text linksbündig und vertikal zentriert. Mithilfe der Elemente `AlignBottom` und `AlignRight` der Enumeration `AlignmentFlag` aus dem Modul `QtCore` erscheint der Text rechtsbündig unten. Bei den Elementen handelt es sich um Bitflaggen, die mithilfe des Operators für das *Bitweise Oder* `|` miteinander verknüpft werden.

Der Text in einem Label kann auch mithilfe von bestimmten HTML-Markierungen formatiert werden. Hier wird ein HTML-Container zur Erstellung eines Hyperlinks eingesetzt. Zusätzlich muss die Methode `setOpenExternalLinks()` mit dem Parameter `True` aufgerufen werden.

Anhang A

Hier im Anhang finden Sie eine Anleitung zum Erstellen von ausführbaren EXE-Dateien, eine Beschreibung zur Installation des Pakets XAMPP für die verschiedenen Betriebssysteme und einige wichtige UNIX-Befehle.

A.1 Paketverwaltungsprogramm »pip«

Das Paketverwaltungsprogramm *pip* dient zur Installation zusätzlicher Module. Unter Windows wurde *pip* bereits gemeinsam mit Python installiert. Unter Ubuntu Linux wird es mit folgendem Aufruf installiert:

```
sudo apt install python3-pip
```

Unter macOS kann es zunächst heruntergeladen und anschließend installiert werden. Dazu sind die beiden folgenden Aufrufe notwendig:

```
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py  
python3 get-pip.py
```

Zur Nutzung von *pip* öffnen Sie unter Windows eine Eingabeaufforderung (siehe [Abschnitt 2.3.2](#), »Ausführen unter Windows«) und wechseln mit den entsprechenden `cd`-Befehlen in das Unterverzeichnis *Scripts*. Unter Ubuntu Linux oder macOS öffnen Sie ein Terminal (siehe [Abschnitt 2.3.3](#), »Ausführen unter Ubuntu Linux und unter macOS«). Führen Sie dann die Installation eines Zusatzmoduls, hier zum Beispiel `matplotlib`, wie folgt durch:

```
pip install matplotlib
```

A.2 Erstellen von EXE-Dateien

Mithilfe des Programms `pyinstaller` können Sie aus jedem Ihrer Python-Programme eine ausführbare Datei erstellen. Das gilt auch für Programme mit Tk-Benutzeroberflächen.

Die ausführbare Datei kann auf einem anderen Rechner mit demselben Betriebssystem gestartet werden. Es ist nicht notwendig, zuvor Python auf dem anderen Rechner zu installieren. Es folgt eine Anleitung zur Erstellung einer ausführbaren Version des Programms `hallo.py`:

- Installieren Sie zunächst das Programm `pyinstaller` mit dem Paketverwaltungsprogramm `pip` (siehe [Abschnitt A.1](#)) mit dem Aufruf: `pip install pyinstaller`.
- Unter Windows: Kopieren Sie die Datei `hallo.py` in das Unterverzeichnis `Scripts` Ihrer Python-Installation (also in `C:\Python\Scripts`), öffnen Sie eine Eingabeaufforderung (siehe [Abschnitt 2.3.2](#), »Ausführen unter Windows«), und wechseln Sie mit den entsprechenden `cd`-Befehlen in das Unterverzeichnis `Scripts`.
- Unter Ubuntu Linux oder macOS: Kopieren Sie die Datei `hallo.py` in ein gewünschtes Verzeichnis, öffnen Sie ein Terminal (siehe [Abschnitt 2.3.3](#), »Ausführen unter Ubuntu Linux und unter macOS«), und wechseln Sie in das gewählte Verzeichnis.
- Rufen Sie das Programm `pyinstaller` mit Ihrem Python-Programm auf, hier also mit: `pyinstaller hallo.py`.
- Im Unterverzeichnis `dist` finden Sie anschließend das Unterverzeichnis `hallo` inklusive der ausführbaren Datei `hallo.exe` bzw. `hallo`.

- Kopieren Sie dieses Verzeichnis vollständig auf den Zielrechner, und rufen Sie dort die ausführbare Datei *hallo.exe* bzw. *hallo* von der Kommandozeile aus auf.

Sollten Sie die ausführbare Datei nicht von der Kommandozeile aus aufrufen, startet das Programm ebenfalls, ist aber bekanntlich schnell wieder zu Ende. Das Ausgabefenster würde nur stehen bleiben, falls am Ende des Programms eine Eingabe verlangt wird, zum Beispiel mit: `input()`.

A.3 Installation von XAMPP

Es wird die Installation des frei verfügbaren, vorkonfigurierten Pakets XAMPP beschrieben, das Sie für Ihre Programmierung mithilfe von PHP und MySQL (bzw. dessen Abspaltung MariaDB) unter Windows, Ubuntu Linux oder macOS einsetzen können. Das Paket umfasst neben dem Datenbankserver MySQL unter anderem einen Apache-Webserver, die Sprache PHP und die Datenbank-Benutzeroberfläche phpMyAdmin.

Laden Sie das für Ihr Betriebssystem passende Paket XAMPP über die Website <https://www.apachefriends.org> herunter, und wechseln Sie in das Download-Verzeichnis.

A.3.1 Installation von XAMPP unter Windows

Das Paket XAMPP für Windows finden Sie in der ausführbaren Datei *xampp-windows-x64-8.0.13-O-VS16-installer*. Durch den Aufruf der Datei beginnen Sie mit der Installation. In manchen Fällen erscheinen zu Beginn zwei Warnungen, unter anderem bezüglich eines laufenden Antivirenprogramms. Hier können Sie fortfahren.

Sie können die vorgeschlagenen Installationsoptionen bestätigen. Beim Zielverzeichnis empfehle ich, *C:\xampp* zu wählen. Starten Sie nach der Installation die Anwendung **XAMPP CONTROL PANEL**. Darin starten und stoppen Sie den Apache-Webserver über den Button rechts neben dem Begriff **APACHE**. Den MariaDB-Datenbankserver starten und stoppen Sie über den Button rechts neben dem Begriff **MySQL**.

Nachdem der Webserver gestartet ist, können Sie HTML-Dateien und PHP-Programme, die Sie im Verzeichnis *C:\xampp\htdocs* ablegen, über den Webserver in Ihrem Browser über die Adresse

http://localhost erreichen. Nachdem der Datenbankserver gestartet ist, können Sie Ihre Datenbanken in der Benutzeroberfläche phpMyAdmin bearbeiten, die Sie im Browser über die Adresse *http://localhost/phpmyadmin* erreichen.

A.3.2 Installation von XAMPP unter Ubuntu Linux

Das Paket XAMPP für Ubuntu Linux finden Sie in der Datei *xampp-linux-x64-8.0.13-0-installer.run*. Öffnen Sie zur Eingabe ein Terminal. Ändern Sie gegebenenfalls die Zugriffsrechte auf die Datei mit dem Befehl:

```
chmod 744 xampp-linux-x64-8.0.13-0-installer.run
```

Starten Sie die Installation mit:

```
sudo ./xampp-linux-x64-8.0.13-0-installer.run
```

Sie können die vorgeschlagenen Installationsoptionen bestätigen. XAMPP wird im Verzeichnis */opt/lampp* installiert. Am Ende der Installation können Sie das Häkchen bei **LAUNCH XAMPP** stehen lassen. Damit wird ein Dialogfeld zum Verwalten der Server aufgerufen. Auf der Registerkarte **MANAGE SERVERS** haben Sie die Möglichkeit, den Apache-Webserver und den MariaDB-Datenbankserver (über MySQL) auszuwählen und rechts über den jeweiligen Button zu starten und zu stoppen. Das Dialogfeld zum Verwalten der Server können Sie auch direkt wie folgt aufrufen:

```
sudo /opt/lampp/manager-linux-x64.run
```

Sie erreichen die Startseite des lokalen Webservers in Ihrem Browser über die Adresse *http://localhost*. Die Benutzeroberfläche phpMyAdmin rufen Sie über die Adresse *http://localhost/phpmyadmin* auf. Ihre HTML-Dateien und PHP-Programme können Sie im Verzeichnis */opt/lampp/htdocs* und in den darunterliegenden Verzeichnissen speichern.

A.3.3 Installation von XAMPP unter macOS

Das Paket XAMPP für macOS finden Sie in der Installationsdatei *xampp-osx-8.0.13-0-installer.dmg*. Durch einen Doppelklick auf diese Datei wird ein neues Laufwerk angelegt. Nun können Sie die Installationsdatei auf diesem Laufwerk aufrufen. Die vorgeschlagenen Installationsoptionen können Sie bestätigen. XAMPP wird im Verzeichnis *Applications/XAMPP* installiert, was im Finder dem Verzeichnis *Programme/XAMPP* entspricht.

Am Ende der Installation können Sie das Häkchen bei **LAUNCH XAMPP** stehen lassen. Damit wird ein Dialogfeld zum Verwalten der Server aufgerufen. Auf der Registerkarte **MANAGE SERVERS** haben Sie die Möglichkeit, den Apache-Webserver und den MariaDB-Datenbankserver (über MySQL) auszuwählen und rechts über den jeweiligen Button zu starten und zu stoppen. Das Dialogfeld zum Verwalten der Server können Sie auch über *Programme/XAMPP/manager-osx* aufrufen.

Sie können die Startseite des lokalen Webservers in Ihrem Browser über die Adresse *http://localhost* erreichen. Die Benutzeroberfläche phpMyAdmin rufen Sie über die Adresse *http://localhost/phpmyadmin* auf. Ihre HTML-Dateien und PHP-Programme können Sie im Verzeichnis *Programme/XAMPP/htdocs* und in den darunterliegenden Verzeichnissen speichern.

A.4 UNIX-Befehle

Zur Verwaltung von Verzeichnissen und Dateien unter dem Betriebssystem UNIX oder einem seiner Abkömmlinge, wie zum Beispiel Ubuntu Linux oder macOS, können Sie mit Kommandozeilenbefehlen arbeiten. Diese können in einem Terminal unter den genannten Betriebssystemen eingegeben werden.

In diesem Abschnitt lernen Sie die nützlichen Befehle `ls`, `mkdir`, `rmdir`, `cd`, `cp`, `mv` und `rm` kennen. Achten Sie auf den Unterschied bei der Groß- und Kleinschreibung. Ein Beispiel: Es kann zwei verschiedene Dateien mit den Namen `hallo.txt` und `Hallo.txt` geben.

A.4.1 Inhalt eines Verzeichnisses

UNIX-Systeme verfügen wie Windows-Systeme über eine Hierarchie von Verzeichnissen. Es gibt also ein Hauptverzeichnis, darunter kann es Unterverzeichnisse geben, die wiederum Unterverzeichnisse haben können.

Mit .. (zwei Punkten) wird immer das jeweils übergeordnete Verzeichnis angesprochen, mit . (einem einzelnen Punkt) das aktuelle Verzeichnis. Diese Bezeichnungen kommen auch bei einigen Befehlen zum Einsatz.

Mithilfe des Befehls `ls -l` können Sie sich eine Liste der Dateien und Unterverzeichnisse des aktuellen Verzeichnisses in ausführlicher Form anzeigen lassen. Dies kann eine nützliche Kontrolle nach jeder Veränderung sein. Eine Beispielausgabe:

```
-rw-r--r-- 1 theis theis    12 Dez 3 08:52 hallo.txt
-rw-r--r-- 1 theis theis    12 Dez 3 08:51 gruss.txt
drwxr-xr-x 2 theis theis 4096 Dez 3 08:57 haus
```

Sie sehen zwei Dateien mit der Endung `.txt` und ein Unterverzeichnis. Die wichtigsten Informationen dazu:

- Im Unterschied zu Windows sind die Rechte klar unterteilt. Nicht jeder Benutzer darf alles. Steht in der ersten Spalte ein `d` (für *directory*), so handelt es sich um ein Unterverzeichnis.
- Anschließend folgen die Rechte bezüglich des Eintrags. Dabei steht `r` (für *read*) für das Leserecht, `w` (für *write*) für das Schreibrecht und `x` (für *execute*) für das Recht, ein Programm auszuführen.
- Diese Rechte werden dreimal nacheinander aufgelistet: zuerst für den aktuellen Benutzer, anschließend für die Arbeitsgruppe des aktuellen Benutzers und zuletzt für alle Benutzer des Systems.
- Die Größe der Dateien wird angezeigt. In diesem Fall sind es jeweils 12 Byte für die beiden Textdateien. Außerdem sehen Sie Datum und Uhrzeit der letzten Änderung.

A.4.2 Verzeichnis anlegen, wechseln und löschen

Der Befehl `mkdir` (*make directory*) dient zum Anlegen eines neuen Verzeichnisses unterhalb des aktuellen Verzeichnisses. Ein Beispiel:

- `mkdir meineTexte` – Anlegen des Unterverzeichnisses `meineTexte`, relativ zum aktuellen Verzeichnis (der Gedankenstrich gehört nicht zum Befehl)

Mit dem Befehl `cd` (*change directory*) wechseln Sie das Verzeichnis. Einige Beispiele:

- `cd meineTexte` – Wechsel in das Unterverzeichnis `meineTexte`, relativ zum aktuellen Verzeichnis
- `cd ..` – Wechsel in das übergeordnete Verzeichnis

- `cd` (ohne weitere Angaben) – Wechsel in Ihr Heimatverzeichnis, unabhängig vom aktuellen Verzeichnis
- `cd /usr/bin` – Wechsel in das absolute Verzeichnis *usr/bin*, unabhängig vom aktuellen Verzeichnis

Der Befehl `rmdir` (*remove directory*) dient zum Löschen eines leeren Unterverzeichnisses, das sich unterhalb des aktuellen Verzeichnisses befindet. Zum Löschen von Dateien aus einem Verzeichnis verweise ich auf den nächsten Abschnitt. Ein Beispiel:

- `rmdir meineTexte` – Löschen des Unterverzeichnisses *meineTexte*, falls es leer ist

A.4.3 Datei kopieren, verschieben und löschen

Textdateien können Sie mit einem Editor anlegen, zum Beispiel mit *gedit*, *vi* oder *nano*. Python-Programme erstellen Sie am besten innerhalb der IDLE-Umgebung.

Zum Kopieren von Dateien nutzen Sie den Befehl `cp` (*copy*). Es werden immer zwei Angaben benötigt, zum einen für die Quelle und zum anderen für das Ziel des Kopiervorgangs. Einige Beispiele:

- `cp hallo.txt gruss.txt` – Kopieren der Datei *hallo.txt* in die Datei *gruss.txt* innerhalb des aktuellen Verzeichnisses
- `cp hallo.txt ..` – Kopieren der Datei *hallo.txt* in das übergeordnete Verzeichnis
- `cp hallo.txt ../nochMehrTexte` – Kopieren der Datei *hallo.txt* in das Verzeichnis *nochMehrTexte*, das sich unter demselben übergeordneten Verzeichnis wie das aktuelle Verzeichnis befindet
- `cp ../hallo.txt .` – Kopieren der Datei *hallo.txt* aus dem übergeordneten Verzeichnis in das aktuelle Verzeichnis (beachten Sie den einzelnen Punkt am Ende des Befehls)

- `cp ../*.txt .` – Kopieren aller Dateien mit der Endung `.txt` aus dem übergeordneten Verzeichnis in das aktuelle Verzeichnis
- `cp ../nochMehrTexte/hallo.txt .` – Kopieren der Datei `hallo.txt` aus dem Verzeichnis `nochMehrTexte` (siehe oben) in das aktuelle Verzeichnis

Mithilfe des Befehls `mv` (*move*) können Sie Dateien umbenennen bzw. verschieben. Die Benutzung ist der von `cp` sehr ähnlich.

Einige Beispiele:

- `mv hallo.txt gruss.txt` – Umbenennen der Datei `hallo.txt` in die Datei `gruss.txt` innerhalb des aktuellen Verzeichnisses
- `mv hallo.txt ..` – Verschieben der Datei `hallo.txt` in das übergeordnete Verzeichnis
- `mv ../hallo.txt .` – Verschieben der Datei `hallo.txt` aus dem übergeordneten Verzeichnis in das aktuelle Verzeichnis
- `mv ../*.txt .` – Verschieben aller Dateien mit der Endung `.txt` aus dem übergeordneten Verzeichnis in das aktuelle Verzeichnis

Zum Löschen von Dateien nutzen Sie den Befehl `rm` (*remove*).

Einige Beispiele:

- `rm hallo.txt` – Löschen der Datei `hallo.txt` innerhalb des aktuellen Verzeichnisses
- `rm *.txt` – Löschen aller Dateien mit der Endung `.txt` innerhalb des aktuellen Verzeichnisses

Stichwortverzeichnis

↓**A** ↓**B** ↓**C** ↓**D** ↓**E** ↓**F** ↓**G** ↓**H** ↓**I** ↓**J** ↓**K** ↓**L** ↓**M** ↓**N** ↓**O** ↓**P** ↓**Q**
↓**R** ↓**S** ↓**T** ↓**U** ↓**V** ↓**W** ↓**X** ↓**Y** ↓**Z**

-= (Zuweisung) [→ 5.1 Allgemeines]

-> (Typhinweis) [→ 3.7 Funktionen und Module]

^ (Bitoperator) [→ 4.1 Zahlen]

^ (Set) [→ 4.6 Sets, Mengen]

_ (match) [→ 3.3 Verzweigungen]

__add__() [→ 6.4 Operatormethoden]

__del__() [→ 6.3 Besondere Member]

__dict__ [→ 6.3 Besondere Member] [→ 8.6 Datenaustausch mit JSON]

__eq__() [→ 6.4 Operatormethoden]

__floordiv__() [→ 6.4 Operatormethoden]

__ge__() [→ 6.4 Operatormethoden]

__gt__() [→ 6.4 Operatormethoden]

__init__() [→ 6.3 Besondere Member]

__le__() [→ 6.4 Operatormethoden]

__lt__() [→ 6.4 Operatormethoden]

__mod__() [→ 6.4 Operatormethoden]

__mul__() [→ 6.4 Operatormethoden]

__ne__() [→ 6.4 Operatormethoden]

`__pow__()` [→ 6.4 Operatormethoden]
`__str__()` [→ 6.3 Besondere Member]
`__sub__()` [→ 6.4 Operatormethoden]
`__truediv__()` [→ 6.4 Operatormethoden]
`!=` (ungleich) [→ 3.3 Verzweigungen] [→ 4.5 Dictionarys]
(neue Zeile) [→ 8.3 Textdateien]
[] (Dictionary) [→ 4.5 Dictionarys]
[] (Liste) [→ 4.3 Listen]
[] (Slice) [→ 4.2 Zeichenketten]
{} (Dictionary) [→ 4.5 Dictionarys]
`@` (Skalarprodukt) [→ 5.8 Weitere mathematische Module]
`@dataclass` [→ 6.8 Datenklassen]
`*` (deque) [→ 7.2 Warteschlangen]
`*` (mehrfache Zuweisung) [→ 4.4 Tupel]
`*` (Multiplikation) [→ 2.1 Python als Taschenrechner]
`*` (Parameter) [→ 5.6 Funktionen]
`*` (Vervielfachung) [→ 4.2 Zeichenketten] [→ 4.3 Listen]
`**` (Exponential) [→ 4.1 Zahlen]
`**=` (Zuweisung) [→ 5.1 Allgemeines]
`*=` (Zuweisung) [→ 5.1 Allgemeines]
`/` (Division) [→ 2.1 Python als Taschenrechner]
`//` (Ganzzahldivision) [→ 2.1 Python als Taschenrechner]
`//=` (Zuweisung) [→ 5.1 Allgemeines]
`/=` (Zuweisung) [→ 5.1 Allgemeines]
`&` (Bitoperator) [→ 4.1 Zahlen]

& (Set) [→ 4.6 Sets, Mengen]
(Kommentar) [→ 2.3 Speichern und Ausführen]
% (Format) [→ 5.2 Ausgabe und Formatierung]
% (Modulo) [→ 2.1 Python als Taschenrechner]
%= (Zuweisung) [→ 5.1 Allgemeines]
+ (Addition) [→ 2.1 Python als Taschenrechner]
+ (deque) [→ 7.2 Warteschlangen]
+ (Verkettung) [→ 4.2 Zeichenketten] [→ 4.3 Listen]
+= (Zuweisung) [→ 5.1 Allgemeines]
< (Format) [→ 5.2 Ausgabe und Formatierung]
< (kleiner) [→ 3.3 Verzweigungen] [→ 10.2 SQLite]
<< (Bitoperator) [→ 4.1 Zahlen]
<> (ungleich) [→ 10.2 SQLite]
= (gleich) [→ 10.2 SQLite]
= (Set) [→ 4.6 Sets, Mengen]
= (Subtraktion) [→ 2.1 Python als Taschenrechner]
= (Zuweisung) [→ 2.1 Python als Taschenrechner] [→ 3.2 Variablen und Operatoren]
== (gleich) [→ 3.3 Verzweigungen] [→ 4.5 Dictionarys]
== (Objekte) [→ 4.8 Referenz, Identität und Kopie]
> (Format) [→ 5.2 Ausgabe und Formatierung]
> (größer) [→ 3.3 Verzweigungen] [→ 10.2 SQLite]
>> (Bitoperator) [→ 4.1 Zahlen]
| (Bitoperator) [→ 4.1 Zahlen]
| (match) [→ 3.3 Verzweigungen]

| (Set) [→ 4.6 Sets, Mengen]
Ob (Präfix) [→ 4.1 Zahlen]
Oo (Präfix) [→ 4.1 Zahlen]
Ox (Präfix) [→ 4.1 Zahlen]
. (Verzeichnis) [→ A.4 UNIX-Befehle]
. (Verzeichnis) [→ A.4 UNIX-Befehle]
:= (kombinierte Zuweisung) [→ 3.4 Schleifen]
\ (Anweisung zerlegen) [→ 5.1 Allgemeines]
~ (Bitoperator) [→ 4.1 Zahlen]

A ↑

abs() [→ 4.1 Zahlen]
Abtastrate [→ 5.8 Weitere mathematische Module]
Abweichung [→ 4.1 Zahlen]
acos() [→ 4.1 Zahlen]
add_cascade() [→ 11.5 Menüs, Messageboxen und Dialogfelder]
add_checkbutton() [→ 11.5 Menüs, Messageboxen und Dialogfelder]
add_command() [→ 11.5 Menüs, Messageboxen und Dialogfelder]
add_radiobutton() [→ 11.5 Menüs, Messageboxen und Dialogfelder]
add_separator() [→ 11.5 Menüs, Messageboxen und Dialogfelder]
add() [→ 4.6 Sets, Mengen]
addButton() [→ 12.3 Widget-Typen]

addItem() [→ 12.3 Widget-Typen]
addWidget() [→ 12.2 Layout und Größe eines Anwendungsfensters]
after() [→ 11.6 Zeichnungen und Animationen]
AlignmentFlag [→ 12.3 Widget-Typen]
anchor [→ 11.1 Einführung]
AND [→ 10.2 SQLite]
and [→ 3.3 Verzweigungen]
Anonyme Funktion [→ 5.6 Funktionen]
ANSI-Kodierung [→ 11.2 Widget-Typen]
Anweisung
 leere [→ 5.1 Allgemeines]
 zerlegen [→ 5.1 Allgemeines]
 zusammensetzen [→ 5.1 Allgemeines]
Anwendungsfenster [→ 12.1 Ein erstes Programm]
 Änderung der Größe [→ 11.4 Geometrie-Manager »place«]
 Endlosschleife [→ 11.1 Einführung]
 Größe [→ 11.1 Einführung] [→ 12.2 Layout und Größe eines Anwendungsfensters]
 Größe begrenzen [→ 11.4 Geometrie-Manager »place«]
 Größe und Position [→ 11.4 Geometrie-Manager »place«]
 öffnen [→ 11.1 Einführung]
 schließen [→ 11.1 Einführung]
 Titel [→ 11.1 Einführung]

unveränderliche Größe [→ 11.1 Einführung]

vorgefertigtes [→ 11.5 Menüs, Messageboxen und Dialogfelder]

zusätzliches [→ 11.5 Menüs, Messageboxen und Dialogfelder]

Anzahl

Dictionary [→ 4.5 Dictionarys]

Liste [→ 4.3 Listen]

Liste, mehrdimensional [→ 4.3 Listen]

Set [→ 4.6 Sets, Mengen]

Zeichenkette [→ 4.2 Zeichenketten]

Apache-Webserver [→ 9.1 Laden und Senden von Internetdaten] [→ A.3 Installation von XAMPP]

append()

deque [→ 7.2 Warteschlangen]

Liste [→ 4.3 Listen]

appendleft() [→ 7.2 Warteschlangen]

argv [→ 5.10 Parameter der Kommandozeile]

Arkusfunktion [→ 4.1 Zahlen]

Array [→ 4.3 Listen]

assoziativer [→ 4.5 Dictionarys]

eindimensionaler [→ 5.8 Weitere mathematische Module]

mehrdimensionaler [→ 5.8 Weitere mathematische Module]

array() [→ 5.8 Weitere mathematische Module]

`arrow` [→ 11.6 Zeichnungen und Animationen]

as

`Aliasname` [→ 5.8 Weitere mathematische Module]
[→ 5.9 Eigene Module]

`Exception` [→ 5.5 Fehler und Ausnahmen]

`ASC` [→ 10.2 SQLite]

`ascii_letters` [→ 4.2 Zeichenketten]

`asin()` [→ 4.1 Zahlen]

`askokcancel()` [→ 11.5 Menüs, Messageboxen und Dialogfelder]

`askretrycancel()` [→ 11.5 Menüs, Messageboxen und Dialogfelder]

`askyesno()` [→ 11.5 Menüs, Messageboxen und Dialogfelder]

`atan()` [→ 4.1 Zahlen]

`Audioausgabe` [→ 7.5 Audioausgabe]

`Ausdruck zusammensetzen` [→ 5.1 Allgemeines]

`Ausführbare Datei` [→ A.2 Erstellen von EXE-Dateien]

`Ausnahme` [→ 3.6 Fehler und Ausnahmen]

`Thread` [→ 7.3 Multithreading]

`Ausnahmebehandlung` [→ 3.6 Fehler und Ausnahmen]

`Auswahlmenü`

`einfaches` [→ 11.2 Widget-Typen]

`mehrfaches` [→ 11.2 Widget-Typen]

`axhline()` [→ 5.8 Weitere mathematische Module]

`axis()` [→ 5.8 Weitere mathematische Module]

`axvline()` [→ 5.8 Weitere mathematische Module]

B ↑

b (Format) [→ 5.2 Ausgabe und Formatierung]

b (Präfix) [→ 4.2 Zeichenketten]

Bedingter Ausdruck [→ 3.3 Verzweigungen]

Beep() [→ 7.5 Audioausgabe]

Benannter Parameter [→ 5.6 Funktionen] [→ 6.3 Besondere Member]

Benutzeroberfläche [→ 11.1 Einführung] [→ 12.1 Ein erstes Programm]

Bereich

range() [→ 3.4 Schleifen]

Slice [→ 4.2 Zeichenketten]

Slice entfernen [→ 4.3 Listen]

Betrag [→ 4.1 Zahlen]

komplexe Zahl [→ 4.1 Zahlen]

bg [→ 11.5 Menüs, Messageboxen und Dialogfelder]

Bild

Darstellung [→ 11.3 Bilder und Mausereignisse] [→ 12.3 Widget-Typen]

Farbe ändern [→ 11.3 Bilder und Mausereignisse]

Farbe ermitteln [→ 11.3 Bilder und Mausereignisse]

Größe [→ 11.3 Bilder und Mausereignisse]

Transparenz ändern [→ 11.3 Bilder und Mausereignisse]

Transparenz ermitteln [→ 11.3 Bilder und Mausereignisse]

bin() [→ 4.1 Zahlen]

`bind_all()` [→ 11.6 Zeichnungen und Animationen]

`bind()` [→ 11.3 Bilder und Mausereignisse]

`Bit` [→ 4.1 Zahlen]

schieben [→ 4.1 Zahlen]

setzen [→ 4.1 Zahlen]

`Bitoperator` [→ 4.1 Zahlen]

`Bogenmaß` [→ 4.1 Zahlen]

`bool` [→ 3.3 Verzweigungen] [→ 4.7 Wahrheitswerte und Nichts]

`bool()` [→ 4.7 Wahrheitswerte und Nichts]

`break` [→ 3.4 Schleifen]

`Breakpoint` [→ 5.5 Fehler und Ausnahmen]

`Browser aufrufen` [→ 9.3 Browser aufrufen]

`Bruch` [→ 4.1 Zahlen]

`Bruchtraining` [→ 5.11 Programm »Bruchtraining«]

`Buchstaben` [→ 4.2 Zeichenketten]

`Button` [→ 11.1 Einführung] [→ 12.1 Ein erstes Programm]

`buttonClicked` [→ 12.3 Widget-Typen]

`Byte` [→ 4.1 Zahlen]

`Byte-Literal` [→ 4.2 Zeichenketten] [→ 9.1 Laden und Senden von Internetdaten]

`bytes` [→ 4.2 Zeichenketten]

decode() [→ 4.2 Zeichenketten]

bytes() [→ 4.2 Zeichenketten]

Callback-Funktion [→ 5.6 Funktionen]

Widget [→ 11.1 Einführung]

Canvas [→ 11.6 Zeichnungen und Animationen]

case [→ 3.3 Verzweigungen]

cd [→ A.4 UNIX-Befehle]

cgi

FieldStorage [→ 9.2 Webserver-Programmierung]

Modul [→ 9.2 Webserver-Programmierung]

cgi-bin [→ 9.2 Webserver-Programmierung]

CGI-Skript [→ 9.2 Webserver-Programmierung]

Standardverzeichnis [→ 9.2 Webserver-Programmierung]

cgitb

enable() [→ 9.2 Webserver-Programmierung]

Modul [→ 9.2 Webserver-Programmierung]

char [→ 11.6 Zeichnungen und Animationen]

Checkbox [→ 12.3 Widget-Typen]

checkbox (HTML) [→ 9.2 Webserver-Programmierung]

Checkbutton [→ 11.2 Widget-Typen]

in Menü [→ 11.5 Menüs, Messageboxen und Dialogfelder]

checked (HTML) [→ 9.2 Webserver-Programmierung]

checkedButton() [→ 12.3 Widget-Typen]

choice()

random [→ 4.3 Listen]

secrets [→ 5.4 Verschlüsselung]

`chr()` [→ 5.7 Eingebaute Funktionen]

`class` [→ 6.2 Klassen, Objekte und eigene Methoden]

`clear()`

deque [→ 7.2 Warteschlangen]

Set [→ 4.6 Sets, Mengen]

`clicked` [→ 12.1 Ein erstes Programm]

`close()`

connection [→ 10.2 SQLite] [→ 10.4 MySQL]

cursor [→ 10.4 MySQL]

Datei [→ 8.2 Öffnen und Schließen einer Datei]

scandir() [→ 8.7 Bearbeitung mehrerer Dateien]

`collections`

deque() [→ 7.2 Warteschlangen]

Modul [→ 7.2 Warteschlangen]

`columnspan` [→ 11.2 Widget-Typen]

`Combobox` [→ 11.2 Widget-Typen] [→ 12.3 Widget-Typen]

`command` [→ 11.1 Einführung]

`commit()` [→ 10.2 SQLite] [→ 10.4 MySQL]

`conjugate()` [→ 4.1 Zahlen]

`connect()`

mysql [→ 10.4 MySQL]

Qt [→ 12.1 Ein erstes Programm]

sqlite3 [→ 10.2 SQLite]

`connection`

close() [→ 10.2 SQLite] [→ 10.4 MySQL]

commit() [→ 10.2 SQLite] [→ 10.4 MySQL]

cursor() [→ 10.2 SQLite] [→ 10.4 MySQL]

mysql [→ 10.4 MySQL]

sqlite3 [→ 10.2 SQLite]

Connector/Python [→ 10.4 MySQL]

continue [→ 3.4 Schleifen]

coords() [→ 11.6 Zeichnungen und Animationen]

copy

deepcopy() [→ 4.8 Referenz, Identität und Kopie] [→ 6.5 Referenz, Identität und Kopie]

Modul [→ 4.8 Referenz, Identität und Kopie] [→ 6.5 Referenz, Identität und Kopie]

copy()

Datei [→ 8.9 Dateien und Verzeichnisse verwalten]

deque [→ 7.2 Warteschlangen]

Set [→ 4.6 Sets, Mengen]

cos() [→ 4.1 Zahlen]

count()

Liste [→ 4.3 Listen]

Liste, Qt [→ 12.3 Widget-Typen]

str [→ 4.2 Zeichenketten]

cp [→ A.4 UNIX-Befehle]

CREATE TABLE [→ 10.2 SQLite]

create_line() [→ 11.6 Zeichnungen und Animationen]

`create_oval()` [→ 11.6 Zeichnungen und Animationen]
`create_polygon()` [→ 11.6 Zeichnungen und Animationen]
`create_rectangle()` [→ 11.6 Zeichnungen und Animationen]
`create_text()` [→ 11.6 Zeichnungen und Animationen]
`cross()` [→ 5.8 Weitere mathematische Module]
CSV-Datei [→ 8.3 Textdateien]
`currentIndex()` [→ 12.3 Widget-Typen]
`currentIndexChanged` [→ 12.3 Widget-Typen]
`currentItem()` [→ 12.3 Widget-Typen]
`currentText()` [→ 12.3 Widget-Typen]
`currentTextChanged` [→ 12.3 Widget-Typen]
`curselection()` [→ 11.2 Widget-Typen]
`cursor`
 `close()` [→ 10.4 MySQL]
 `execute()` [→ 10.4 MySQL]
 `fetchall()` [→ 10.4 MySQL]
`cursor()` [→ 10.2 SQLite] [→ 10.4 MySQL]

D ↑

`d` (Format) [→ 5.2 Ausgabe und Formatierung]
`dataclasses` [→ 6.8 Datenklassen]
`DATE` [→ 10.4 MySQL]
`Datei`
 `anlegen` [→ A.4 UNIX-Befehle]
 `Eigenschaften` [→ 8.8 Informationen über Dateien]

Existenz prüfen [→ 8.9 Dateien und Verzeichnisse verwalten]

feste Struktur [→ 8.4 Dateien mit festgelegter Struktur]

Größe [→ 8.8 Informationen über Dateien]

kopieren [→ 8.9 Dateien und Verzeichnisse verwalten]
[→ A.4 UNIX-Befehle]

Liste erstellen [→ 8.7 Bearbeitung mehrerer Dateien]

löschen [→ 8.9 Dateien und Verzeichnisse verwalten]
[→ A.4 UNIX-Befehle]

öffnen [→ 8.2 Öffnen und Schließen einer Datei]

Position ändern [→ 8.2 Öffnen und Schließen einer Datei]
[→ 8.4 Dateien mit festgelegter Struktur]

Rechte [→ A.4 UNIX-Befehle]

schließen [→ 8.2 Öffnen und Schließen einer Datei]

speichern [→ 8.1 Dateitypen]

Text anhängen [→ 8.3 Textdateien]

Text lesen [→ 8.3 Textdateien]

Text lesen und schreiben [→ 8.4 Dateien mit festgelegter Struktur]

Text schreiben [→ 8.3 Textdateien]

Text schreiben, formatiert [→ 8.4 Dateien mit festgelegter Struktur]

umbenennen [→ 8.9 Dateien und Verzeichnisse verwalten]
[→ A.4 UNIX-Befehle]

verschieben [→ 8.9 Dateien und Verzeichnisse verwalten]
[→ A.4 UNIX-Befehle]

verwalten [→ 8.9 Dateien und Verzeichnisse verwalten]

Zugriffsart [→ 8.1 Dateitypen]

Zugriffszeitpunkt [→ 8.8 Informationen über Dateien]

Datenbank [→ 10.1 Aufbau von Datenbanken]

Aktionsabfrage [→ 10.2 SQLite]

Auswahlabfrage [→ 10.2 SQLite]

auswählen [→ 10.4 MySQL]

Datensatz [→ 10.1 Aufbau von Datenbanken]

Datensatz ändern [→ 10.2 SQLite]

Datensatz anzeigen [→ 10.2 SQLite]

Datensatz einfügen [→ 10.2 SQLite]

Datensatz filtern [→ 10.2 SQLite]

Datensatz löschen [→ 10.2 SQLite]

Datensatz sortieren [→ 10.2 SQLite]

Datensatzanzahl begrenzen [→ 10.5 Spiel, Version mit Highscore-Datenbank]

Datensatz-Cursor [→ 10.2 SQLite]

eindeutiger Index [→ 10.2 SQLite]

Ergebnis der Abfrage [→ 10.2 SQLite]

erzeugen [→ 10.2 SQLite]

Feld [→ 10.1 Aufbau von Datenbanken]

Internet [→ 10.3 SQLite auf dem Webserver]

Managementsystem [→ 10.2 SQLite] [→ 10.4 MySQL]

Primärschlüssel [→ 10.2 SQLite]

Server [→ 10.4 MySQL]

Tabelle [→ 10.1 Aufbau von Datenbanken]

Tabelle erzeugen [→ 10.2 SQLite]
Verbindung herstellen [→ 10.2 SQLite]
verwalten [→ 10.4 MySQL]

Datenklasse [→ 6.8 Datenklassen]

Datentyp [→ 4.1 Zahlen]

- bool* [→ 3.3 Verzweigungen]
- DATE* [→ 10.4 MySQL]
- Datenbank* [→ 10.1 Aufbau von Datenbanken]
- DOUBLE* [→ 10.4 MySQL]
- ermitteln* [→ 4.1 Zahlen]
- float* [→ 4.1 Zahlen]
- INT* [→ 10.4 MySQL]
- int* [→ 4.1 Zahlen]
- INTEGER* [→ 10.2 SQLite]
- NoneType* [→ 4.7 Wahrheitswerte und Nichts]
- REAL* [→ 10.2 SQLite]
- str* [→ 4.2 Zeichenketten]
- TEXT* [→ 10.2 SQLite]
- VARCHAR* [→ 10.4 MySQL]

Datum [→ 7.1 Datum und Uhrzeit]

Debugger [→ 5.5 Fehler und Ausnahmen]

`decode()` [→ 4.2 Zeichenketten]

`deepcopy()` [→ 4.8 Referenz, Identität und Kopie] [→ 6.5 Referenz, Identität und Kopie]

`def`

Funktion [→ 3.7 Funktionen und Module]

Methode [→ 6.2 Klassen, Objekte und eigene Methoden]

`degrees()` [→ 4.1 Zahlen]

Dekorator [→ 6.8 Datenklassen]

`del`

Destruktor [→ 6.3 Besondere Member]

Dictionary [→ 4.5 Dictionarys]

Liste [→ 4.3 Listen]

Referenz [→ 4.8 Referenz, Identität und Kopie]

`DELETE` [→ 10.2 SQLite]

`delete()`

Entry [→ 11.2 Widget-Typen]

ScrolledText [→ 11.2 Widget-Typen]

`denominator` [→ 4.1 Zahlen]

`deque()` [→ 7.2 Warteschlangen]

`DESC` [→ 10.2 SQLite]

Deserialisierung [→ 8.5 Serialisierung mit »pickle«]

`destroy()` [→ 11.1 Einführung]

Destruktor [→ 6.3 Besondere Member]

Dezimalformat [→ 5.2 Ausgabe und Formatierung]

Dezimalzahl [→ 4.1 Zahlen]

Dictionary [→ 4.5 Dictionarys]

aktualisieren [→ 4.5 Dictionarys]

erzeugen [→ 4.5 Dictionarys]

KeyError [→ 8.10 Beispielprojekt Morsezeichen]

leeres [→ 4.7 Wahrheitswerte und Nichts]

nur Schlüssel [→ 4.5 Dictionarys]

nur Werte [→ 4.5 Dictionarys]

Schlüssel prüfen [→ 4.5 Dictionarys]

Vergleich [→ 4.5 Dictionarys]

View [→ 4.5 Dictionarys]

digits [→ 4.2 Zeichenketten]

DirEntry [→ 8.7 Bearbeitung mehrerer Dateien]

discard() [→ 4.6 Sets, Mengen]

Division [→ 2.1 Python als Taschenrechner]

DOUBLE [→ 10.4 MySQL]

Double-Ended Queue [→ 7.2 Warteschlangen]

DoubleVar [→ 11.2 Widget-Typen]

Dreieck [→ 11.6 Zeichnungen und Animationen]

Dualformat [→ 5.2 Ausgabe und Formatierung]

Dualzahl [→ 4.1 Zahlen] [→ 4.1 Zahlen]

dump()

json [→ 8.6 Datenaustausch mit JSON]

pickle [→ 8.5 Serialisierung mit »pickle«]

E ↑

e (Exponential) [→ 4.1 Zahlen]

e (Format) [→ 5.2 Ausgabe und Formatierung]

e hoch x [→ 4.1 Zahlen]

EchoMode [→ 12.3 Widget-Typen]

Eigenschaft [→ 6.1 Was ist OOP?]

Eingabe

mit Hilfestellung [→ 5.1 Allgemeines]

nicht sinnvoll [→ 5.5 Fehler und Ausnahmen]

wiederholen [→ 3.6 Fehler und Ausnahmen]

Zahl [→ 3.2 Variablen und Operatoren]

Zeichenkette [→ 3.2 Variablen und Operatoren]

Eingabefeld

einzeiliges [→ 11.2 Widget-Typen] [→ 12.3 Widget-Typen]

Inhalt [→ 11.2 Widget-Typen] [→ 12.3 Widget-Typen]

löschen [→ 11.2 Widget-Typen]

mehrzeiliges [→ 11.2 Widget-Typen] [→ 12.3 Widget-Typen]

Passwort [→ 11.2 Widget-Typen]

Eingebaute Funktion [→ 5.7 Eingebaute Funktionen]

Einrückung [→ 3.3 Verzweigungen]

doppelte [→ 3.4 Schleifen]

Einzelschrittverfahren [→ 5.5 Fehler und Ausnahmen]

Element [→ 4.2 Zeichenketten]

elif [→ 3.3 Verzweigungen]

else [→ 3.3 Verzweigungen]

empty() [→ 7.2 Warteschlangen]

enable() [→ 9.2 Webserver-Programmierung]

end [→ 5.2 Ausgabe und Formatierung]

`endswith()` [→ 4.2 Zeichenketten]

`Entry` [→ 11.2 Widget-Typen]

löschen [→ 11.2 Widget-Typen]

Passwort [→ 11.2 Widget-Typen]

`Entwicklung` [→ 3.5 Entwicklung eines Programms]

`Entwicklungsumgebung` [→ 1.5 Installation unter Windows]

`enum` [→ 6.9 Enumerationen]

`Enumeration` [→ 6.9 Enumerationen]

`Ereignis`

Maus [→ 11.3 Bilder und Mausereignisse]

Qt [→ 12.1 Ein erstes Programm]

Tastatur [→ 11.6 Zeichnungen und Animationen]

`Eulersche Zahl` [→ 4.1 Zahlen]

`eval()` [→ 5.1 Allgemeines]

`except` [→ 3.6 Fehler und Ausnahmen]

`Exception` [→ 3.6 Fehler und Ausnahmen]

`exec()` [→ 5.1 Allgemeines]

QApplication [→ 12.1 Ein erstes Programm]

`execute()` [→ 10.2 SQLite] [→ 10.4 MySQL]

`exists()` [→ 8.9 Dateien und Verzeichnisse verwalten]

`exit()` [→ 8.3 Textdateien]

`Exklusives Oder, bitweise` [→ 4.1 Zahlen]

`exp()` [→ 4.1 Zahlen]

`expand` [→ 11.6 Zeichnungen und Animationen]

`Exponentialformat` [→ 5.2 Ausgabe und Formatierung]

Exponentialoperator [→ 4.1 Zahlen]

extend() [→ 7.2 Warteschlangen]

extendleft() [→ 7.2 Warteschlangen]

F ↑

f (Format) [→ 5.2 Ausgabe und Formatierung]

f (String-Literal) [→ 3.2 Variablen und Operatoren]

factorial() [→ 4.1 Zahlen]

Fakultät [→ 4.1 Zahlen]

False [→ 3.3 Verzweigungen] [→ 4.7 Wahrheitswerte und Nichts]

Farbe (HTML) [→ 11.3 Bilder und Mausereignisse]

Fehler [→ 3.6 Fehler und Ausnahmen] [→ 5.5 Fehler und Ausnahmen]

erzeugen [→ 5.5 Fehler und Ausnahmen]

Laufzeitfehler [→ 5.5 Fehler und Ausnahmen]

logischer [→ 5.5 Fehler und Ausnahmen]

Syntaxfehler [→ 5.5 Fehler und Ausnahmen]

unterscheiden [→ 5.5 Fehler und Ausnahmen]

fetchall() [→ 10.4 MySQL]

FieldStorage [→ 9.2 Webserver-Programmierung]

FIFO-Queue [→ 7.2 Warteschlangen]

fill [→ 11.6 Zeichnungen und Animationen]

filter() [→ 5.3 Funktionen für Iterables]

find_overlapping() [→ 11.6 Zeichnungen und Animationen]

find() [→ 4.2 Zeichenketten]

`findall()` [→ 7.4 Reguläre Ausdrücke]

`Fließkommazahl` [→ 4.1 Zahlen]

`float` (Datentyp) [→ 4.1 Zahlen]

`float()` [→ 3.2 Variablen und Operatoren]

`for`

Liste filtern [→ 4.3 Listen]

range() [→ 3.4 Schleifen]

Schleife [→ 3.4 Schleifen]

`form` (HTML) [→ 9.1 Laden und Senden von Internetdaten]

Formatierung [→ 3.2 Variablen und Operatoren] [→ 5.2 Ausgabe und Formatierung]

`Fraction()` [→ 4.1 Zahlen]

`fractions`

Fraction() [→ 4.1 Zahlen]

Modul [→ 4.1 Zahlen]

`Frame` [→ 11.2 Widget-Typen]

`from` [→ 5.9 Eigene Module]

`frozenset()` [→ 4.6 Sets, Mengen]

`Funktion` [→ 3.7 Funktionen und Module]

als Parameter [→ 5.6 Funktionen]

anonyme [→ 5.6 Funktionen]

Aufruf [→ 3.7 Funktionen und Module]

beenden [→ 3.7 Funktionen und Module]

benannter Parameter [→ 5.6 Funktionen]

Callback [→ 5.6 Funktionen]

Definition [→ 3.7 Funktionen und Module]
eingebaute [→ 5.7 Eingebaute Funktionen]
mehrere Rückgabewerte [→ 5.6 Funktionen]
Name [→ 3.7 Funktionen und Module]
optionaler Parameter [→ 5.6 Funktionen]
Parameter [→ 3.7 Funktionen und Module]
Parameterübergabe [→ 5.6 Funktionen]
rekursive [→ 5.6 Funktionen]
Rückgabewert [→ 3.7 Funktionen und Module]
variable Parameteranzahl [→ 5.6 Funktionen]
verzögerter Aufruf [→ 11.6 Zeichnungen und Animationen]
Funktionale Programmierung [→ 5.3 Funktionen für Iterables]
Funktionsgraph [→ 5.8 Weitere mathematische Module]

G ↑

Ganze Zahl [→ 4.1 Zahlen]
Ganzzahldivision [→ 2.1 Python als Taschenrechner]
`gcd()` [→ 4.1 Zahlen]
`geometric_mean()` [→ 5.8 Weitere mathematische Module]
Geometrie-Manager
 `grid` [→ 11.1 Einführung]
 `pack` [→ 11.1 Einführung]
 `place` [→ 11.4 Geometrie-Manager »place«]
`geometry()` [→ 11.4 Geometrie-Manager »place«]

`get_ident()` [→ 7.3 Multithreading]
`get()`
 Entry [→ 11.2 Widget-Typen]
 Listbox [→ 11.2 Widget-Typen]
 PhotoImage [→ 11.3 Bilder und Mausereignisse]
 Queue [→ 7.2 Warteschlangen]
 Scale [→ 11.2 Widget-Typen]
 ScrolledText [→ 11.2 Widget-Typen]
 Spinbox [→ 11.2 Widget-Typen]
 Widget-Variable [→ 11.2 Widget-Typen]
`glob()` [→ 8.7 Bearbeitung mehrerer Dateien]
`global` [→ 5.6 Funktionen]
 Globaler Namensraum [→ 5.6 Funktionen]
`grid` [→ 11.1 Einführung]
`grid()`
 Geometrie-Manager [→ 11.1 Einführung]
 matplotlib [→ 5.8 Weitere mathematische Module]
 Grid-Layout [→ 12.2 Layout und Größe eines Anwendungsfensters]
 Größter gemeinsamer Teiler [→ 4.1 Zahlen]
 GUI [→ 11.1 Einführung] [→ 12.1 Ein erstes Programm]

H ↑

Haltepunkt [→ 5.5 Fehler und Ausnahmen]
`harmonic_mean()` [→ 5.8 Weitere mathematische Module]
`hex()` [→ 4.1 Zahlen]

Hexadezimalformat [→ 5.2 Ausgabe und Formatierung]

Hexadezimalzahl [→ 4.1 Zahlen]

HTML

Auswahlmenü [→ 9.2 Webserver-Programmierung]

Dokument [→ 9.2 Webserver-Programmierung]

Eingabefeld [→ 9.1 Laden und Senden von Internetdaten]

Eingabefeld, mehrzeilig [→ 9.2 Webserver-Programmierung]

Farbe [→ 11.3 Bilder und Mausereignisse]

Formular [→ 9.1 Laden und Senden von Internetdaten]
[→ 9.2 Webserver-Programmierung]

Formular senden [→ 9.2 Webserver-Programmierung]

Formularelement [→ 9.2 Webserver-Programmierung]

Passwort [→ 9.2 Webserver-Programmierung]

Radiobutton [→ 9.2 Webserver-Programmierung]

`htmlentities()` [→ 9.1 Laden und Senden von Internetdaten]

Hyperlink [→ 12.3 Widget-Typen]



IDLE [→ 1.5 Installation unter Windows]

benutzen [→ 2.1 Python als Taschenrechner]

Debugger [→ 5.5 Fehler und Ausnahmen]

`if` [→ 3.3 Verzweigungen]

`case` [→ 3.3 Verzweigungen]

Liste filtern [→ 4.3 Listen]

imag [→ 4.1 Zahlen]

Imaginärteil [→ 4.1 Zahlen]

import [→ 3.2 Variablen und Operatoren] [→ 5.9 Eigene Module]

in

Dictionary [→ 4.5 Dictionarys]

Liste filtern [→ 4.3 Listen]

Sequenz [→ 4.2 Zeichenketten]

Set [→ 4.6 Sets, Mengen]

Index [→ 4.2 Zeichenketten]

negativ [→ 4.2 Zeichenketten]

index()

deque [→ 7.2 Warteschlangen]

Liste [→ 4.3 Listen]

input (HTML) [→ 9.1 Laden und Senden von Internetdaten]

input() [→ 3.2 Variablen und Operatoren]

mit Hilfestellung [→ 5.1 Allgemeines]

INSERT INTO [→ 10.2 SQLite]

insert()

deque [→ 7.2 Warteschlangen]

Listbox [→ 11.2 Widget-Typen]

Liste [→ 4.3 Listen]

ScrolledText [→ 11.2 Widget-Typen]

Installation [→ 1.5 Installation unter Windows]

Instanz [→ 6.2 Klassen, Objekte und eigene Methoden]

INT [→ 10.4 MySQL]

int (Datentyp) [→ 4.1 Zahlen]

int() [→ 3.2 Variablen und Operatoren]

INTEGER [→ 10.2 SQLite]

IntEnum [→ 6.9 Enumerationen]

Internet [→ 9.1 Laden und Senden von Internetdaten]

Daten lesen [→ 9.1 Laden und Senden von Internetdaten]

Daten senden [→ 9.1 Laden und Senden von Internetdaten]

Datenbank [→ 10.3 SQLite auf dem Webserver]

Interpunktionszeichen [→ 4.2 Zeichenketten]

IntVar [→ 11.2 Widget-Typen]

Inversion, bitweise [→ 4.1 Zahlen]

io [→ 5.8 Weitere mathematische Module]

is [→ 4.8 Referenz, Identität und Kopie] [→ 6.5 Referenz, Identität und Kopie]

isChecked()

QCheckBox [→ 12.3 Widget-Typen]

QRadioButton [→ 12.3 Widget-Typen]

isclose() [→ 4.1 Zahlen]

isqrt() [→ 4.1 Zahlen]

itemClicked [→ 12.3 Widget-Typen]

itemconfigure() [→ 11.6 Zeichnungen und Animationen]

items() [→ 4.5 Dictionarys]

Iterable [→ 4.2 Zeichenketten]

filtern [→ 5.3 Funktionen für Iterables]

Funktionsaufruf [→ 5.3 Funktionen für Iterables]

verbinden [→ 5.3 Funktionen für Iterables]

J ↑

j (komplexe Zahl) [→ 4.1 Zahlen]

JavaScript Object Notation [→ 8.6 Datenaustausch mit JSON]

JSON [→ 8.6 Datenaustausch mit JSON]

json

dump() [→ 8.6 Datenaustausch mit JSON]

load() [→ 8.6 Datenaustausch mit JSON]

K ↑

KeyError, Dictionary [→ 8.10 Beispielprojekt Morsezeichen]

keys() [→ 4.5 Dictionarys]

Klasse [→ 6.1 Was ist OOP?]

Ableitung [→ 6.6 Vererbung]

Basisklasse [→ 6.6 Vererbung]

Definition [→ 6.2 Klassen, Objekte und eigene Methoden]

vereinfachte [→ 6.8 Datenklassen]

Kodierung [→ 9.1 Laden und Senden von Internetdaten]

Kommandozeile [→ 2.3 Speichern und Ausführen]

Parameter [→ 5.10 Parameter der Kommandozeile]

Kommentar [→ 2.3 Speichern und Ausführen]

Komplexe Zahl [→ 4.1 Zahlen]

Konjugiert komplex [→ 4.1 Zahlen]

Konstante [→ 4.1 Zahlen]

Aufzählung [→ 6.9 Enumerationen]

Konstruktor [→ 6.3 Besondere Member]

Vererbung [→ 6.6 Vererbung]

Kontextmenü [→ 11.5 Menüs, Messageboxen und Dialogfelder]

Kopfrechnen [→ 3.1 Ein Spiel programmieren]

Kosinus [→ 4.1 Zahlen]

Kreis [→ 11.6 Zeichnungen und Animationen]

Kreuzprodukt [→ 5.8 Weitere mathematische Module]

L ↑

Label [→ 11.1 Einführung] [→ 12.2 Layout und Größe eines Anwendungsfensters] [→ 12.3 Widget-Typen]

Ausrichtung [→ 12.3 Widget-Typen]

Bild [→ 12.3 Widget-Typen]

Formatierung [→ 12.3 Widget-Typen]

Hyperlink [→ 12.3 Widget-Typen]

lambda [→ 5.6 Funktionen]

Länge

Dictionary [→ 4.5 Dictionarys]

Liste [→ 4.3 Listen]

Liste, mehrdimensional [→ 4.3 Listen]

Set [→ 4.6 Sets, Mengen]

Zeichenkette [→ 4.2 Zeichenketten]

Laufzeitfehler [→ 5.5 Fehler und Ausnahmen]
legend() [→ 5.8 Weitere mathematische Module]
len()
 Dictionary [→ 4.5 Dictionarys]
 Liste [→ 4.3 Listen]
 Liste, mehrdimensional [→ 4.3 Listen]
 Set [→ 4.6 Sets, Mengen]
 Zeichenkette [→ 4.2 Zeichenketten]
LibreOffice Calc [→ 8.3 Textdateien]
LIFO-Queue [→ 7.2 Warteschlangen]
LifoQueue [→ 7.2 Warteschlangen]
LIKE [→ 10.2 SQLite]
LIMIT [→ 10.5 Spiel, Version mit Highscore-Datenbank]
limit_denominator() [→ 4.1 Zahlen]
Linie [→ 11.6 Zeichnungen und Animationen]
Linksbündig [→ 5.2 Ausgabe und Formatierung]
linspace() [→ 5.8 Weitere mathematische Module]
Linux [→ 1.6 Installation unter Ubuntu Linux]
List Comprehension [→ 4.3 Listen]
Listbox [→ 11.2 Widget-Typen]
 Auswahl, einfache [→ 11.2 Widget-Typen]
 Auswahl, mehrfache [→ 11.2 Widget-Typen]
 füllen [→ 11.2 Widget-Typen]
Liste [→ 4.3 Listen]
 ändern [→ 4.3 Listen]

Anzahl bestimmter Elemente [→ 4.3 Listen]
einfache Auswahl [→ 12.3 Widget-Typen]
Eintrag [→ 12.3 Widget-Typen]
Element anfügen [→ 4.3 Listen]
Element einfügen [→ 4.3 Listen]
Element löschen [→ 4.3 Listen]
filtern [→ 4.3 Listen]
leere [→ 4.7 Wahrheitswerte und Nichts]
mehrdimensionale [→ 4.3 Listen]
mehrfache Auswahl [→ 12.3 Widget-Typen]
Methoden [→ 4.3 Listen]
Position ermitteln [→ 4.3 Listen]
sortieren [→ 4.3 Listen]
umdrehen [→ 4.3 Listen]

`load()`

json [→ 8.6 Datenaustausch mit JSON]
pickle [→ 8.5 Serialisierung mit »pickle«]

`localtime()` [→ 7.1 Datum und Uhrzeit]

`log()` [→ 4.1 Zahlen]
`log10()` [→ 4.1 Zahlen]

Logarithmus [→ 4.1 Zahlen]

Logischer Fehler [→ 5.5 Fehler und Ausnahmen]
Logischer Operator [→ 3.3 Verzweigungen]

Lokaler Namensraum [→ 5.6 Funktionen]

Lokaler Webserver [→ 9.1 Laden und Senden von Internetdaten]

ls -l [→ A.4 UNIX-Befehle]

M ↑

macOS [→ 1.7 Installation unter macOS]

mainloop() [→ 11.1 Einführung]

map() [→ 5.3 Funktionen für Iterables]

MariaDB [→ 10.4 MySQL] [→ A.3 Installation von XAMPP]

match [→ 3.3 Verzweigungen]

math

acos() [→ 4.1 Zahlen]

asin() [→ 4.1 Zahlen]

atan() [→ 4.1 Zahlen]

cos() [→ 4.1 Zahlen]

degrees() [→ 4.1 Zahlen]

e [→ 4.1 Zahlen]

exp() [→ 4.1 Zahlen]

factorial() [→ 4.1 Zahlen]

gcd() [→ 4.1 Zahlen]

isclose() [→ 4.1 Zahlen]

isqrt() [→ 4.1 Zahlen]

log() [→ 4.1 Zahlen]

log10() [→ 4.1 Zahlen]

Modul [→ 4.1 Zahlen]

pi [→ 4.1 Zahlen]

prod() [→ 4.1 Zahlen]

radians() [→ 4.1 Zahlen]

remainder() [→ 4.1 Zahlen]

sin() [→ 4.1 Zahlen]

sqrt() [→ 4.1 Zahlen]

tan() [→ 4.1 Zahlen]

Mathematische Darstellung [→ 5.8 Weitere mathematische Module]

matplotlib [→ 5.8 Weitere mathematische Module]

matrix() [→ 5.8 Weitere mathematische Module]

Matrizenrechnung [→ 5.8 Weitere mathematische Module]

Maus

Ereignis [→ 11.3 Bilder und Mausereignisse]

Position [→ 11.3 Bilder und Mausereignisse]

max() [→ 5.7 Eingebaute Funktionen]

maxsize() [→ 11.4 Geometrie-Manager »place«]

mean() [→ 5.8 Weitere mathematische Module]

median_high() [→ 5.8 Weitere mathematische Module]

median_low() [→ 5.8 Weitere mathematische Module]

median() [→ 5.8 Weitere mathematische Module]

Mehrfachvererbung [→ 6.7 Mehrfachvererbung]

Member [→ 6.1 Was ist OOP?]

Mengenlehre [→ 4.6 Sets, Mengen]

Menu [→ 11.5 Menüs, Messageboxen und Dialogfelder]

Menüleiste [→ 11.5 Menüs, Messageboxen und Dialogfelder]

Menüpunkt

erstellen [→ 11.5 Menüs, Messageboxen und Dialogfelder]

Tastaturbedienung [→ 11.5 Menüs, Messageboxen und Dialogfelder]

Messagebox [→ 11.5 Menüs, Messageboxen und Dialogfelder]

Methode [→ 6.1 Was ist OOP?]

benannter Parameter [→ 6.3 Besondere Member]

optionaler Parameter [→ 6.3 Besondere Member]

`min()` [→ 5.7 Eingebaute Funktionen]

`minsize()` [→ 11.4 Geometrie-Manager »place«]

Mischen [→ 4.3 Listen]

Mittelwert [→ 5.8 Weitere mathematische Module]

`mkdir` [→ A.4 UNIX-Befehle]

`mktime()` [→ 7.1 Datum und Uhrzeit]

`mode()` [→ 5.8 Weitere mathematische Module]

Modul

Aliasname [→ 5.8 Weitere mathematische Module]
[→ 5.9 Eigene Module]

cgi [→ 9.2 Webserver-Programmierung]

cgitb [→ 9.2 Webserver-Programmierung]

collections [→ 7.2 Warteschlangen]

copy [→ 4.8 Referenz, Identität und Kopie] [→ 6.5
Referenz, Identität und Kopie]

einbinden [→ 3.2 Variablen und Operatoren] [→ 5.9
Eigene Module]

enum [→ 6.9 Enumerationen]
erzeugen [→ 5.9 Eigene Module]
fractions [→ 4.1 Zahlen]
glob [→ 8.7 Bearbeitung mehrerer Dateien]
math [→ 4.1 Zahlen]
matplotlib [→ 5.8 Weitere mathematische Module]
mysql.connector [→ 10.4 MySQL]
numpy [→ 5.8 Weitere mathematische Module]
os [→ 8.7 Bearbeitung mehrerer Dateien]
os.path [→ 8.9 Dateien und Verzeichnisse verwalten]
pickle [→ 8.5 Serialisierung mit »pickle«]
PyQt6.QtCore.Qt [→ 12.3 Widget-Typen]
PyQt6.QtGui [→ 12.3 Widget-Typen]
PyQt6.QtWidgets [→ 12.1 Ein erstes Programm]
queue [→ 7.2 Warteschlangen]
random [→ 3.2 Variablen und Operatoren]
re [→ 7.4 Reguläre Ausdrücke]
scipy [→ 5.8 Weitere mathematische Module]
shutil [→ 8.9 Dateien und Verzeichnisse verwalten]
sqlite3 [→ 10.2 SQLite]
statistics [→ 5.8 Weitere mathematische Module]
string [→ 4.2 Zeichenketten]
sys [→ 5.10 Parameter der Kommandozeile] [→ 8.3 Textdateien]
threading [→ 7.3 Multithreading]

time [→ 7.1 Datum und Uhrzeit]

tkinter [→ 11.1 Einführung]

tkinter.messagebox [→ 11.5 Menüs, Messageboxen und Dialogfelder]

urllib [→ 9.1 Laden und Senden von Internetdaten]

urllib.parse [→ 9.1 Laden und Senden von Internetdaten]

urllib.request [→ 9.1 Laden und Senden von Internetdaten]

webbrowser [→ 9.3 Browser aufrufen]

winsound [→ 7.5 Audioausgabe]

Zusatzmodul installieren [→ A.1 Paketverwaltungsprogramm »pip«]

Modularisierung [→ 3.7 Funktionen und Module] [→ 5.9 Eigene Module]

Morsezeichen [→ 8.10 Beispielprojekt Morsezeichen]

move() [→ 8.9 Dateien und Verzeichnisse verwalten] [→ 11.6 Zeichnungen und Animationen] [→ 12.1 Ein erstes Programm]

MS Excel [→ 8.3 Textdateien]

multimode() [→ 5.8 Weitere mathematische Module]

multiple (HTML) [→ 9.2 Webserver-Programmierung]

Multithreading [→ 7.3 Multithreading]

mv [→ A.4 UNIX-Befehle]

MySQL [→ 10.4 MySQL] [→ A.3 Installation von XAMPP]

Schnittstelle [→ 10.4 MySQL]

mysql.connector [→ 10.4 MySQL]

N ↑

Nachkommastellen [→ 2.1 Python als Taschenrechner]

Anzahl [→ 5.2 Ausgabe und Formatierung]

Nachrichtenbox [→ 11.5 Menüs, Messageboxen und Dialogfelder]

name [→ 8.7 Bearbeitung mehrerer Dateien]

Namensraum [→ 5.6 Funktionen]

Nenner [→ 4.1 Zahlen]

nicht, logisches [→ 3.3 Verzweigungen] [→ 10.2 SQLite]

None [→ 4.7 Wahrheitswerte und Nichts]

NoneType [→ 4.7 Wahrheitswerte und Nichts]

NOT [→ 10.2 SQLite]

not [→ 3.3 Verzweigungen]

not in [→ 4.2 Zeichenketten]

numerator [→ 4.1 Zahlen]

Numerische Funktion [→ 5.8 Weitere mathematische Module]

numpy [→ 5.8 Weitere mathematische Module]

Array [→ 5.8 Weitere mathematische Module]

O ↑

o (Format) [→ 5.2 Ausgabe und Formatierung]

Objekt [→ 6.2 Klassen, Objekte und eigene Methoden]

ausgeben [→ 6.3 Besondere Member]

deserialisieren [→ 8.5 Serialisierung mit »pickle«]

Dictionary [→ 6.3 Besondere Member]

endet [→ 6.3 Besondere Member]
erzeugen [→ 6.2 Klassen, Objekte und eigene Methoden]
gleicher Inhalt [→ 4.8 Referenz, Identität und Kopie]
Identität [→ 4.8 Referenz, Identität und Kopie] [→ 6.5 Referenz, Identität und Kopie]
initialisieren [→ 6.3 Besondere Member]
kopieren [→ 4.8 Referenz, Identität und Kopie] [→ 6.5 Referenz, Identität und Kopie]
Referenz [→ 4.8 Referenz, Identität und Kopie] [→ 6.5 Referenz, Identität und Kopie]
serialisieren [→ 8.5 Serialisierung mit »pickle«]
Typ ermitteln [→ 4.1 Zahlen]

Objektorientierte Programmierung [→ 6.1 Was ist OOP?]
oct() [→ 4.1 Zahlen]
oder

bitweises [→ 4.1 Zahlen]
logisches [→ 3.3 Verzweigungen] [→ 10.2 SQLite]
offvalue [→ 11.2 Widget-Typen]
Oktalformat [→ 5.2 Ausgabe und Formatierung]
Oktalzahl [→ 4.1 Zahlen]
onvalue [→ 11.2 Widget-Typen]
OOP [→ 6.1 Was ist OOP?]
open()
Datei [→ 8.2 Öffnen und Schließen einer Datei]
webbrowser [→ 9.3 Browser aufrufen]

Operator

Bit- [→ 4.1 Zahlen]

für Sequenz [→ 4.2 Zeichenketten]

logischer [→ 3.3 Verzweigungen]

Rangfolge [→ 2.1 Python als Taschenrechner] [→ 3.3 Verzweigungen]

Vergleich [→ 3.3 Verzweigungen]

Verkettung [→ 4.2 Zeichenketten]

Vervielfachung [→ 4.2 Zeichenketten]

Operatormethode [→ 6.4 Operatormethoden]

Optionaler Parameter [→ 5.6 Funktionen] [→ 6.3 Besondere Member]

OR [→ 10.2 SQLite]

or [→ 3.3 Verzweigungen]

ord() [→ 5.7 Eingebaute Funktionen]

ORDER BY [→ 10.2 SQLite]

Orientation [→ 12.3 Widget-Typen]

os

Modul [→ 8.7 Bearbeitung mehrerer Dateien]

remove() [→ 8.9 Dateien und Verzeichnisse verwalten]

stat() [→ 8.8 Informationen über Dateien]

os.path [→ 8.9 Dateien und Verzeichnisse verwalten]

exists() [→ 8.9 Dateien und Verzeichnisse verwalten]

outline [→ 11.6 Zeichnungen und Animationen]

Oval [→ 11.6 Zeichnungen und Animationen]

pack [→ 11.1 Einführung]

mehrere Spalten [→ 11.2 Widget-Typen]

pack() [→ 11.1 Einführung]

padx [→ 11.1 Einführung]

pady [→ 11.1 Einführung]

Paketverwaltungsprogramm [→ 1.5 Installation unter Windows] [→ A.1 Paketverwaltungsprogramm »pip«]

Parallele Programme [→ 7.3 Multithreading]

Parameter [→ 3.7 Funktionen und Module]

benannter [→ 5.6 Funktionen] [→ 6.3 Besondere Member]

optionaler [→ 5.6 Funktionen] [→ 6.3 Besondere Member]

Substitution [→ 10.2 SQLite]

Übergabe [→ 5.6 Funktionen]

variable Anzahl [→ 5.6 Funktionen]

pass [→ 5.1 Allgemeines]

password (HTML) [→ 9.2 Webserver-Programmierung]

Passwort [→ 5.4 Verschlüsselung] [→ 12.3 Widget-Typen]

Pfeilkopf [→ 11.6 Zeichnungen und Animationen]

PhotoImage [→ 11.3 Bilder und Mausereignisse]

PHP [→ 9.1 Laden und Senden von Internetdaten] [→ 9.1 Laden und Senden von Internetdaten] [→ A.3 Installation von XAMPP]

phpMyAdmin [→ 10.4 MySQL] [→ A.3 Installation von XAMPP]

Pi [→ 4.1 Zahlen]

pickle [→ 8.5 Serialisierung mit »pickle«]

dump() [→ 8.5 Serialisierung mit »pickle«]

load() [→ 8.5 Serialisierung mit »pickle«]

pip [→ A.1 Paketverwaltungsprogramm »pip«]

Windows [→ 1.5 Installation unter Windows]

place [→ 11.4 Geometrie-Manager »place«]

place() [→ 11.4 Geometrie-Manager »place«]

PlaySound() [→ 7.5 Audioausgabe]

plot() [→ 5.8 Weitere mathematische Module]

Polygon [→ 11.6 Zeichnungen und Animationen]

pop() [→ 7.2 Warteschlangen]

popleft() [→ 7.2 Warteschlangen]

POST-Methode [→ 9.1 Laden und Senden von Internetdaten]

PRIMARY KEY [→ 10.2 SQLite]

print() [→ 2.2 Erstes Programm]

Formatierung [→ 3.2 Variablen und Operatoren] [→ 5.2 Ausgabe und Formatierung]

lange Ausgabe [→ 2.3 Speichern und Ausführen]

Trennen und Beenden [→ 5.2 Ausgabe und Formatierung]

Verkettung [→ 2.3 Speichern und Ausführen]

PriorityQueue [→ 7.2 Warteschlangen]

prod() [→ 4.1 Zahlen]

Produkt [→ 4.1 Zahlen]

Programm

Abbruch vermeiden [→ 3.6 Fehler und Ausnahmen]

anhalten [→ 7.1 Datum und Uhrzeit]

ausführen, IDLE [→ 2.3 Speichern und Ausführen]

ausführen, Kommandozeile [→ 2.3 Speichern und Ausführen]

eingeben [→ 2.2 Erstes Programm]

entwickeln [→ 3.5 Entwicklung eines Programms]

zerlegen [→ 3.7 Funktionen und Module]

Prozentformat [→ 5.2 Ausgabe und Formatierung]

punctuation [→ 4.2 Zeichenketten]

put()

PhotoImage [→ 11.3 Bilder und Mausereignisse]

Queue [→ 7.2 Warteschlangen]

pyinstaller [→ A.2 Erstellen von EXE-Dateien]

pyplot [→ 5.8 Weitere mathematische Module]

PyQt [→ 12.1 Ein erstes Programm]

PyQt6

QtCore.Qt [→ 12.3 Widget-Typen]

QtGui [→ 12.3 Widget-Typen]

QtWidgets [→ 12.1 Ein erstes Programm]

Python

auf dem Webserver [→ 9.2 Webserver-Programmierung]

Interpreter [→ 2.3 Speichern und Ausführen]

Version [→ 2.3 Speichern und Ausführen]

Q ↑

- QAbstractItemView [→ 12.3 Widget-Typen]
- QApplication [→ 12.1 Ein erstes Programm]
- QButtonGroup [→ 12.3 Widget-Typen]
- QCheckBox [→ 12.3 Widget-Typen]
- QComboBox [→ 12.3 Widget-Typen]
- QDoubleSpinBox [→ 12.3 Widget-Typen]
- QGridLayout [→ 12.2 Layout und Größe eines Anwendungsfensters]
- QLabel [→ 12.2 Layout und Größe eines Anwendungsfensters] [→ 12.3 Widget-Typen]
- QLineEdit [→ 12.3 Widget-Typen]
- QListWidget [→ 12.3 Widget-Typen]
- QListWidgetItem [→ 12.3 Widget-Typen]
- QPixmap [→ 12.3 Widget-Typen]
- QPushButton [→ 12.1 Ein erstes Programm]
- QRadioButton [→ 12.3 Widget-Typen]
- qsize() [→ 7.2 Warteschlangen]
- QSlider [→ 12.3 Widget-Typen]
- QSpinBox [→ 12.3 Widget-Typen]
- Qt [→ 12.1 Ein erstes Programm]
- QTextEdit [→ 12.3 Widget-Typen]
- QtGui [→ 12.3 Widget-Typen]
- Quadrat [→ 11.6 Zeichnungen und Animationen]

queue [→ 7.2 Warteschlangen]
quit() [→ 12.1 Ein erstes Programm]
QWidget [→ 12.1 Ein erstes Programm]

R ↑

radians() [→ 4.1 Zahlen]
radio (HTML) [→ 9.2 Webserver-Programmierung]
Radiobutton [→ 11.2 Widget-Typen] [→ 12.3 Widget-Typen]
 Gruppe [→ 12.3 Widget-Typen]
 in Menü [→ 11.5 Menüs, Messageboxen und Dialogfelder]
Rahmen [→ 11.6 Zeichnungen und Animationen]
raise [→ 5.5 Fehler und Ausnahmen]
randbelow() [→ 5.4 Verschlüsselung]
randint() [→ 3.2 Variablen und Operatoren]
random
 choice() [→ 4.3 Listen]
 randint() [→ 3.2 Variablen und Operatoren]
 seed() [→ 3.2 Variablen und Operatoren]
 shuffle() [→ 4.3 Listen]
range() [→ 3.4 Schleifen]
re
 findall() [→ 7.4 Reguläre Ausdrücke]
 Modul [→ 7.4 Reguläre Ausdrücke]
 sub() [→ 7.4 Reguläre Ausdrücke]
read()

Textdatei [→ 8.3 Textdateien]

wavfile [→ 5.8 Weitere mathematische Module]

`readline()` [→ 8.3 Textdateien]

`readlines()` [→ 8.3 Textdateien]

`REAL` [→ 10.2 SQLite]

`real` [→ 4.1 Zahlen]

`Realteil` [→ 4.1 Zahlen]

`Rechteck` [→ 11.6 Zeichnungen und Animationen]

`Rechtsbündig` [→ 5.2 Ausgabe und Formatierung]

`Referenz`

löschen [→ 4.8 Referenz, Identität und Kopie]

Objekt [→ 4.8 Referenz, Identität und Kopie] [→ 6.5
Referenz, Identität und Kopie]

übergeben [→ 5.6 Funktionen]

`Regulärer Ausdruck` [→ 7.4 Reguläre Ausdrücke]

`Rekursion` [→ 5.6 Funktionen] [→ 8.7 Bearbeitung mehrerer
Dateien]

`relief` [→ 11.1 Einführung]

`relx` [→ 11.4 Geometrie-Manager »place«]

`rely` [→ 11.4 Geometrie-Manager »place«]

`remainder()` [→ 4.1 Zahlen]

`remove()`

Datei [→ 8.9 Dateien und Verzeichnisse verwalten]

Liste [→ 4.3 Listen]

`replace()` [→ 4.2 Zeichenketten]

Reservierte Wörter [→ 2.1 Python als Taschenrechner]

resizable() [→ 11.1 Einführung]

Rest [→ 2.1 Python als Taschenrechner] [→ 4.1 Zahlen]

return [→ 3.7 Funktionen und Module]

mehrere Werte [→ 5.6 Funktionen]

reverse() [→ 4.3 Listen]

reversed() [→ 5.7 Eingebaute Funktionen]

rfind() [→ 4.2 Zeichenketten]

rm [→ A.4 UNIX-Befehle]

rmdir [→ A.4 UNIX-Befehle]

rotate() [→ 7.2 Warteschlangen]

Rückgabewert [→ 3.7 Funktionen und Module]

mehrere [→ 5.6 Funktionen]

Runden [→ 4.1 Zahlen]

RuntimeError [→ 5.5 Fehler und Ausnahmen]

rwx [→ A.4 UNIX-Befehle]

S ↑

Scale [→ 11.2 Widget-Typen]

scandir() [→ 8.7 Bearbeitung mehrerer Dateien]

Schädlicher Code [→ 9.1 Laden und Senden von Internetdaten] [→ 10.2 SQLite]

Schieberegler [→ 11.2 Widget-Typen]

Schleife

abbrechen [→ 3.4 Schleifen]

for [→ 3.4 Schleifen]

fortfahren [→ 3.4 Schleifen]

while [→ 3.4 Schleifen]

Schlüssel [→ 4.5 Dictionarys]

scipy [→ 5.8 Weitere mathematische Module]

Scrollbar [→ 11.2 Widget-Typen]

zuordnen [→ 11.2 Widget-Typen]

ScrolledText [→ 11.2 Widget-Typen]

einfügen [→ 11.2 Widget-Typen]

Inhalt [→ 11.2 Widget-Typen]

löschen [→ 11.2 Widget-Typen]

secrets [→ 5.4 Verschlüsselung]

choice() [→ 5.4 Verschlüsselung]

randbelow() [→ 5.4 Verschlüsselung]

seed() [→ 3.2 Variablen und Operatoren]

seek() [→ 8.2 Öffnen und Schließen einer Datei] [→ 8.4 Dateien mit festgelegter Struktur]

SELECT [→ 10.2 SQLite]

select (HTML) [→ 9.2 Webserver-Programmierung]

selected (HTML) [→ 9.2 Webserver-Programmierung]

selectedItems() [→ 12.3 Widget-Typen]

SelectionMode [→ 12.3 Widget-Typen]

selectmode [→ 11.2 Widget-Typen]

self [→ 6.2 Klassen, Objekte und eigene Methoden]

sender() [→ 12.3 Widget-Typen]

sep [→ 5.2 Ausgabe und Formatierung]

Sequenz [→ 4.2 Zeichenketten]

Bereich [→ 4.2 Zeichenketten]

Operator [→ 4.2 Zeichenketten]

sortieren [→ 5.7 Eingebaute Funktionen]

umdrehen [→ 5.7 Eingebaute Funktionen]

Serialisierung [→ 8.5 Serialisierung mit »pickle«]

Set

Differenzmenge [→ 4.6 Sets, Mengen]

Element hinzufügen [→ 4.6 Sets, Mengen]

Element löschen [→ 4.6 Sets, Mengen]

erzeugen [→ 4.6 Sets, Mengen]

kopieren [→ 4.6 Sets, Mengen]

leeren [→ 4.6 Sets, Mengen]

leeres [→ 4.7 Wahrheitswerte und Nichts]

Teilmenge [→ 4.6 Sets, Mengen]

unveränderliches [→ 4.6 Sets, Mengen]

Vereinigungsmenge [→ 4.6 Sets, Mengen]

`set_title()` [→ 5.8 Weitere mathematische Module]

`set_xlabel()` [→ 5.8 Weitere mathematische Module]

`set_ylabel()` [→ 5.8 Weitere mathematische Module]

`set()`

Set [→ 4.6 Sets, Mengen]

Spinbox [→ 11.2 Widget-Typen]

Widget-Variable [→ 11.2 Widget-Typen]

`setAlignment()` [→ 12.3 Widget-Typen]

`setChecked()`
 QCheckBox [→ 12.3 Widget-Typen]
 QRadioButton [→ 12.3 Widget-Typen]

`setCurrentRow()` [→ 12.3 Widget-Typen]

`setDecimals()` [→ 12.3 Widget-Typen]

`setEchoMode()` [→ 12.3 Widget-Typen]

`setEditable()` [→ 12.3 Widget-Typen]

`setEnabled()` [→ 12.3 Widget-Typen]

`setFixedHeight()` [→ 12.3 Widget-Typen]

`setFixedSize()` [→ 12.2 Layout und Größe eines Anwendungsfensters]

`setFixedWidth()` [→ 12.3 Widget-Typen]

`setLayout()` [→ 12.2 Layout und Größe eines Anwendungsfensters]

`setMaximum()`
 QSlider [→ 12.3 Widget-Typen]
 QSpinBox [→ 12.3 Widget-Typen]

`setMinimum()`
 QSlider [→ 12.3 Widget-Typen]
 QSpinBox [→ 12.3 Widget-Typen]

`setOpenExternalLinks()` [→ 12.3 Widget-Typen]

`setOrientation()` [→ 12.3 Widget-Typen]

`setPageStep()` [→ 12.3 Widget-Typen]

`setPixmap()` [→ 12.3 Widget-Typen]

`setSelectionMode()` [→ 12.3 Widget-Typen]

`setSingleStep()`

QSlider [→ 12.3 Widget-Typen]

QSpinBox [→ 12.3 Widget-Typen]

`setStyleSheet()` [→ 12.3 Widget-Typen]

`setText()` [→ 12.2 Layout und Größe eines Anwendungsfensters]

`setTickInterval()` [→ 12.3 Widget-Typen]

`setTickPosition()` [→ 12.3 Widget-Typen]

`setValue()`

QSlider [→ 12.3 Widget-Typen]

QSpinBox [→ 12.3 Widget-Typen]

`setWindowTitle()` [→ 12.1 Ein erstes Programm]

`shape` [→ 5.8 Weitere mathematische Module]

`show` [→ 11.2 Widget-Typen]

`show()`

messagebox [→ 11.5 Menüs, Messageboxen und Dialogfelder]

pyplot [→ 5.8 Weitere mathematische Module]

QWidget [→ 12.1 Ein erstes Programm]

`showerror()` [→ 11.5 Menüs, Messageboxen und Dialogfelder]

`showinfo()` [→ 11.5 Menüs, Messageboxen und Dialogfelder]

`showwarning()` [→ 11.5 Menüs, Messageboxen und Dialogfelder]

`shuffle()` [→ 4.3 Listen]

`shutil`

copy() [→ 8.9 Dateien und Verzeichnisse verwalten]
Modul [→ 8.9 Dateien und Verzeichnisse verwalten]
move() [→ 8.9 Dateien und Verzeichnisse verwalten]

Signal (Qt) [→ 12.1 Ein erstes Programm]
 Sender [→ 12.3 Widget-Typen]

Signalverarbeitung [→ 5.8 Weitere mathematische Module]

SimpleQueue [→ 7.2 Warteschlangen]

sin() [→ 4.1 Zahlen]

Sinus [→ 4.1 Zahlen]

Skalarprodukt [→ 5.8 Weitere mathematische Module]

sleep() [→ 7.1 Datum und Uhrzeit]

Slice [→ 4.2 Zeichenketten]

Slider [→ 12.3 Widget-Typen]

Slot [→ 12.1 Ein erstes Programm]

sort() [→ 4.3 Listen]

sorted() [→ 5.7 Eingebaute Funktionen]

Spiel Kopfrechnen [→ 3.1 Ein Spiel programmieren]

Spinbox [→ 11.2 Widget-Typen] [→ 12.3 Widget-Typen]

split() [→ 4.2 Zeichenketten]

SQL [→ 10.1 Aufbau von Datenbanken]

SQL-Befehl senden [→ 10.2 SQLite]

SQL-Injection [→ 10.2 SQLite]

sqlite3

connect() [→ 10.2 SQLite]

connection [→ 10.2 SQLite]

cursor [→ 10.2 SQLite]

Modul [→ 10.2 SQLite]

`sqrt()` [→ 4.1 Zahlen]

`starred expression` [→ 4.4 Tupel]

`startswith()` [→ 4.2 Zeichenketten]

`stat()` [→ 8.8 Informationen über Dateien]

`state` [→ 11.2 Widget-Typen]

`stateChanged` [→ 12.3 Widget-Typen]

`statistics` [→ 5.8 Weitere mathematische Module]

`Statistik` [→ 5.8 Weitere mathematische Module]

`sticky` [→ 11.1 Einführung]

`str`

count() [→ 4.2 Zeichenketten]

Datentyp [→ 4.2 Zeichenketten]

endswith() [→ 4.2 Zeichenketten]

find() [→ 4.2 Zeichenketten]

replace() [→ 4.2 Zeichenketten]

rfind() [→ 4.2 Zeichenketten]

split() [→ 4.2 Zeichenketten]

startswith() [→ 4.2 Zeichenketten]

strip() [→ 4.2 Zeichenketten]

`str()` [→ 4.2 Zeichenketten]

`strftime()` [→ 7.1 Datum und Uhrzeit]

`String` [→ 4.2 Zeichenketten]

`string` [→ 4.2 Zeichenketten]

ascii_letters [→ 4.2 Zeichenketten]
digits [→ 4.2 Zeichenketten]
leerer [→ 4.7 Wahrheitswerte und Nichts]
punctuation [→ 4.2 Zeichenketten]
String-Literal [→ 3.2 Variablen und Operatoren] [→ 5.2 Ausgabe und Formatierung]
StringVar [→ 11.2 Widget-Typen]
strip() [→ 4.2 Zeichenketten]
struct_time [→ 7.1 Datum und Uhrzeit]
sub() [→ 7.4 Reguläre Ausdrücke]
subplots() [→ 5.8 Weitere mathematische Module]
Suchtext [→ 4.2 Zeichenketten]
sum() [→ 5.7 Eingebaute Funktionen]
super() [→ 6.6 Vererbung]
Syntaxfehler [→ 5.5 Fehler und Ausnahmen]
sys [→ 5.10 Parameter der Kommandozeile] [→ 8.3 Textdateien]
Systemton [→ 7.5 Audioausgabe] [→ 11.5 Menüs, Messageboxen und Dialogfelder]

T ↑

takeItem() [→ 12.3 Widget-Typen]
tan() [→ 4.1 Zahlen]
Tangens [→ 4.1 Zahlen]
target [→ 7.3 Multithreading]
Taschenrechner [→ 2.1 Python als Taschenrechner]

Tastaturereignis [→ 11.6 Zeichnungen und Animationen]
Taste F5 [→ 2.3 Speichern und Ausführen] [→ 5.5 Fehler und Ausnahmen]
Teilmenge [→ 4.6 Sets, Mengen]
Terminal [→ 2.3 Speichern und Ausführen]
TEXT [→ 10.2 SQLite]
text [→ 11.1 Einführung]
text() [→ 12.3 Widget-Typen]
textarea (HTML) [→ 9.2 Webserver-Programmierung]
textChanged [→ 12.3 Widget-Typen]
Textfeld [→ 11.6 Zeichnungen und Animationen]
Text-Widget [→ 11.2 Widget-Typen]
Thread
 Ausnahme [→ 7.3 Multithreading]
 gemeinsame Daten [→ 7.3 Multithreading]
 Identifikation [→ 7.3 Multithreading]
 Klasse [→ 7.3 Multithreading]
threading
 get_ident() [→ 7.3 Multithreading]
 Modul [→ 7.3 Multithreading]
Tickmark [→ 12.3 Widget-Typen]
TickPosition [→ 12.3 Widget-Typen]
time
 Modul [→ 7.1 Datum und Uhrzeit]
 sleep() [→ 7.1 Datum und Uhrzeit]

time() [→ 7.1 Datum und Uhrzeit]

`time()` [→ 7.1 Datum und Uhrzeit]

Titelzeile [→ 12.1 Ein erstes Programm]

`title()`

Anwendungsfenster [→ 11.1 Einführung]

matplotlib [→ 5.8 Weitere mathematische Module]

Tk

Bibliothek [→ 11.1 Einführung]

Klasse [→ 11.1 Einführung]

`tk_popup()` [→ 11.5 Menüs, Messageboxen und Dialogfelder]

`tkinter` [→ 11.1 Einführung]

`tkinter.messagebox` [→ 11.5 Menüs, Messageboxen und Dialogfelder]

`tkinter.scrolledtext` [→ 11.2 Widget-Typen]

`toggled` [→ 12.3 Widget-Typen]

Toleranz [→ 4.1 Zahlen]

Tonsignal [→ 8.10 Beispielprojekt Morsezeichen]

`toPlainText()` [→ 12.3 Widget-Typen]

`Toplevel()` [→ 11.5 Menüs, Messageboxen und Dialogfelder]

`transparency_get()` [→ 11.3 Bilder und Mausereignisse]

`transparency_set()` [→ 11.3 Bilder und Mausereignisse]

Trennlinie [→ 11.5 Menüs, Messageboxen und Dialogfelder]

Trennzeichen [→ 5.2 Ausgabe und Formatierung]

True [→ 3.3 Verzweigungen] [→ 4.7 Wahrheitswerte und Nichts]

try [→ 3.6 Fehler und Ausnahmen]

Tupel [→ 4.4 Tupel]

entpacken [→ 4.4 Tupel]

leeres [→ 4.7 Wahrheitswerte und Nichts]

verpacken [→ 4.4 Tupel]

type hint [→ 3.7 Funktionen und Module]

type() [→ 4.1 Zahlen]

Typhinweis [→ 3.7 Funktionen und Module] [→ 6.8 Datenklassen]

U ↑

Überladung [→ 3.7 Funktionen und Module] [→ 6.3 Besondere Member]

Überschreibung [→ 6.6 Vererbung]

Übungsaufgaben [→ 1.4 Übungen]

Ubuntu Linux [→ 1.6 Installation unter Ubuntu Linux]

Befehle [→ A.4 UNIX-Befehle]

Uhrzeit [→ 7.1 Datum und Uhrzeit]

Umwandlung

in Zahl [→ 3.2 Variablen und Operatoren]

in Zeichenkette [→ 4.2 Zeichenketten]

Zahlensystem [→ 4.1 Zahlen]

und

bitweises [→ 4.1 Zahlen]

logisches [→ 3.3 Verzweigungen] [→ 10.2 SQLite]

Unicode [→ 5.7 Eingebaute Funktionen]

UPDATE [→ 10.2 SQLite]

update() [→ 4.5 Dictionarys]

urlencode() [→ 9.1 Laden und Senden von Internetdaten]

urllib [→ 9.1 Laden und Senden von Internetdaten]

urllib.parse

Modul [→ 9.1 Laden und Senden von Internetdaten]

urlencode() [→ 9.1 Laden und Senden von Internetdaten]

urllib.request

Modul [→ 9.1 Laden und Senden von Internetdaten]

urlopen() [→ 9.1 Laden und Senden von Internetdaten]

urlretrieve() [→ 9.1 Laden und Senden von Internetdaten]

urlopen() [→ 9.1 Laden und Senden von Internetdaten]

urlretrieve() [→ 9.1 Laden und Senden von Internetdaten]

USE [→ 10.4 MySQL]

UTF-8 [→ 9.1 Laden und Senden von Internetdaten]

V ↑

value()

QSlider [→ 12.3 Widget-Typen]

QSpinBox [→ 12.3 Widget-Typen]

valueChanged

QSlider [→ 12.3 Widget-Typen]

QSpinBox [→ 12.3 Widget-Typen]

`ValueError` [→ 3.6 Fehler und Ausnahmen] [→ 5.5 Fehler und Ausnahmen]

`values()` [→ 4.5 Dictionaries]

`VARCHAR` [→ 10.4 MySQL]

`Variable` [→ 2.1 Python als Taschenrechner] [→ 3.2 Variablen und Operatoren]

kontrollieren [→ 5.5 Fehler und Ausnahmen]

Name [→ 2.1 Python als Taschenrechner]

`Vektorrechnung` [→ 5.8 Weitere mathematische Module]

`Vererbung` [→ 6.6 Vererbung]

mehrfache [→ 6.7 Mehrfachvererbung]

`Vergleichsoperator` [→ 3.3 Verzweigungen]

`Verschlüsselung` [→ 5.4 Verschlüsselung]

`Verzeichnis`

aktuelles [→ A.4 UNIX-Befehle]

anlegen [→ A.4 UNIX-Befehle]

Inhalt [→ A.4 UNIX-Befehle]

löschen [→ A.4 UNIX-Befehle]

Rechte [→ A.4 UNIX-Befehle]

übergeordnetes [→ A.4 UNIX-Befehle]

verwalten [→ 8.9 Dateien und Verzeichnisse verwalten]

wechseln [→ A.4 UNIX-Befehle]

`Verzweigung` [→ 3.3 Verzweigungen]

Bedingter Ausdruck [→ 3.3 Verzweigungen]

if [→ 3.3 Verzweigungen]

match [→ 3.3 Verzweigungen]
mehrere Operatoren [→ 3.3 Verzweigungen]
mehrfache [→ 3.3 Verzweigungen]
Vieleck [→ 11.6 Zeichnungen und Animationen]
View [→ 4.5 Dictionarys]
Vorzeichen [→ 3.3 Verzweigungen]

W ↑

Wahrheitswert [→ 3.3 Verzweigungen] [→ 4.7
Wahrheitswerte und Nichts]
Wahrscheinlichkeit [→ 5.11 Programm »Bruchtraining«]
walrus-operator [→ 3.4 Schleifen]
Warteschlange [→ 7.2 Warteschlangen]
WAV-Datei [→ 5.8 Weitere mathematische Module] [→ 7.5
Audioausgabe]
wavfile [→ 5.8 Weitere mathematische Module]
webbrowser
 Modul [→ 9.3 Browser aufrufen]
 open() [→ 9.3 Browser aufrufen]
Webserver [→ 9.1 Laden und Senden von Internetdaten]
 Programmierung [→ 9.2 Webserver-Programmierung]
WHERE [→ 10.2 SQLite]
while [→ 3.4 Schleifen]
Widget [→ 11.1 Einführung] [→ 12.1 Ein erstes Programm]
 Abstand [→ 11.1 Einführung]

anordnen (grid) [→ 11.1 Einführung] [→ 12.2 Layout und Größe eines Anwendungsfensters]

anordnen (pack) [→ 11.1 Einführung]

anordnen (place) [→ 11.4 Geometrie-Manager »place«]

anordnen, Frame [→ 11.2 Widget-Typen]

anordnen, relativ [→ 11.4 Geometrie-Manager »place«]

Aufschrift [→ 11.1 Einführung] [→ 12.2 Layout und Größe eines Anwendungsfensters]

Ausdehnung [→ 11.6 Zeichnungen und Animationen]

Ausgabe [→ 11.1 Einführung]

Breite [→ 11.1 Einführung] [→ 12.3 Widget-Typen]

Button [→ 11.1 Einführung] [→ 12.1 Ein erstes Programm]

Checkbox [→ 12.3 Widget-Typen]

Checkbutton [→ 11.2 Widget-Typen]

Combobox [→ 12.3 Widget-Typen]

deaktivieren [→ 11.2 Widget-Typen] [→ 12.3 Widget-Typen]

Eingabe [→ 12.3 Widget-Typen] [→ 12.3 Widget-Typen]

Entry [→ 11.2 Widget-Typen]

Ereignis verbinden [→ 11.3 Bilder und Mausereignisse]

erhobenes oder vertieftes [→ 11.1 Einführung]

Frame [→ 11.2 Widget-Typen]

Funktion ausführen [→ 11.1 Einführung]

Hintergrundfarbe [→ 11.5 Menüs, Messageboxen und Dialogfelder] [→ 12.3 Widget-Typen]

Höhe [→ 12.3 Widget-Typen]

Label [→ 11.1 Einführung] [→ 12.2 Layout und Größe eines Anwendungsfensters]

Listbox [→ 11.2 Widget-Typen]

Liste [→ 12.3 Widget-Typen]

Orientierung [→ 11.1 Einführung]

Position ändern [→ 11.4 Geometrie-Manager »place«]

Radiobutton [→ 11.2 Widget-Typen] [→ 12.3 Widget-Typen]

Rahmen [→ 11.1 Einführung]

Scale [→ 11.2 Widget-Typen]

Scrollbar [→ 11.2 Widget-Typen]

ScrolledText [→ 11.2 Widget-Typen]

Slider [→ 12.3 Widget-Typen]

Spinbox [→ 11.2 Widget-Typen] [→ 12.3 Widget-Typen]

Text [→ 11.2 Widget-Typen]

Textausrichtung [→ 11.1 Einführung]

Variable [→ 11.2 Widget-Typen]

width [→ 11.1 Einführung]

Linie [→ 11.6 Zeichnungen und Animationen]

Windows [→ 1.5 Installation unter Windows]

Winkelfunktion

math [→ 4.1 Zahlen]

numpy [→ 5.8 Weitere mathematische Module]

winsound [→ 7.5 Audioausgabe]

Beep() [→ 7.5 Audioausgabe]

PlaySound() [→ 7.5 Audioausgabe]

Wissenschaftliche Berechnungen [→ 5.8 Weitere mathematische Module]

`write()`

Textdatei [→ 8.3 Textdateien]

wavfile [→ 5.8 Weitere mathematische Module]

`writelines()` [→ 8.3 Textdateien]

Wurzel [→ 4.1 Zahlen] [→ 4.1 Zahlen]

X ↑

x (Format) [→ 5.2 Ausgabe und Formatierung]

XAMPP [→ 9.1 Laden und Senden von Internetdaten] [→ A.3 Installation von XAMPP]

`xlabel()` [→ 5.8 Weitere mathematische Module]

Y ↑

`ylabel()` [→ 5.8 Weitere mathematische Module]

`yscrollcommand` [→ 11.2 Widget-Typen]

`yview` [→ 11.2 Widget-Typen]

Z ↑

Zahl [→ 4.1 Zahlen]

annähern [→ 4.1 Zahlen]

auswählen [→ 12.3 Widget-Typen]

formatieren [→ 5.2 Ausgabe und Formatierung] [→ 5.2 Ausgabe und Formatierung]

ganze [→ 4.1 Zahlen]

komplex [→ 4.1 Zahlen]

mit Nachkommastellen [→ 4.1 Zahlen]

Zahlensystem [→ 4.1 Zahlen]

Zähler [→ 4.1 Zahlen]

Zeichenkette [→ 4.2 Zeichenketten]

Änderbarkeit [→ 4.2 Zeichenketten]

formatieren [→ 5.2 Ausgabe und Formatierung]

Leerzeichen entfernen [→ 4.2 Zeichenketten]

mehrzeilige [→ 4.2 Zeichenketten]

Suchen und Ersetzen [→ 4.2 Zeichenketten] [→ 7.4

Reguläre Ausdrücke]

zerlegen [→ 4.2 Zeichenketten]

Zeichnung [→ 11.6 Zeichnungen und Animationen]

Zeichnungsobjekt [→ 11.6 Zeichnungen und Animationen]

ändern [→ 11.6 Zeichnungen und Animationen]

animieren [→ 11.6 Zeichnungen und Animationen]

Position [→ 11.6 Zeichnungen und Animationen]

Überlappung [→ 11.6 Zeichnungen und Animationen]

verschieben [→ 11.6 Zeichnungen und Animationen]

Zeilenende [→ 5.2 Ausgabe und Formatierung]

Zeitangabe

Differenz [→ 7.1 Datum und Uhrzeit]

Formatierung [→ 7.1 Datum und Uhrzeit]

in Sekunden [→ 7.1 Datum und Uhrzeit]

Nullpunkt [→ 7.1 Datum und Uhrzeit]

Umwandlung [→ 7.1 Datum und Uhrzeit]

Zentralwert [→ 5.8 Weitere mathematische Module]

ZeroDivisionError [→ 5.5 Fehler und Ausnahmen]

Ziffern [→ 4.2 Zeichenketten]

zip() [→ 5.3 Funktionen für Iterables]

Zufall

Element [→ 4.3 Listen] [→ 5.4 Verschlüsselung]

ganze Zahl [→ 3.2 Variablen und Operatoren]

mischen [→ 4.3 Listen]

random [→ 3.2 Variablen und Operatoren]

secrets [→ 5.4 Verschlüsselung]

Zuweisung [→ 2.1 Python als Taschenrechner] [→ 3.2 Variablen und Operatoren]

kombinierte Operatoren [→ 5.1 Allgemeines]

mehrfache [→ 4.4 Tupel]

Rechtliche Hinweise

Das vorliegende Werk ist in all seinen Teilen urheberrechtlich geschützt. Weitere Hinweise dazu finden Sie in den Allgemeinen Geschäftsbedingungen des Anbieters, bei dem Sie das Werk erworben haben.

Markenschutz

Die in diesem Werk wiedergegebenen Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. können auch ohne besondere Kennzeichnung Marken sein und als solche den gesetzlichen Bestimmungen unterliegen.

Haftungsausschluss

Ungeachtet der Sorgfalt, die auf die Erstellung von Text, Abbildungen und Programmen verwendet wurde, können weder Verlag noch Autor*innen, Herausgeber*innen, Übersetzer*innen oder Anbieter für mögliche Fehler und deren Folgen eine juristische Verantwortung oder irgendeine Haftung übernehmen.

Über den Autor



Thomas Theis ist Dipl.-Ing. Technische Informatik. Als Softwareentwickler verfügt er über langjährige Erfahrung, ebenso als IT-Dozent, unter anderem an der Fachhochschule Aachen. Er leitet Schulungen zu C/C++, Visual Basic und Webprogrammierung und ist Autor vieler erfolgreicher Fachbücher.

Dokumentenarchiv

Das Dokumentenarchiv umfasst alle Abbildungen und ggf. Tabellen und Fußnoten dieses E-Books im Überblick.



Abbildung 1.1 Startmenü mit Eintrag zu Python 3.10

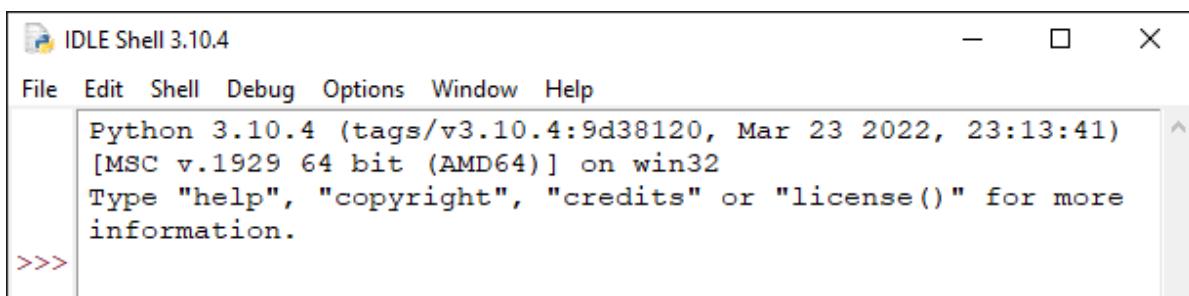
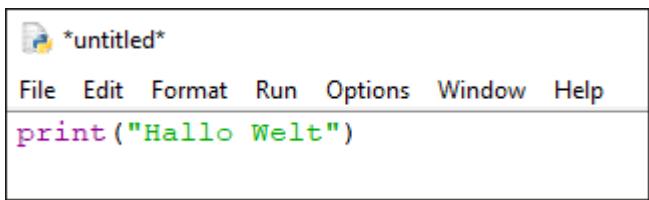


Abbildung 2.1 Python-Entwicklungsumgebung IDLE



The image shows a screenshot of a Python code editor. At the top, there is a menu bar with the following items: File, Edit, Format, Run, Options, Window, and Help. Below the menu bar, the title bar displays the text "*untitled*". In the main editing area, there is a single line of Python code: `print("Hallo Welt")`. The code is highlighted in green, indicating it is valid Python syntax.

```
*untitled*
File Edit Format Run Options Window Help
print("Hallo Welt")
```

Abbildung 2.2 Eingabe des Programms in neuem Fenster

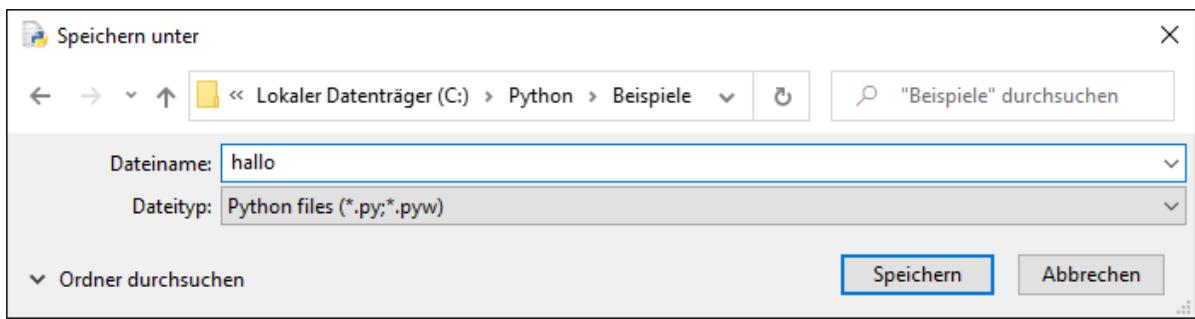
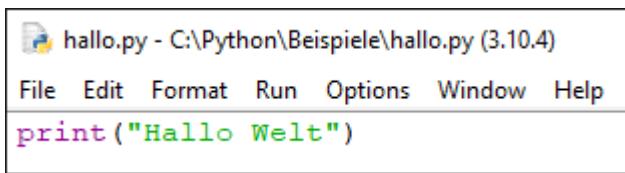
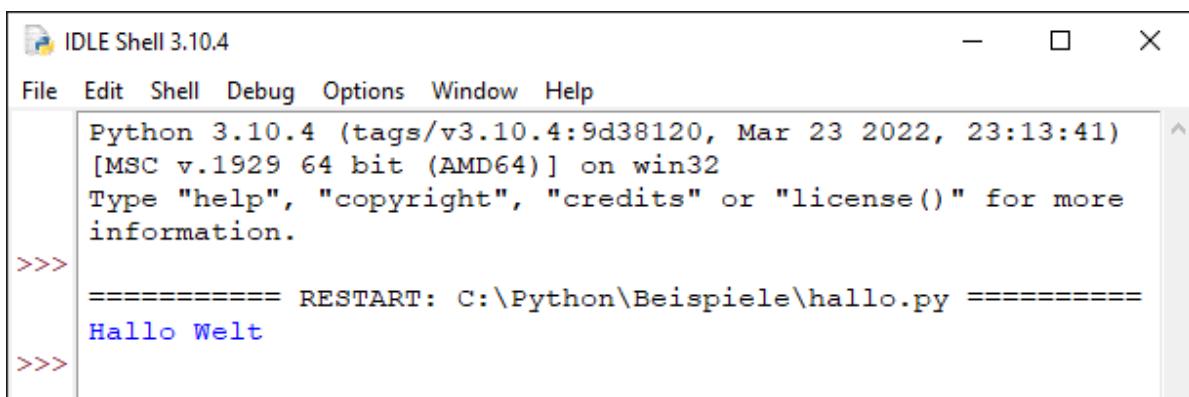


Abbildung 2.3 Speichern des Python-Programms



```
hallo.py - C:\Python\Beispiele\hallo.py (3.10.4)
File Edit Format Run Options Window Help
print("Hallo Welt")
```

Abbildung 2.4 Dateiname und Pfad in der Titelzeile

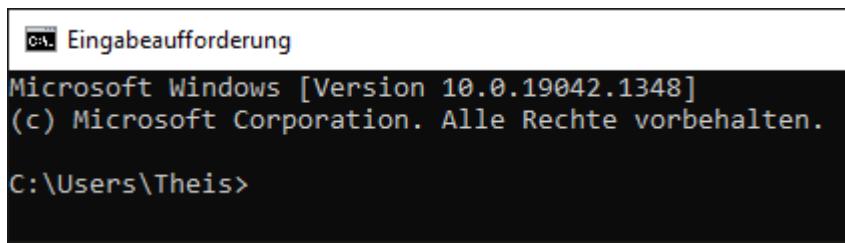


The screenshot shows the Python IDLE Shell interface. The title bar reads "IDLE Shell 3.10.4". The menu bar includes "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The main window displays the following text:

```
Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41)
[MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more
information.

>>> ===== RESTART: C:\Python\Beispiele\hallo.py =====
Hello Welt
>>>
```

Abbildung 2.5 Ergebnis des Programms



```
Eingabeaufforderung
Microsoft Windows [Version 10.0.19042.1348]
(c) Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\Theis>
```

Abbildung 2.6 Kommandozeilenfenster

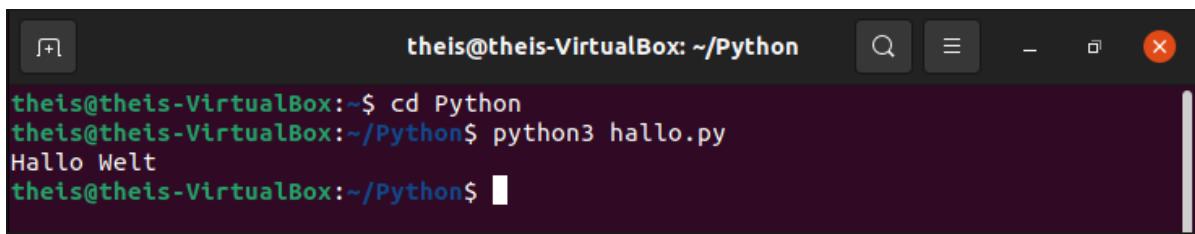
```
Eingabeaufforderung
Microsoft Windows [Version 10.0.19042.1348]
(c) Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\Theis>cd \Python\Beispiele

C:\Python\Beispiele>python hallo.py
Hallo Welt

C:\Python\Beispiele>
```

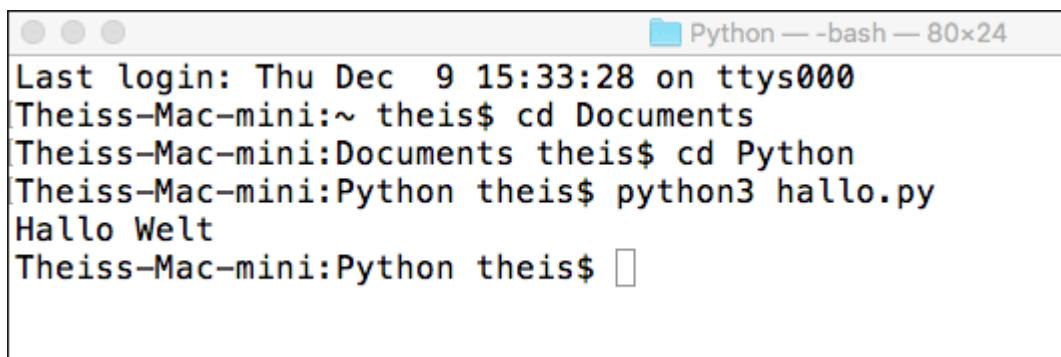
Abbildung 2.7 Aufruf im Kommandozeilenfenster



The screenshot shows a terminal window titled "theis@theis-VirtualBox: ~/Python". The window contains the following text:

```
theis@theis-VirtualBox:~$ cd Python
theis@theis-VirtualBox:~/Python$ python3 hallo.py
Hallo Welt
theis@theis-VirtualBox:~/Python$
```

Abbildung 2.8 Aufruf von Python 3 in Ubuntu Linux



```
Last login: Thu Dec  9 15:33:28 on ttys000
Theiss-Mac-mini:~ theis$ cd Documents
Theiss-Mac-mini:Documents theis$ cd Python
Theiss-Mac-mini:Python theis$ python3 hallo.py
Hallo Welt
Theiss-Mac-mini:Python theis$
```

Abbildung 2.9 Aufruf von Python 3 in macOS

```
-----  
*** Trennung ***  
-----  
x = 12 , y = 5  
-----  
*** Trennung ***  
-----  
x + y = 17  
-----  
*** Trennung ***  
-----  
x - y = 7  
-----  
*** Trennung ***  
-----
```

Abbildung 3.1 Einfache Funktionen

```
|      Hallo Welt  
|  
|Hallo Welt|
```

Abbildung 4.1 Löschen von Zeichen am Anfang und am Ende des Textes

```
Zahlen: 14.285714285714286 0.2857142857142857

Format f, Standard: 14.285714 14.285714 0.285714
Format f, nach Komma: 14.2857142857142864755815026
Format f, gesamt: 14.2857142857

Format e, Standard: 1.428571e+01
Format e, nach Komma: 1.429e+01
Format e, gesamt: 1.429e+01

Format %, Standard: 28.571429%
Format %, nach Komma: 28.571%
Format %, gesamt: 28.571%
```

Abbildung 5.1 Formatierung von Zahlen mit Nachkommastellen

61	111101	75	3d
62	111110	76	3e
63	111111	77	3f
64	1000000	100	40
65	1000001	101	41
66	1000010	102	42

Abbildung 5.2 Formatierung von ganzen Zahlen

Nr	Name	Anz	EP	GP
0023	Apfel	1	2.95 Euro	2.95 Euro
0008	Banane	3	1.45 Euro	4.35 Euro
0042	Pfirsich	5	3.05 Euro	15.25 Euro

Abbildung 5.3 Formatierung von Zeichenketten

Inch	cm
15.0	38.1
20.0	50.8
25.0	63.5
30.0	76.2
35.0	88.9
40.0	101.6

Abbildung 5.4 Übung »u_literal«

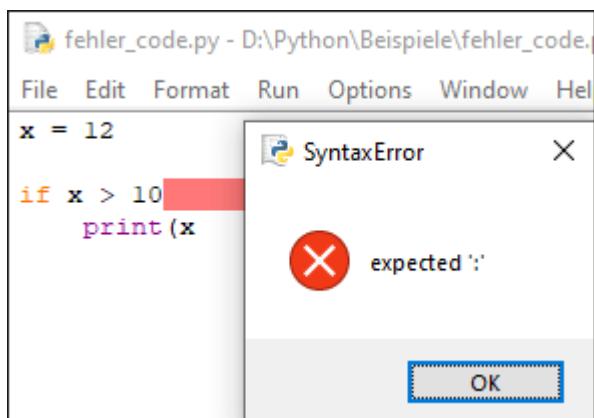


Abbildung 5.5 Anzeige des ersten Fehlers

The screenshot shows a Python code editor window titled "fehler_code.py - D:\Python\Beispiele\fehler_code.py (3.10)". The code contains the following lines:

```
x = 12
if x > 10:
    print(x
```

A modal dialog box titled "SyntaxError" is displayed, indicating the error: "'(' was never closed". The dialog has an "OK" button.

Abbildung 5.6 Anzeige des zweiten Fehlers

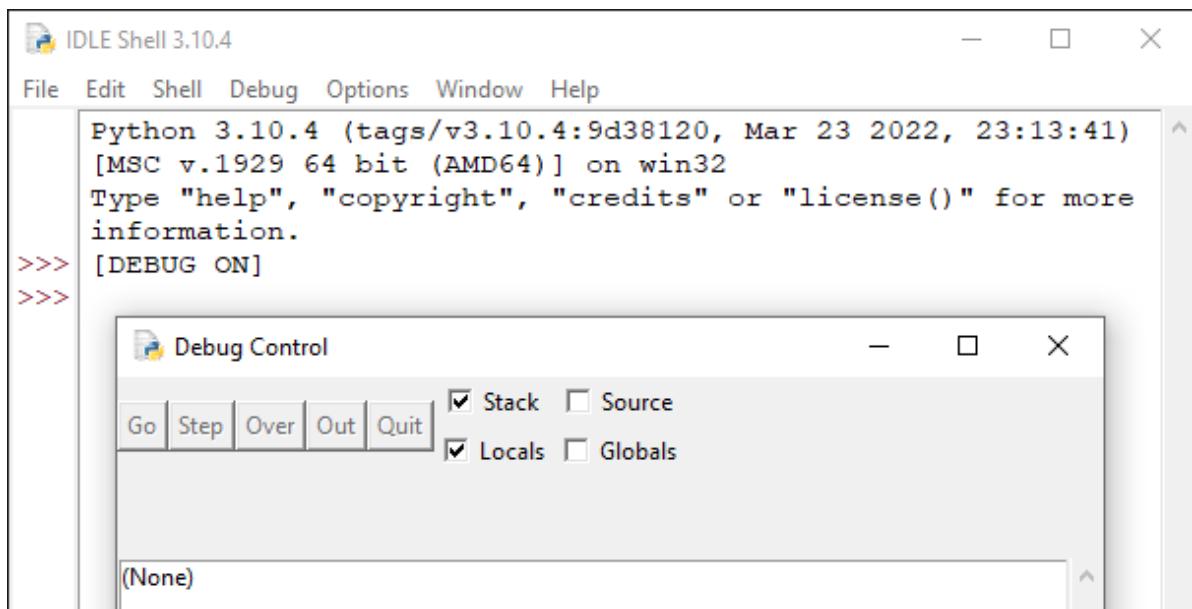


Abbildung 5.7 Dialogfeld »Debug Control« und Meldung

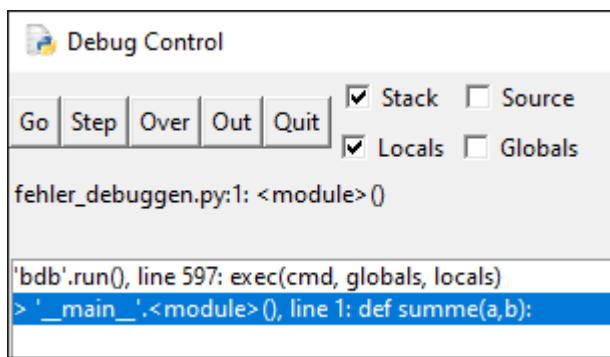


Abbildung 5.8 Nach dem Start des Programms

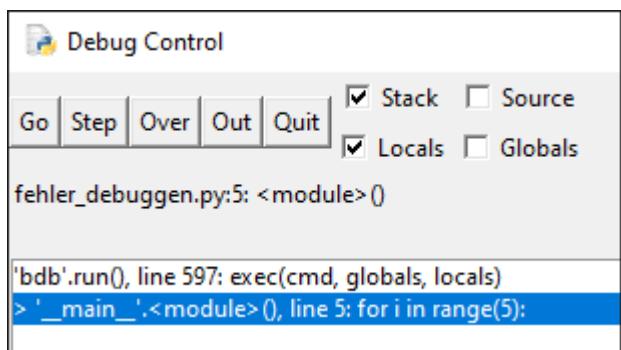


Abbildung 5.9 Erste ausgeführte Programmzeile

```
__package__     None
__spec__       None
erg           10
i              0
summe         <function sum...001F5827324D0>
```

Abbildung 5.10 Hauptprogramm: aktuelle Werte von »i« und »erg«

Locals	
a	10
b	2
c	12

Abbildung 5.11 Funktion: aktuelle Werte von »a«, »b« und »c«

```
>>> [DEBUG ON]
>>> ====== RESTART:
10
11
```

Abbildung 5.12 Ausgabe der ersten Ergebnisse in der »Idle Shell«

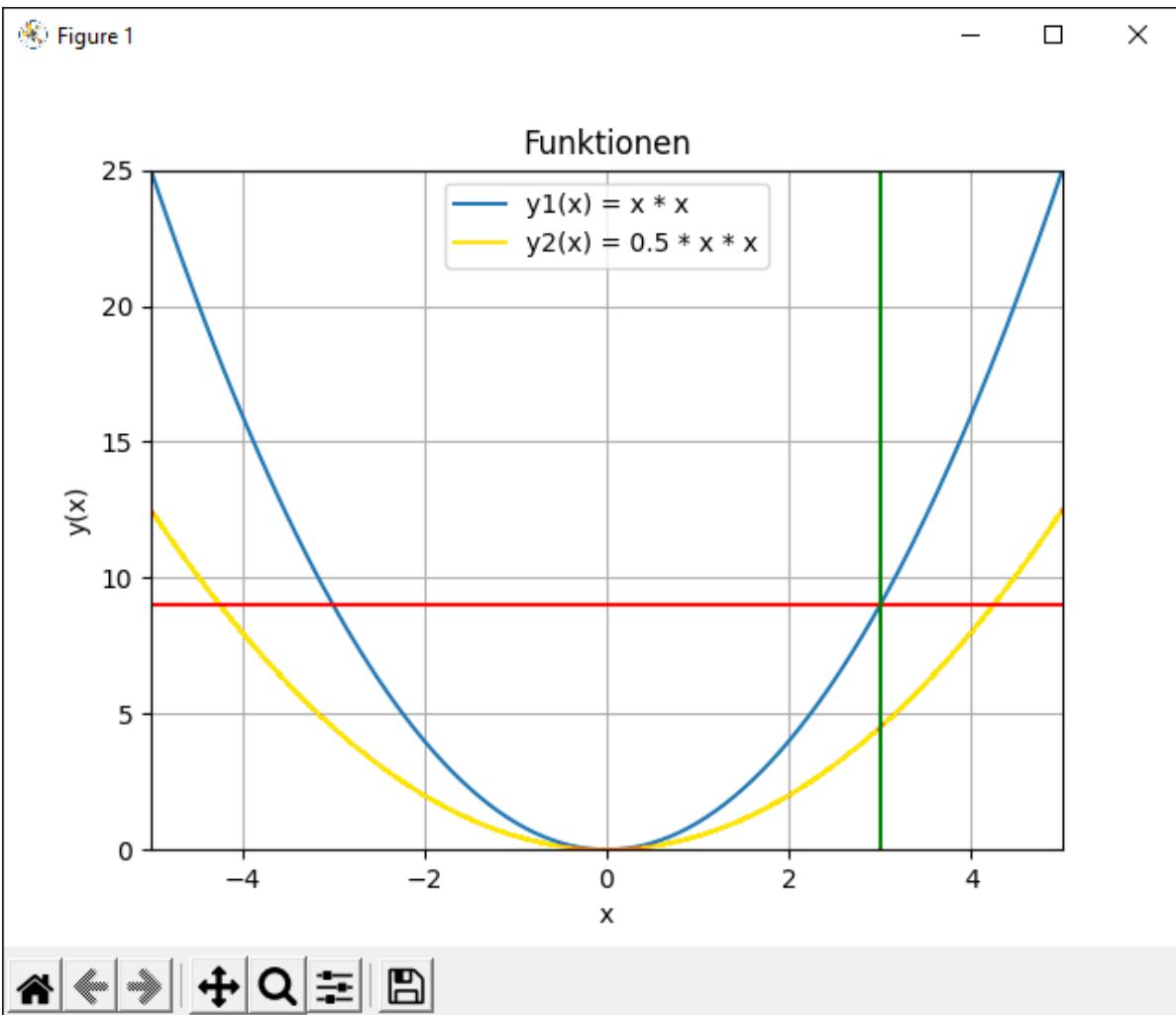


Abbildung 5.13 Funktionen $y_1(x) = x^2$ und $y_2(x) = 0.5x^2$

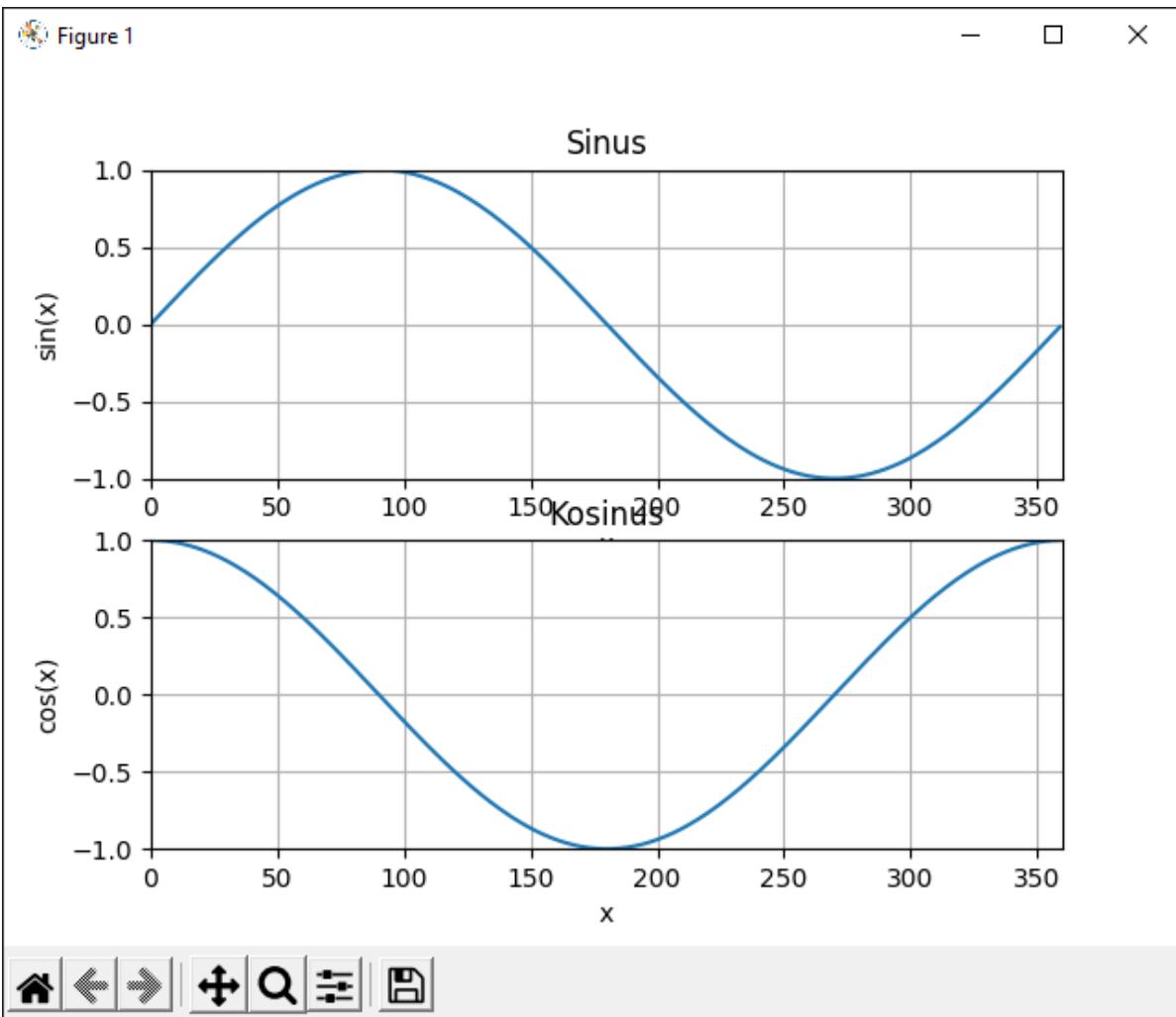


Abbildung 5.14 Funktionen $y_1(x) = \sin(x)$ und $y_2(x) = \cos(x)$

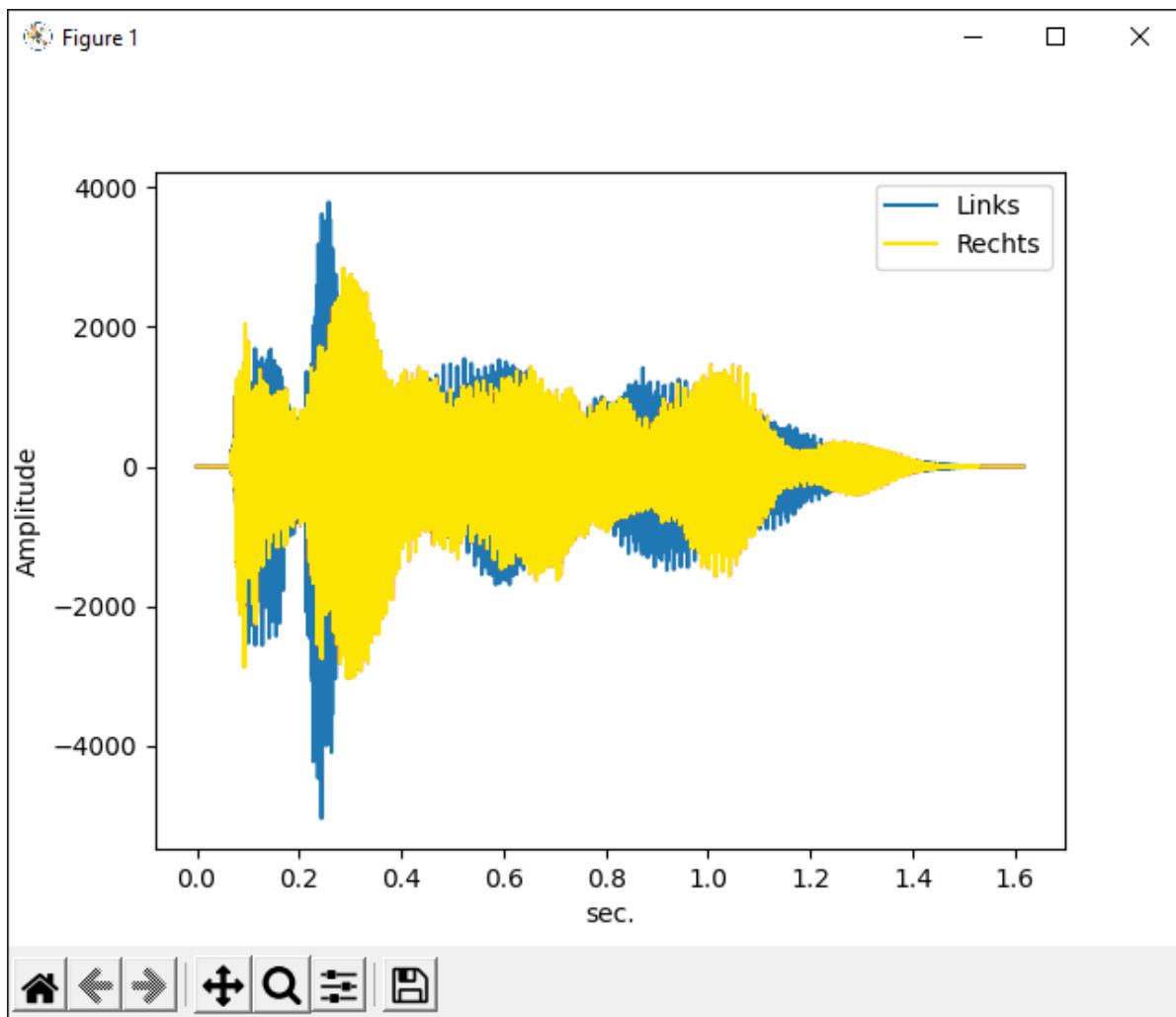


Abbildung 5.15 Darstellung von digitalen Audiosignalen

	A	B	C	D	E	F
1	Maier	Hans	6714	3500	15.03.1962	
2	Schmitz	Peter	81343	3750	12.04.1958	
3	Mertens	Julia	2297	3621,5	30.12.1959	
4						

Abbildung 8.1 CSV-Datei in MS Excel

	A	B	C	D	E	F
1	Maier	Hans	6714	3500	15.03.1962	
2	Schmitz	Peter	81343	3750	12.04.1958	
3	Mertens	Julia	2297	3621,5	30.12.1959	
4	Weber	Jürgen	4711	2900	12.08.1976	
5						

Abbildung 8.2 CSV-Datei in MS Excel, ergänzt

```
Bitte eingeben (0: Ende, 1: Highscores, 2: Spielen): 1
P. Name          Zeit
1. Ole           8.03 sec
2. Rudi          8.36 sec
3. Gerd          9.36 sec
4. Ben           12.42 sec
5. Tom           12.75 sec
Bitte eingeben (0: Ende, 1: Highscores, 2: Spielen): 2
Bitte geben Sie Ihren Namen ein (max. 10 Zeichen): Paul
Aufgabe 1 von 5: 20 + 30 : 50
*** Richtig ***
Aufgabe 2 von 5: 29 + 29 : 58
*** Richtig ***
Aufgabe 3 von 5: 23 + 21 : 44
*** Richtig ***
Aufgabe 4 von 5: 29 + 10 : 39
*** Richtig ***
Aufgabe 5 von 5: 27 + 30 : 57
*** Richtig ***
Ergebnis: 5 von 5 in 9.48 Sekunden, Highscore
P. Name          Zeit
1. Ole           8.03 sec
2. Rudi          8.36 sec
3. Gerd          9.36 sec
4. Paul          9.48 sec
5. Ben           12.42 sec
6. Tom           12.75 sec
Bitte eingeben (0: Ende, 1: Highscores, 2: Spielen): 0
```

Abbildung 8.3 Eingabebeispiel

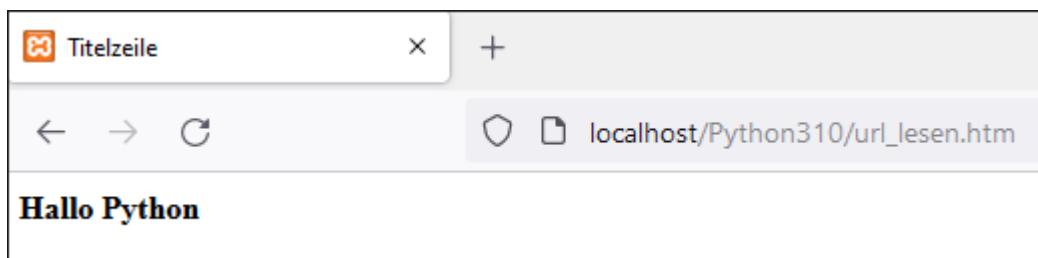


Abbildung 9.1 Erstes HTML-Dokument

A screenshot of a web browser window titled "Daten senden". The address bar shows "localhost/Python310/senden_post.htm". The page content is a form with the following text:
Bitte senden Sie Ihre Daten:
Nachname: Maier
Vorname: Werner

Abbildung 9.2 Formular mit Beispieleingabe

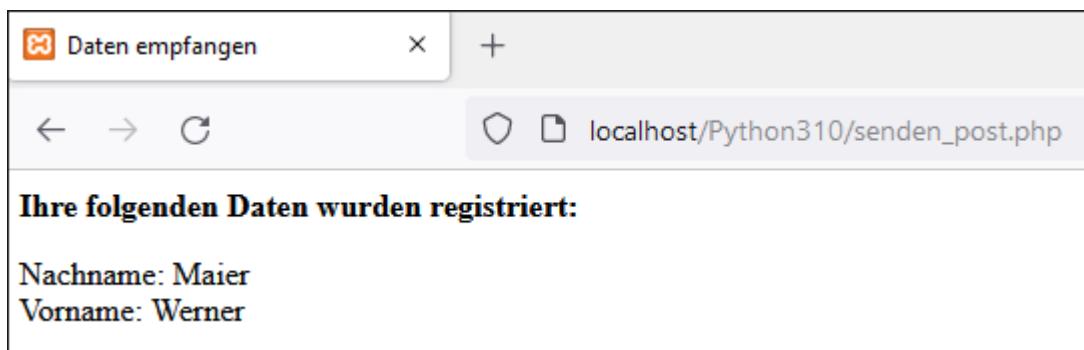


Abbildung 9.3 Antwort des PHP-Programms

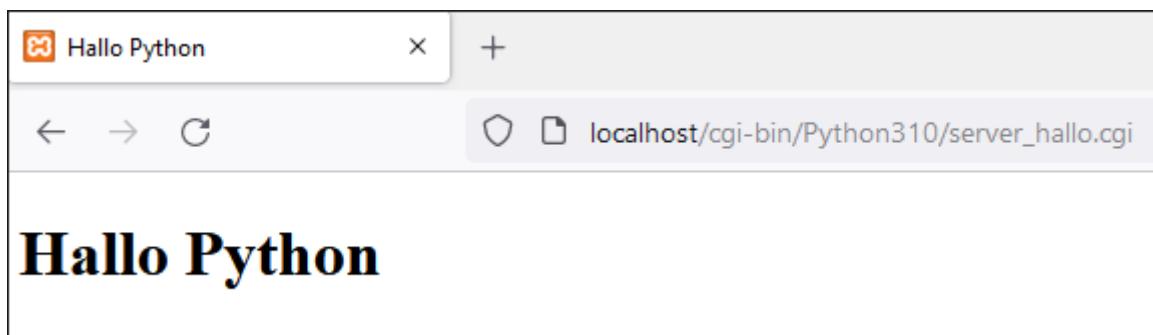


Abbildung 9.4 Erstes Python-HTML-Dokument

 Daten senden X +

← → ⌂ localhost/Python310/server_antworten.htm

Bitte senden Sie Ihre Daten:

Maier	Nachname
Werner	Vorname

Daten absenden

Abbildung 9.5 Formular mit Beispieleingabe

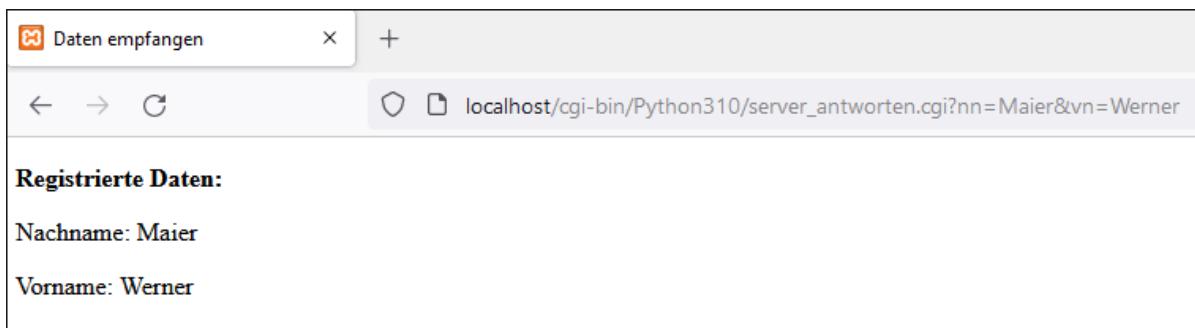
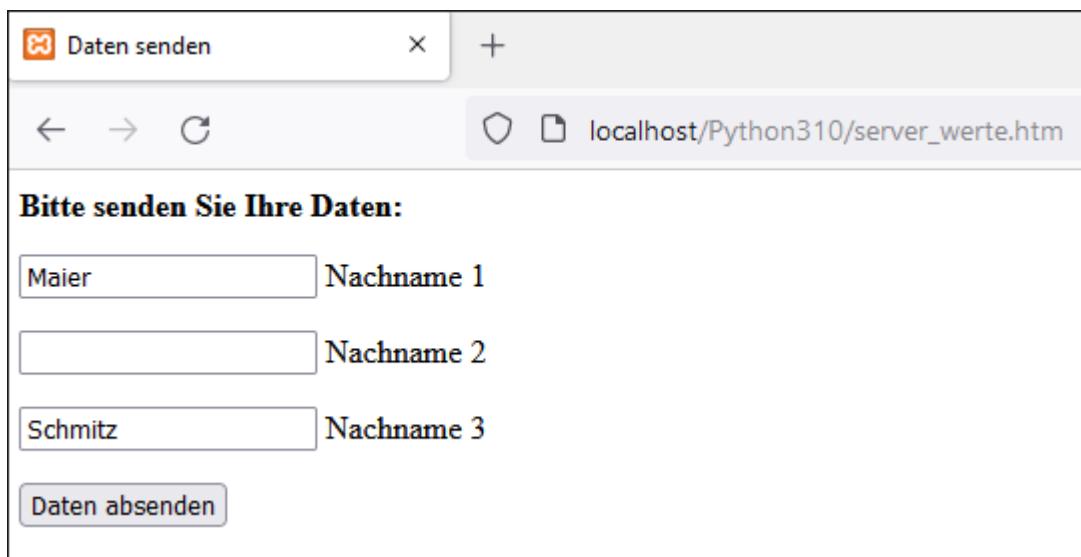


Abbildung 9.6 Antwort des Python-Programms

A screenshot of a web browser window titled "Daten senden". The address bar shows "localhost/Python310/server_werte.htm". The main content area contains the following text:

Bitte senden Sie Ihre Daten:

<input type="text" value="Maier"/>	Nachname 1
<input type="text"/>	Nachname 2
<input type="text" value="Schmitz"/>	Nachname 3

Abbildung 9.7 Formular mit Beispieleingabe



Abbildung 9.8 Antwort des Python-Programms

Daten senden +

localhost/Python310/server_pizza.htm

Python Pizza Service

Stammkunde (5% Rabatt) Code
 Neukunde

Maier Nachname Werner Vorname
Pepperoniweg Straße 4 Hausnr.
63999 PLZ Teigdorf Ort

Sorte: Pizza Python (8.50 €) ▾
Zusätze: Extra Pepperoni (1.00 €) ^
Extra Oliven (1.20 €) ▾

Express-Service (max. 30 Minuten, 1.50 € extra)

Bitte zweimal klingeln

Bemerkung: //

Abbildung 9.9 Python Pizza Service, Bestellung

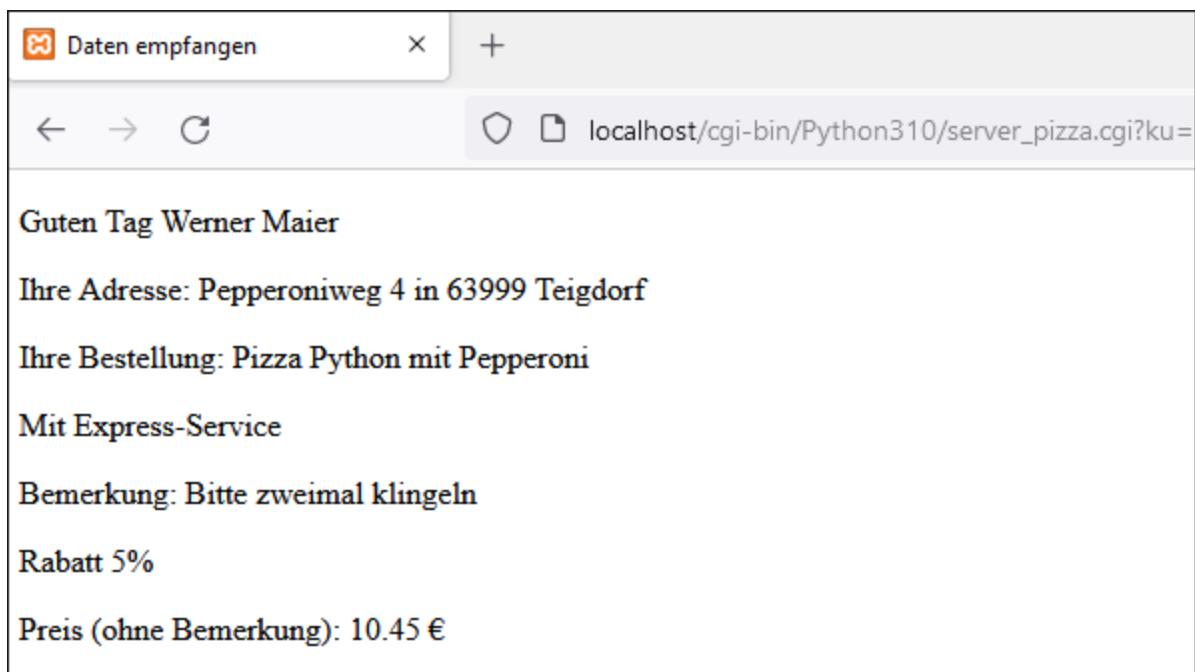


Abbildung 9.10 Python Pizza Service, Bestätigung der Bestellung

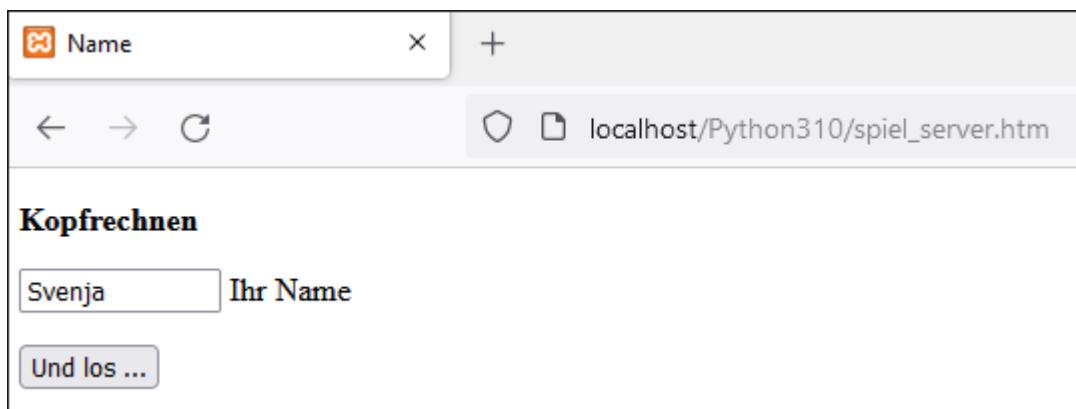


Abbildung 9.11 Beginn des Spiels

Aufgaben [+](#)

[←](#) [→](#) [C](#) localhost/cgi-bin/Python310/spiel_server_a.cgi?name=Svenja

Kopfrechnen

Hello **Svenja**, Ihre Aufgaben

1. $13 + 16 =$

2. $14 + 14 =$

3. $12 + 11 =$

4. $26 + 11 =$

5. $15 + 18 =$

[Fertig](#)

Abbildung 9.12 Fünf Aufgaben mit Lösungsversuchen

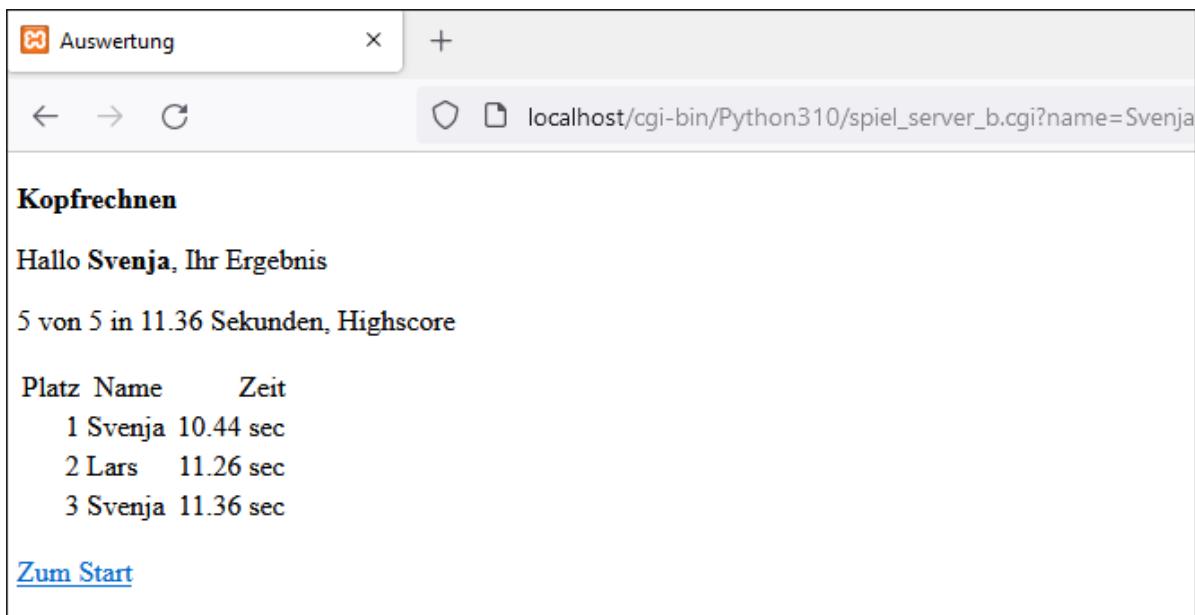


Abbildung 9.13 Bewertung, Highscore

Name	Vorname	PNr.	Gehalt	Geburtstag
Mertens	Julia	2297	3621.5	30.12.1959
Maier	Hans	6714	3500.0	15.03.1962
Schmitz	Peter	81343	3750.0	12.04.1958

Abbildung 10.1 Datenbankinhalte auf dem Webserver



Abbildung 11.1 Erstes Programm unter Windows 10



Abbildung 11.2 Erstes Programm unter Ubuntu Linux

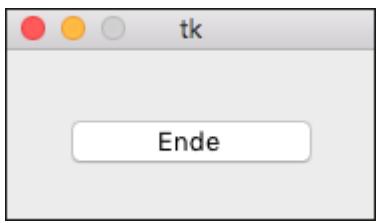


Abbildung 11.3 Erstes Programm unter macOS

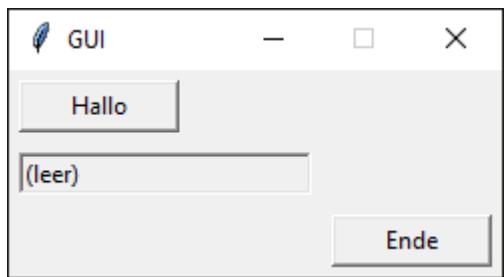


Abbildung 11.4 Anordnung von Widgets

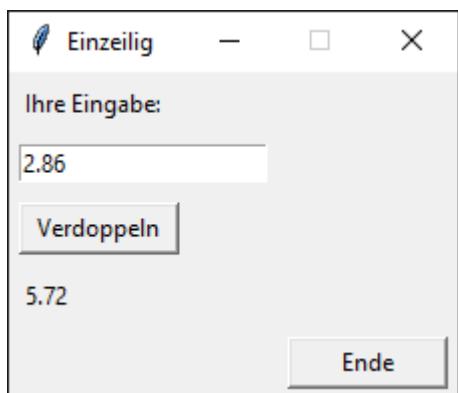


Abbildung 11.5 Einzeiliges Eingabefeld

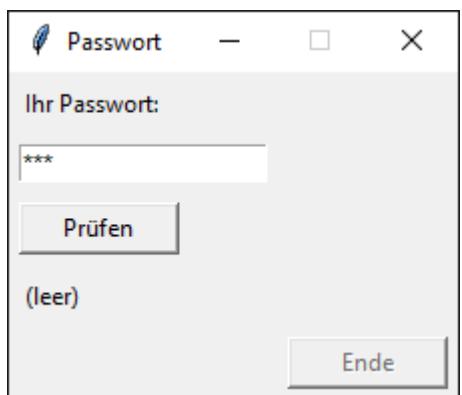


Abbildung 11.6 Versteckte Eingabe, Widget deaktivieren

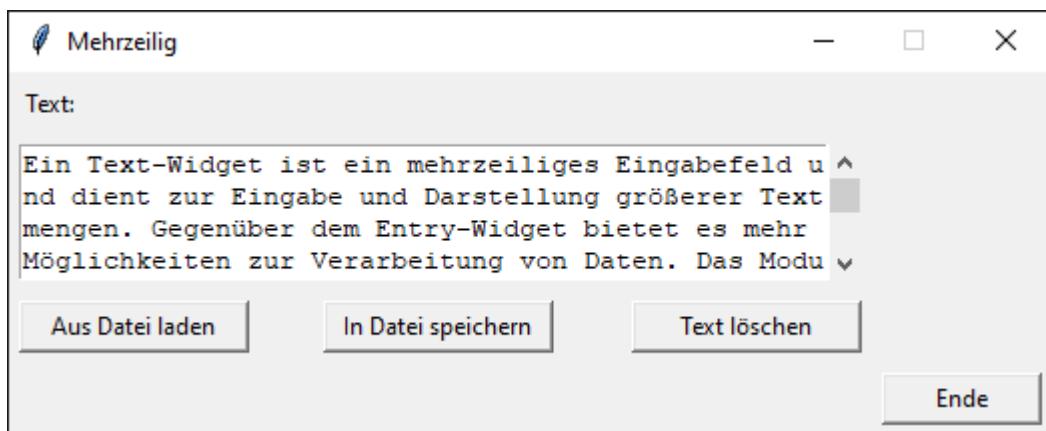


Abbildung 11.7 ScrolledText-Widget mit Inhalt

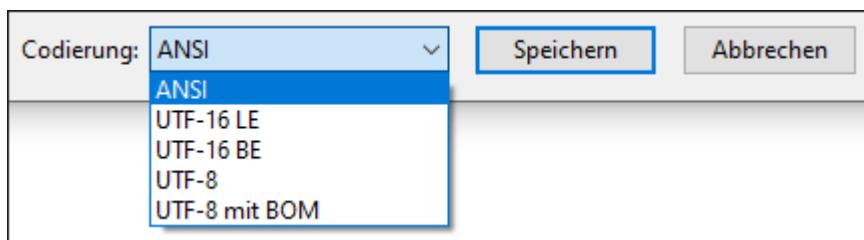


Abbildung 11.8 In ANSI-Kodierung speichern

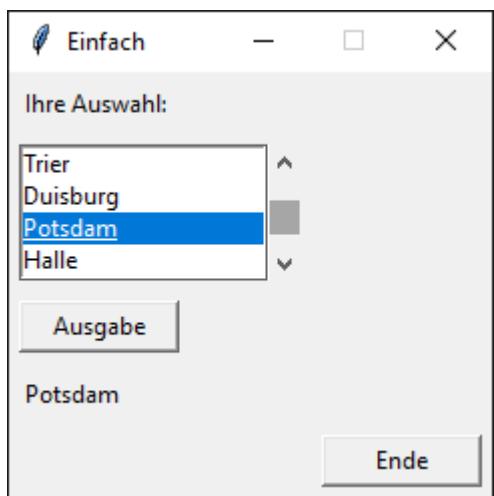


Abbildung 11.9 Liste mit einfacher Auswahl

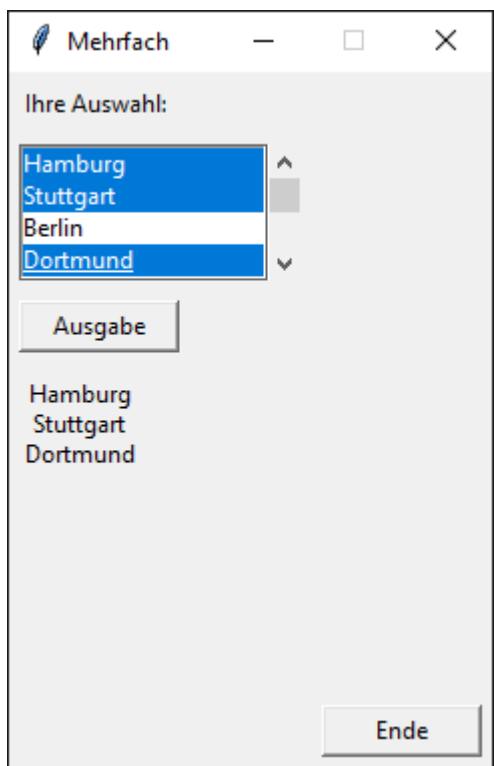


Abbildung 11.10 Liste mit mehrfacher Auswahl

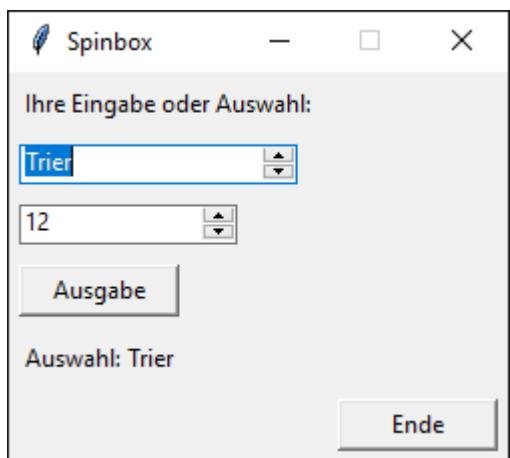


Abbildung 11.11 Auswahl eines Eintrags in der oberen Spinbox

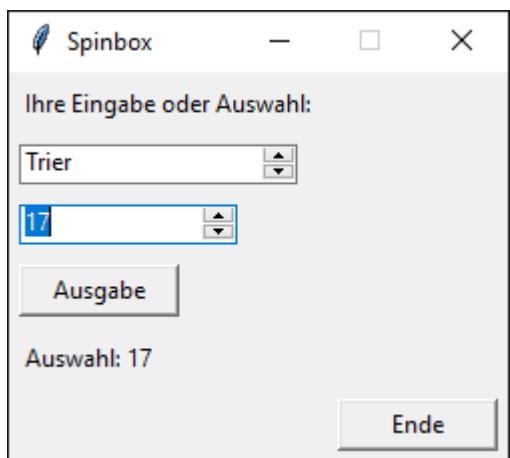


Abbildung 11.12 Auswahl eines Eintrags in der unteren Spinbox

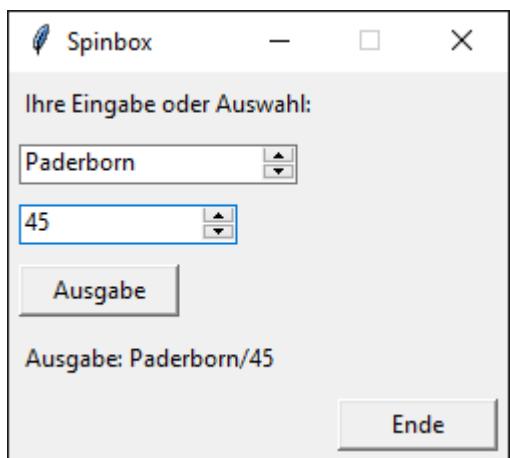


Abbildung 11.13 Eingabe von neuen Einträgen

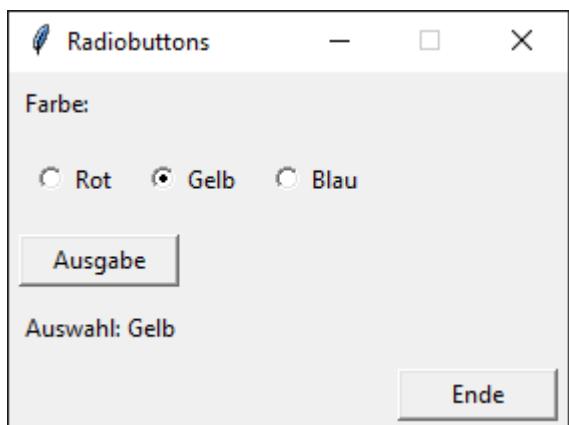


Abbildung 11.14 Gruppe von Radiobuttons

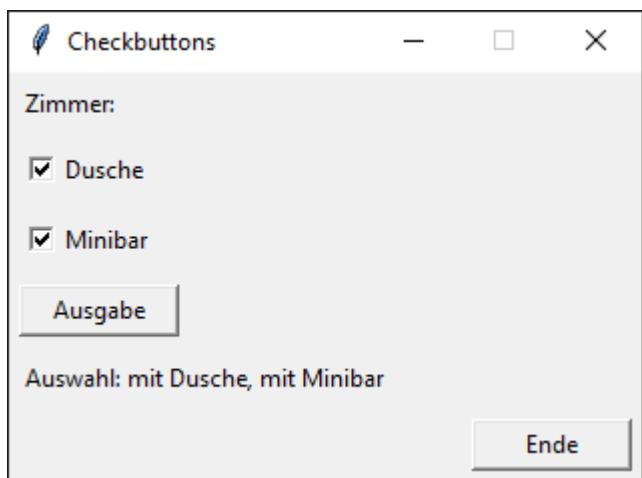


Abbildung 11.15 Nach der Auswahl beider Optionen

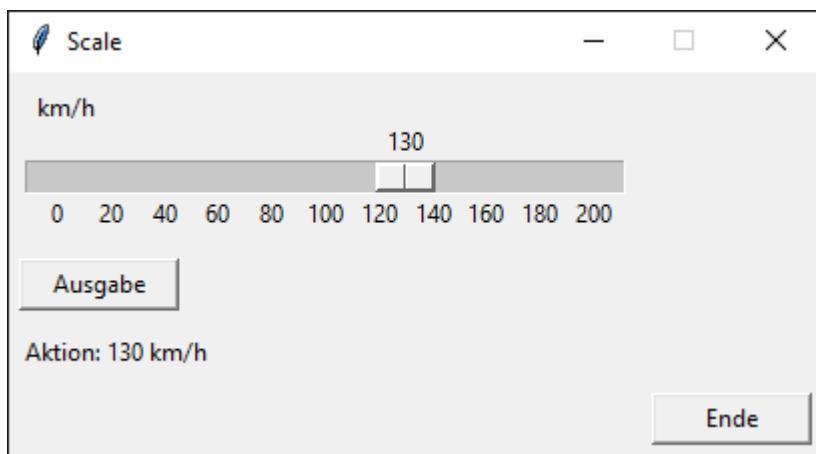


Abbildung 11.16 Nach dem Schieben auf 130

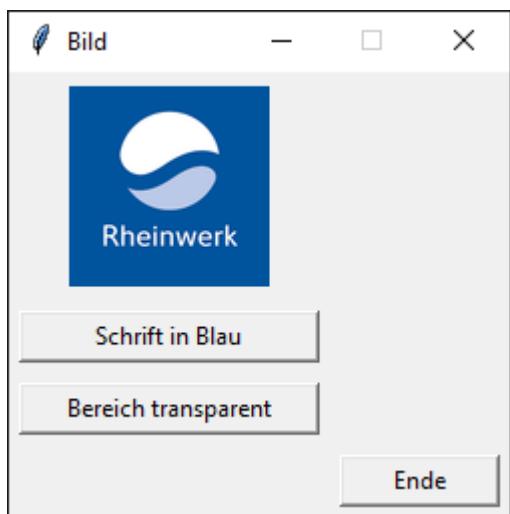


Abbildung 11.17 Nach dem Aufruf des Programms

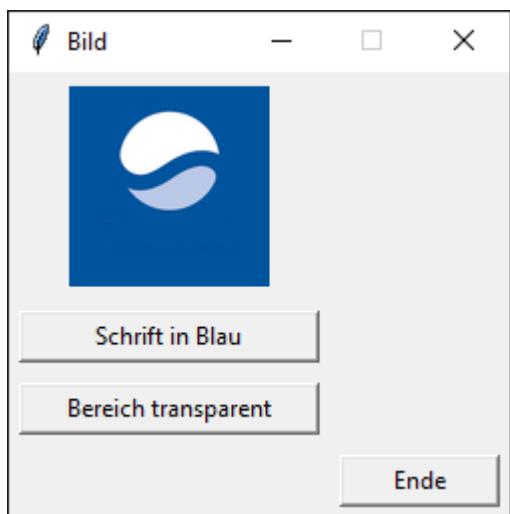


Abbildung 11.18 Nach dem Löschen des Schriftzugs

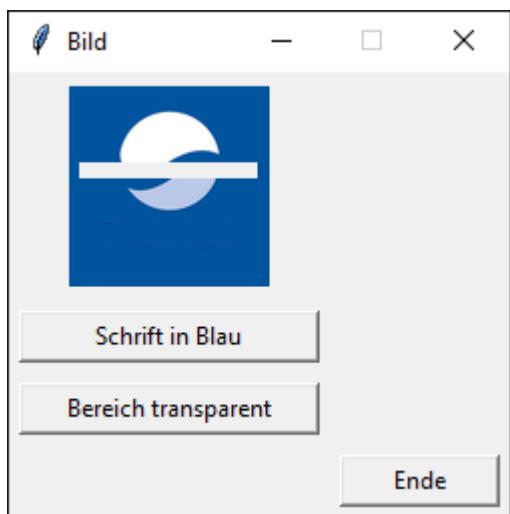


Abbildung 11.19 Nach dem Ändern der Transparenz

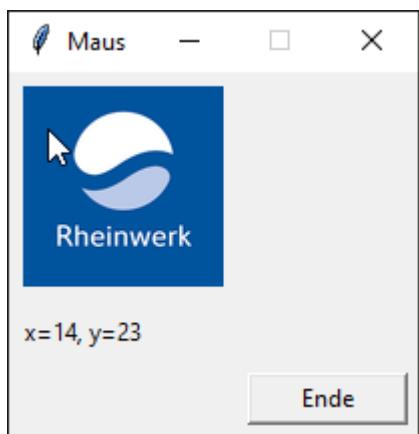


Abbildung 11.20 Maus im Bereich links oben

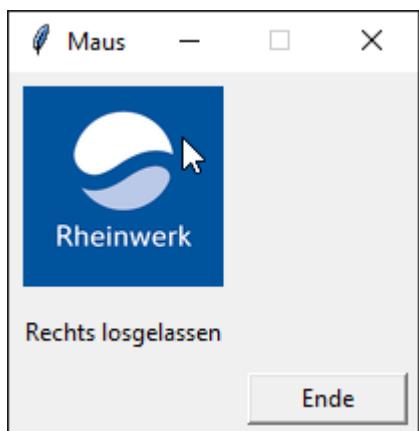


Abbildung 11.21 Rechte Maustaste losgelassen

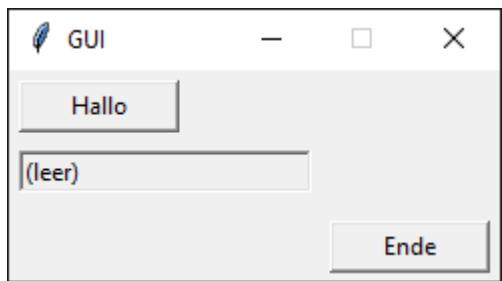


Abbildung 11.22 Fenstergröße und absolute Position

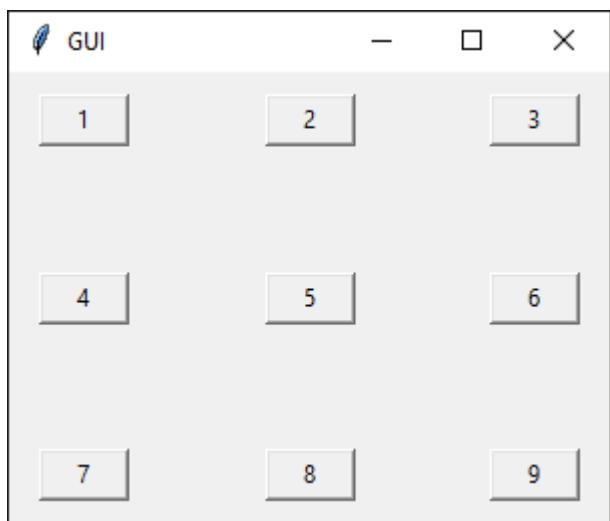


Abbildung 11.23 Relative Position nach dem Start



Abbildung 11.24 Relative Position nach Verkleinerung



Abbildung 11.25 Position ändern

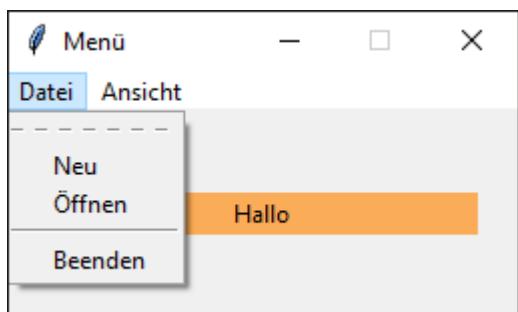


Abbildung 11.26 Menü »Datei«

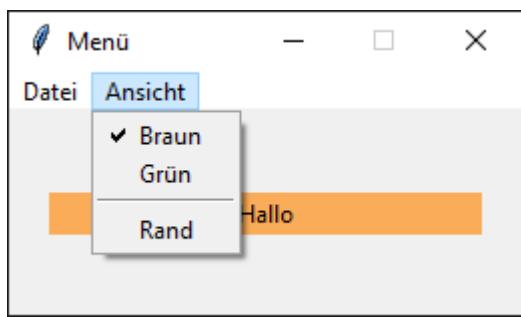


Abbildung 11.27 Menü »Ansicht«

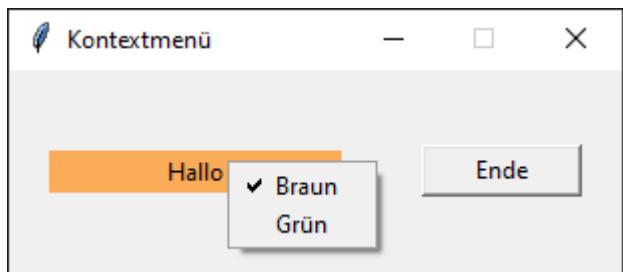


Abbildung 11.28 Kontextmenü für das Label

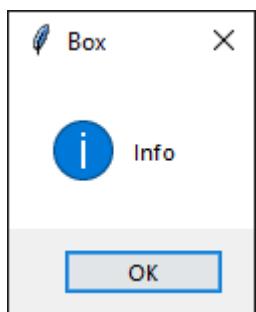


Abbildung 11.29 Information, mit Bestätigung

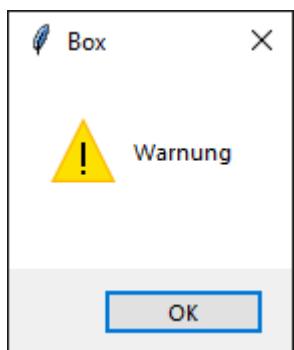


Abbildung 11.30 Warnung, mit Bestätigung

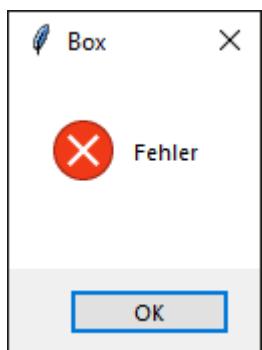


Abbildung 11.31 Fehler, mit Bestätigung

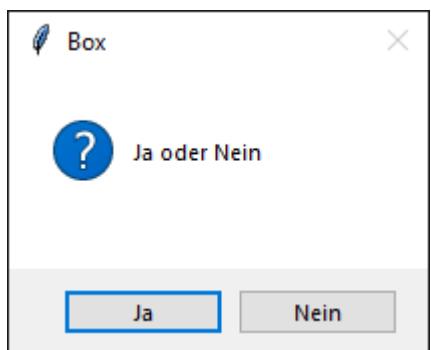


Abbildung 11.32 Mit »Ja« oder »Nein« antworten

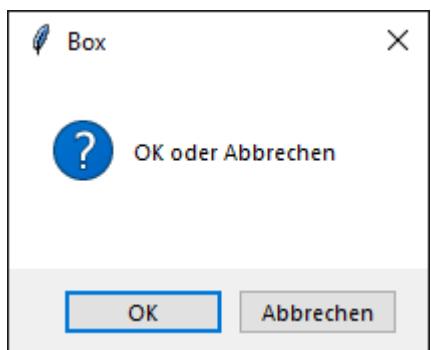


Abbildung 11.33 Mit »OK« oder »Abbrechen« antworten

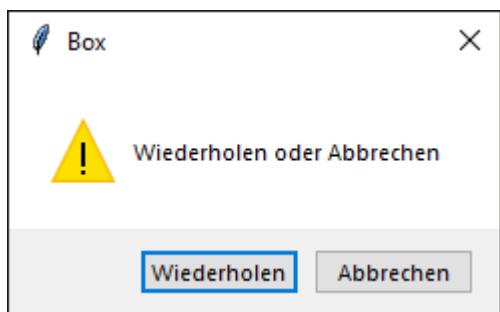


Abbildung 11.34 Mit »Wiederholen« oder »Abbrechen« antworten

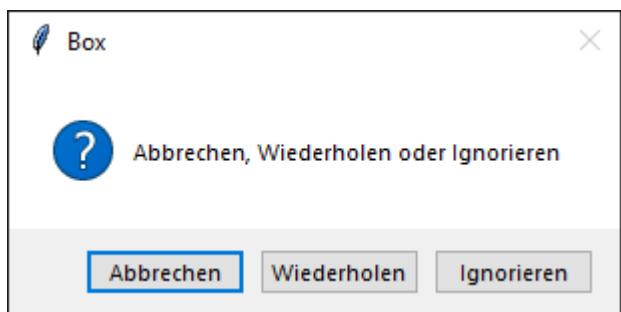


Abbildung 11.35 Eine allgemeine Frage beantworten

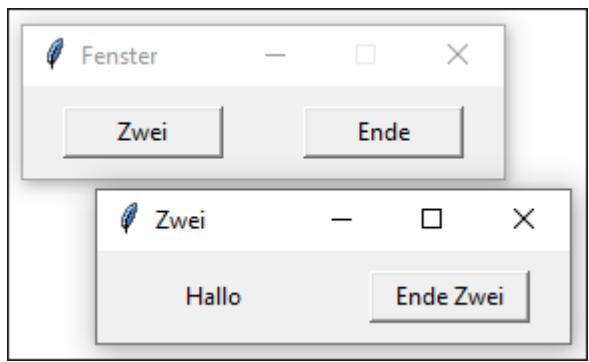


Abbildung 11.36 Anwendungsfenster und zweites Dialogfeld

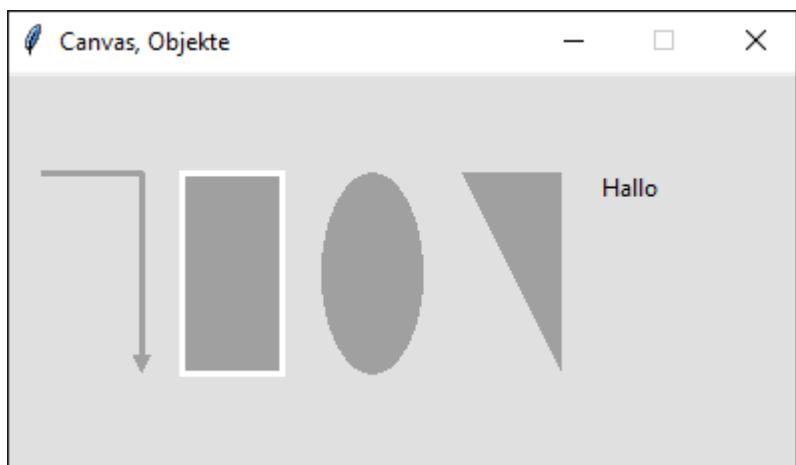


Abbildung 11.37 Zeichnungsobjekte

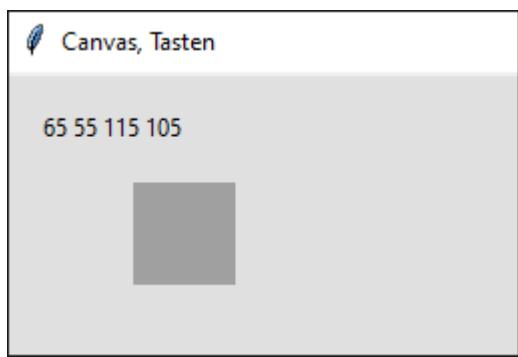


Abbildung 11.38 Zeichnungsobjekte bewegen, Position ausgeben

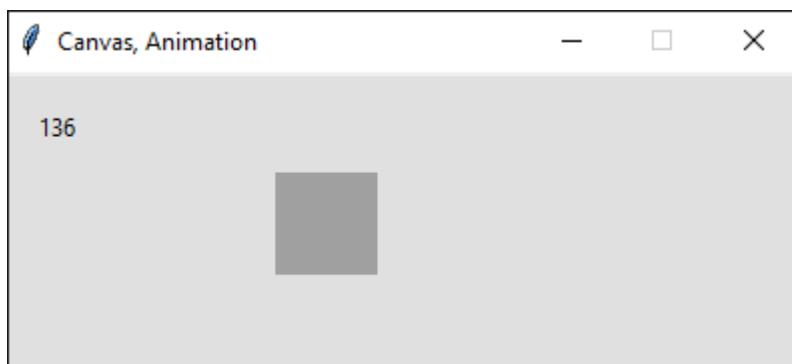


Abbildung 11.39 Animierte Verschiebung



Abbildung 11.40 Kollision von zwei Zeichnungsobjekten

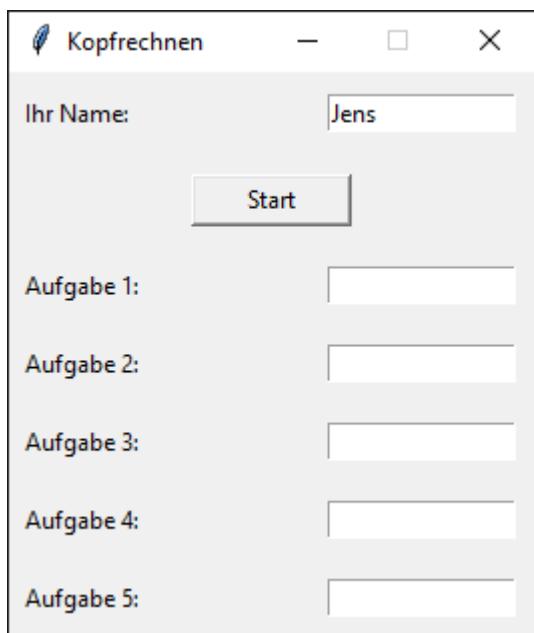


Abbildung 11.41 Nach dem Start

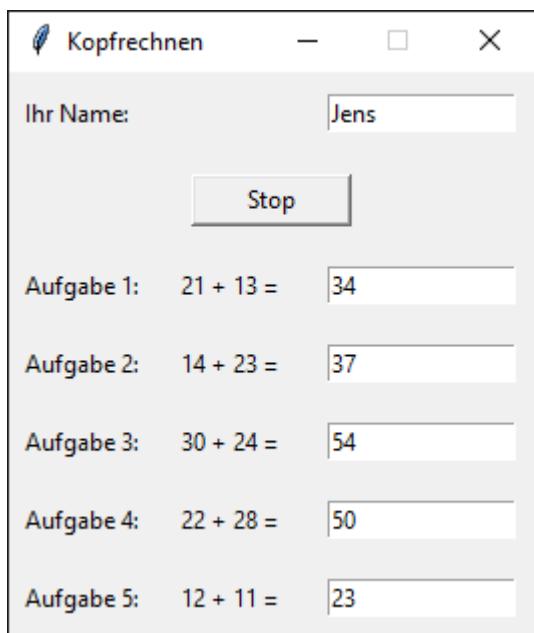


Abbildung 11.42 Aufgaben und Ergebnisse

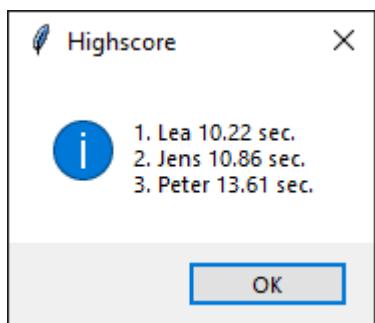


Abbildung 11.43 Highscore-Liste

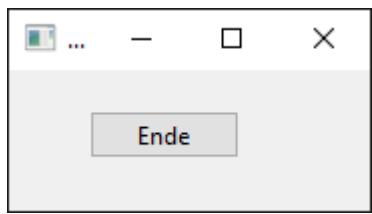


Abbildung 12.1 Erstes Programm mit PyQt

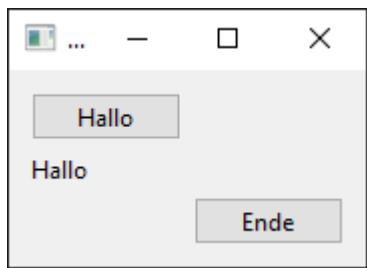


Abbildung 12.2 Anordnung der Widgets in einem Raster

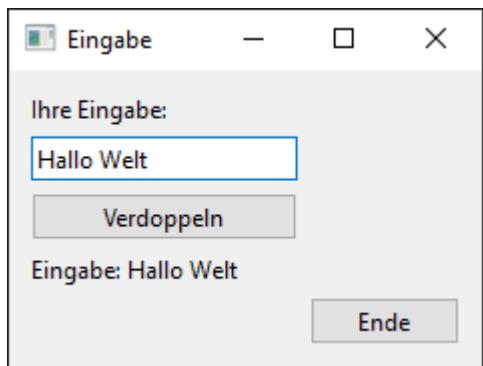


Abbildung 12.3 Ereignis »Änderung des Inhalts«

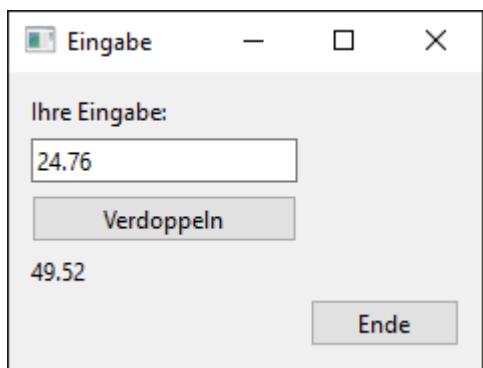


Abbildung 12.4 Betätigung des Buttons »Verdoppeln«

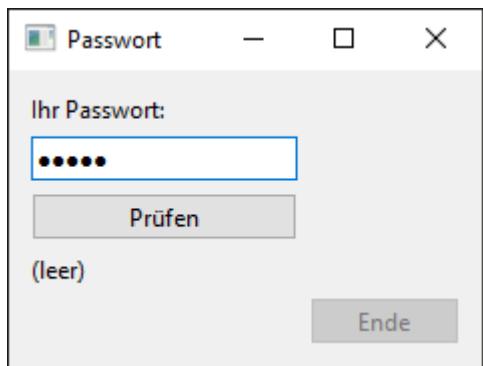


Abbildung 12.5 Versteckte Eingabe

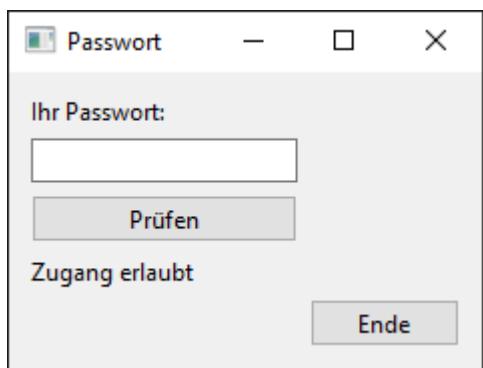


Abbildung 12.6 Aktivierung eines Widgets

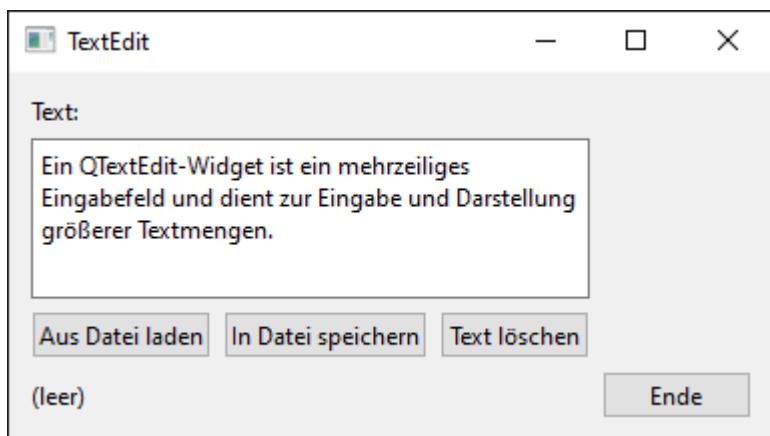


Abbildung 12.7 Mehrzeiliges Eingabefeld

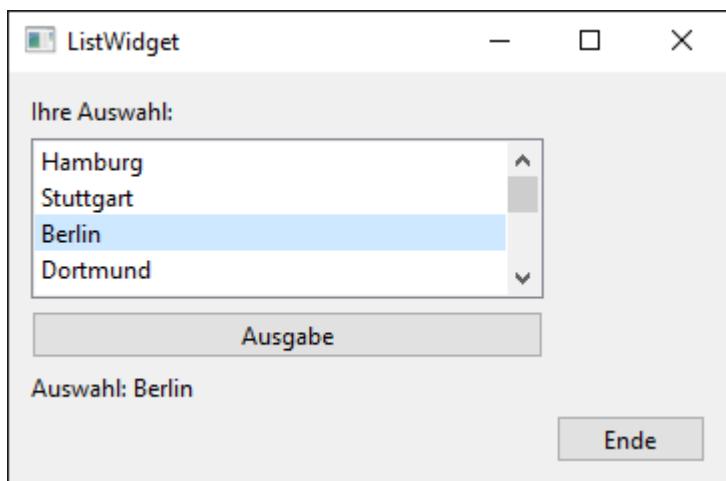


Abbildung 12.8 Liste mit einfacher Auswahl

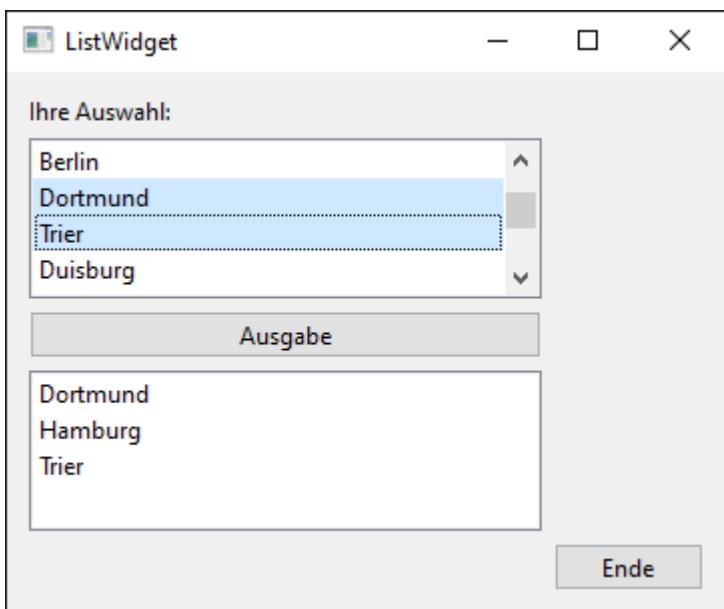


Abbildung 12.9 Liste mit mehrfacher Auswahl

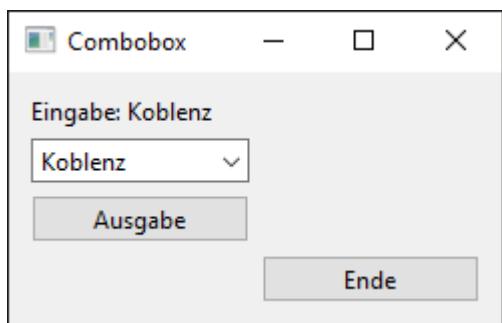


Abbildung 12.10 Eingabe eines Texts

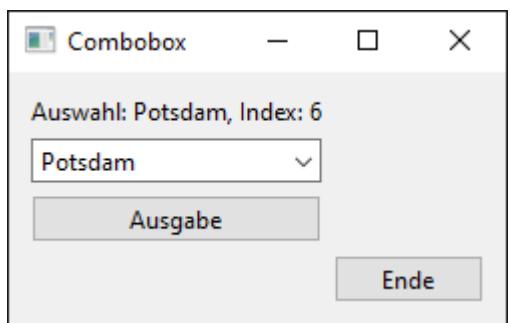


Abbildung 12.11 Auswahl eines Eintrags

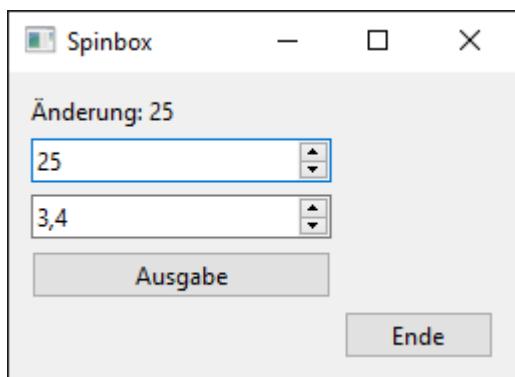


Abbildung 12.12 Wert der ersten Spinbox nach Eingabe

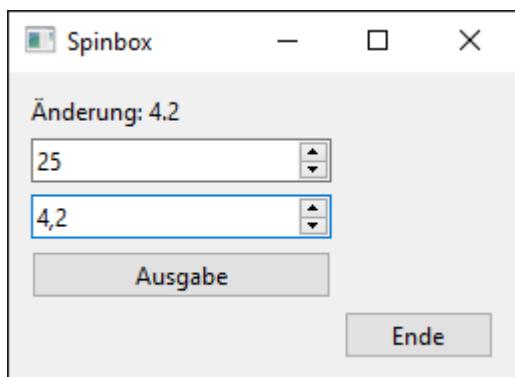


Abbildung 12.13 Wert der zweiten Spinbox nach Auswahl

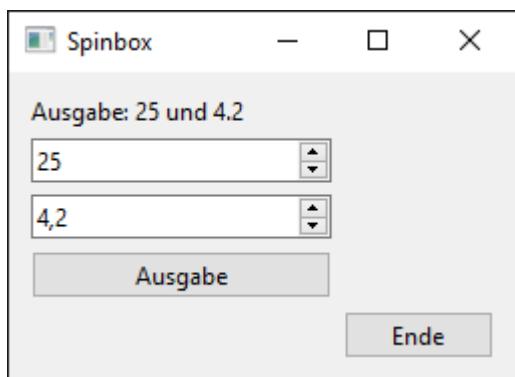


Abbildung 12.14 Werte der beiden Spinboxen

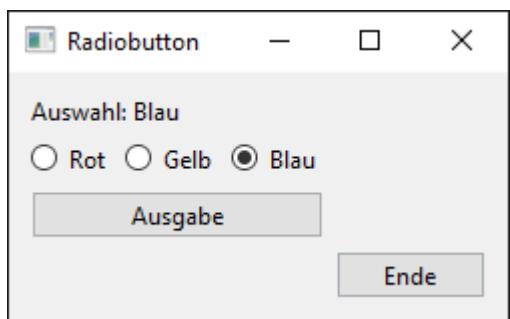


Abbildung 12.15 Markierung eines RadioButtons

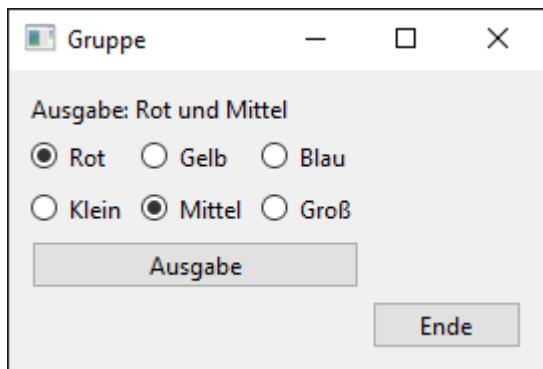


Abbildung 12.16 Mehrere Gruppen von Radiobuttons

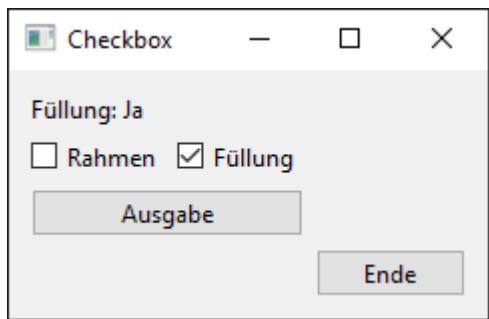


Abbildung 12.17 Nach dem Einschalten der Checkbox »Füllung«

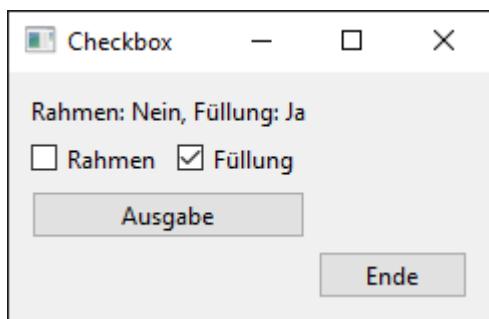


Abbildung 12.18 Ausgabe des Zustands beider Checkboxen

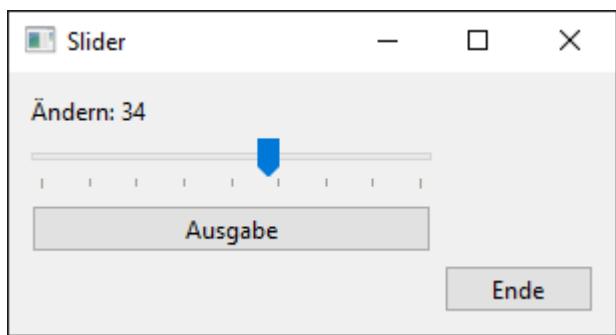


Abbildung 12.19 Nach Änderung des Werts



Abbildung 12.20 Bilder, Formate und Hyperlinks