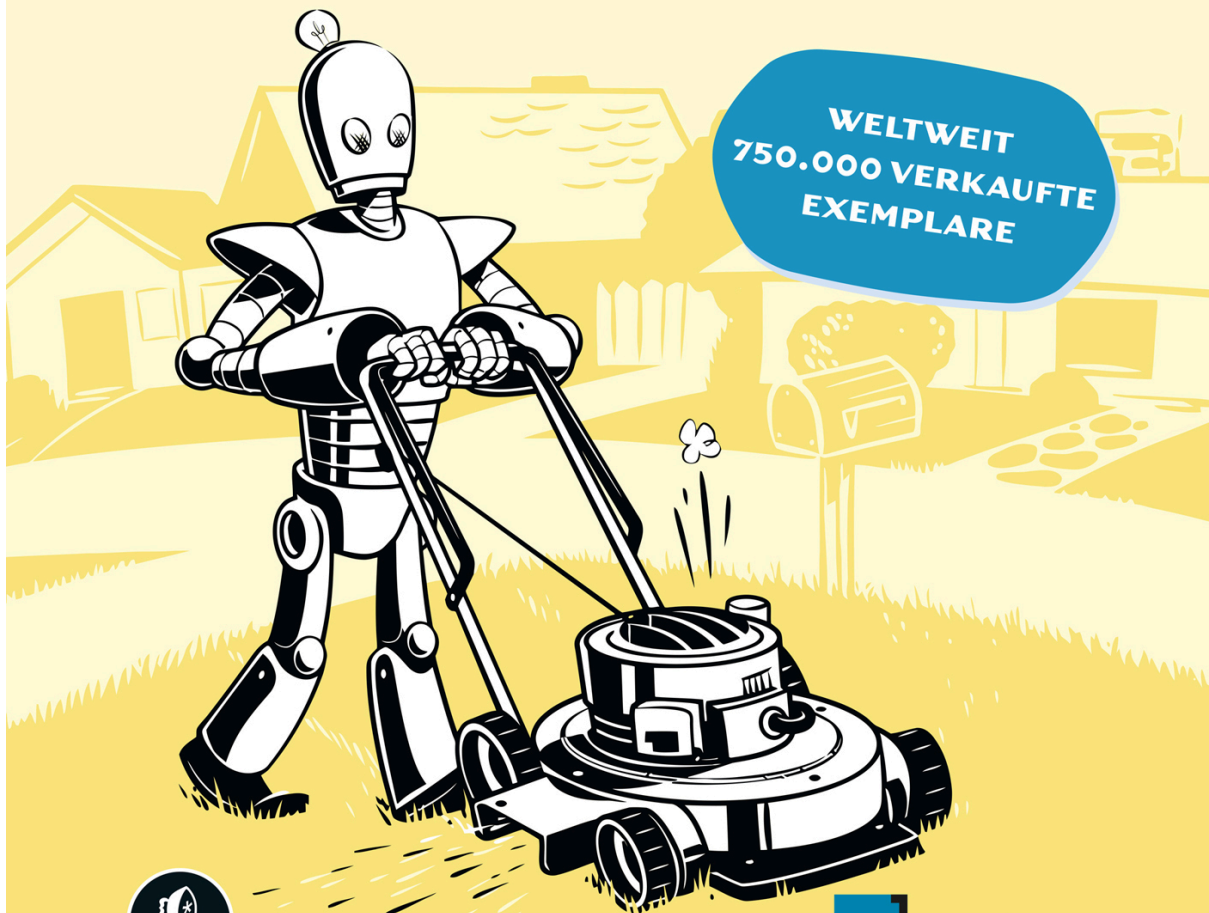


3. AUFLAGE

ROUTINEAUFGABEN MIT PYTHON AUTOMATISIEREN

PRAKTISCHE PROGRAMMIERLÖSUNGEN
FÜR EINSTEIGER*INNEN

AL SWEIGART



WELTWEIT
750.000 VERKAUFTE
EXEMPLARE



dpunkt.verlag

Inhalt

Cover

Hinweise zur Benutzung

Titel

Impressum

Inhalt

Widmung

Über den Autor

Über den technischen Gutachter

Vorwort

Danksagungen

Einführung

Für wen ist dieses Buch geschrieben?

Programmiergepflogenheiten in diesem Buch

Was ist Programmierung?

Was ist Python?

Weit verbreitete Mythen über Programmierung

Programmierer müssen nicht viel Mathematik beherrschen

Sie sind nicht zu alt, um Programmierung zu lernen

KI wird Programmierer nicht ersetzen

Über dieses Buch

Python herunterladen und installieren

Mu herunterladen und installieren

Mu starten

IDLE starten

Die interaktive Shell

Wie Sie Hilfe finden

Programmierfragen schlaue stellen

Neu in der dritten Auflage

Zusammenfassung

Teil I: Programmiergrundlagen

1 Python-Grundlagen

Ausdrücke in die interaktive Shell eingeben

Zeichenketten: Verkettung und Wiederholung

Werte in Variablen speichern

Zuweisungsanweisungen

Variablenamen

Ihr erstes Programm

Das Programm unter der Lupe

Kommentare

Die Funktion print()

Die Funktion input()

Die Begrüßungsnachricht

Die Funktion len()

Die Funktionen str(), int() und float()

Die Funktion type()

Die Funktionen round() und abs()

Wie Computer Daten in Binärzahlen speichern

Zusammenfassung

Übungsaufgaben

2 if-else und Ablaufsteuerung

Boolesche Werte

Vergleichsoperatoren

Boolesche Operatoren

Verwendung von booleschen und Vergleichsoperatoren

Bestandteile von Kontrollstrukturen

Bedingungen

Codeblöcke

Programmausführung

Kontrollstrukturanweisungen

if

else

elif

Ein kurzes Programm: Gegenteil-Tag

Ein kurzes Programm: Unehrllicher Kapazitätsrechner

Zusammenfassung

Übungsfragen

3 Schleifen

while-Schleifen-Anweisungen

Eine lästige while-Schleife

break-Anweisungen

continue-Anweisungen

for-Schleifen und die Funktion range()

Eine gleichwertige while-Schleife

Argumente für range()

Importieren von Modulen

Ein Programm vorzeitig mit sys.exit() beenden

Ein kurzes Programm: Zahl erraten

Ein kurzes Programm: Stein, Schere, Papier

Zusammenfassung

Übungsfragen

4 Funktionen

Funktionen erstellen

Argumente und Parameter

Rückgabewerte und return-Anweisungen

Der Wert »None«

Benannte Parameter

Der Call Stack

Lokaler und globaler Geltungsbereich

Regeln zum Gültigkeitsbereich

Die global-Anweisung

Identifikation des Gültigkeitsbereichs

Ausnahmebehandlung

Ein kurzes Programm: Zigzag

Ein kurzes Programm: Spike

Zusammenfassung

Übungsfragen

Übungsprogramme

Die Collatz-Sequenz

Eingabevalidierung

5 Debugging

Ausnahmen auslösen

Assertions

Loggen

Das Logging-Modul

Protokolldateien

Eine schlechte Praxis: Debuggen mit print()

Loglevel

Deaktiviertes Logging

Der Debugger von Mu

Debugging eines Additionsprogramms

Breakpoints setzen

 Zusammenfassung

 Übungsfragen

 Übungsprogramm: Münzwurf debuggen

6 Listen

 Der Listendatentyp

Indizes

Negative Indizes

Ausschnitte (Slices)

Die Funktion len()

Wertaktualisierungen

Verkettung und Vervielfältigung

del-Anweisungen

 Arbeiten mit Listen

for-Schleifen und Listen

Die Operatoren in und not in

Der Trick der Mehrfachzuweisung

Enumerierung von Listenelementen

Zufällige Auswahl und Anordnung

 Erweiterte Zuweisungsoperatoren

 Methoden

Werte finden

Werte hinzufügen

Werte entfernen

Werte sortieren

Umkehren von Werten

 Kurzschlussauswertung mit booleschen Operatoren

Ein kurzes Programm: Magisches Orakel mit einer Liste

Sequenz-Datentypen

Veränderliche und unveränderliche Datentypen

Der Tupel-Datentyp

Typumwandlung zwischen Liste und Tupel

Referenzen

Argumente

Die Funktionen `copy()` und `deepcopy()`

Ein kurzes Programm: der Matrix-Bildschirmschoner

Zusammenfassung

Übungsfragen

Übungsprogramme

Komma-Code

Münzwurf-Serien

7 Dictionaries und Strukturierung von Daten

Der Dictionary-Datentyp

Vergleich von Dictionary und Liste

Schlüssel und Werte zurückgeben

Überprüfen, ob ein Schlüssel existiert

Standardwerte festlegen

Reale Objekte mithilfe von Datenstrukturen modellieren

Projekt 1: Interaktiver Schachbrett-Simulator

Schritt 1: Das Programm einrichten

Schritt 2: Eine Schachbrettvorlage erstellen

Schritt 3: Das aktuelle Schachbrett ausgeben

Schritt 4: Manipulation des Schachbretts

Verschachtelte Dictionaries und Listen

Zusammenfassung

Übungsfragen

Übungsprogramme

Schach-Dictionary-Validator

Fantasy-Spiel-Inventar

Listen-zu-Dictionary-Beute-Umwandlung

8 Strings und Textbearbeitung

Arbeiten mit Strings

String-Literale

Indizes und Slices

Die in- und not in-Operatoren

f-Strings

Alternativen zu f-Strings: %s und format()

Nützliche String-Methoden

Groß- und Kleinschreibung ändern

Überprüfen von String-Eigenschaften

Den Anfang oder das Ende eines Strings überprüfen

Strings zusammenfügen und aufteilen

Text ausrichten und zentrieren

Entfernen von Leerzeichen

Numerische Codepunkte von Zeichen

Strings kopieren und einfügen

Projekt 2: Aufzählungszeichen zum Wiki-Markup hinzufügen

Schritt 1: Kopieren und Einfügen aus der Zwischenablage

Schritt 2: Zeilen des Textes trennen

Schritt 3: Die bearbeiteten Zeilen verbinden

Ein kurzes Programm: Pig Latin

Zusammenfassung

Übungsaufgaben

Übungsprogramm: Tabellen-Drucker

Teil II: Aufgaben automatisieren

9 Texterkennung mit regulären Ausdrücken

Textmuster ohne reguläre Ausdrücke finden

Textmuster mit regulären Ausdrücken finden

Die Syntax regulärer Ausdrücke

Gruppieren mit Klammern

Escape-Sequenzen verwenden

Zeichen aus alternativen Gruppen abgleichen

Alle Übereinstimmungen zurückgeben

Qualifizierer-Syntax: Welche Zeichen sollen abgeglichen werden?

Zeichenklassen und negative Zeichenklassen verwenden

Kurzschreibweisen für Zeichenklassen verwenden

Alles mit dem Punktzeichen finden

Seien Sie vorsichtig, was Sie suchen

Quantifizierer-Syntax: Wie viele Qualifizierer sollen gefunden werden?

Ein optionales Muster finden

Null oder mehr Qualifizierer finden

Finden von einem oder mehreren Qualifizierern

Finden einer bestimmten Anzahl von Qualifizierern

Gieriges und nichtgieriges Suchen

Alles finden

Zeilenumbruchzeichen finden

Finden am Anfang und Ende eines Strings

Groß-/Kleinschreibungsunabhängiges Suchen

Strings ersetzen

Komplexe Regexes im ausführlichen Modus beherrschen

Kombinieren von re.IGNORECASE, re.DOTALL und re.VERBOSE

Projekt 3: Kontaktinformationen aus umfangreichen Dokumenten extrahieren

Schritt 1: Eine Regex für Telefonnummern erstellen

Schritt 2: Erstellen Sie eine Regex für E-Mail-Adressen

Schritt 3: Alle Übereinstimmungen im Text der Zwischenablage finden

Schritt 4: Fassen Sie die Übereinstimmungen zu einem String zusammen
Ideen für ähnliche Programme

Humre: ein Modul für menschenlesbare Regexes

Zusammenfassung

Übungsaufgaben

Übungsprogramme

Starke Passwörtererkennung

Regex-Version der strip()-Methode

10 Dateien lesen und schreiben

Dateien und Dateipfade

Pfad-Trennzeichen standardisieren

Pfade verbinden

Zugriff auf das aktuelle Arbeitsverzeichnis

Zugriff auf das Home-Verzeichnis

Absolute und relative Pfade angeben

Neue Ordner anlegen

Umgang mit absoluten und relativen Pfaden

Die Bestandteile eines Dateipfads ermitteln

Dateigrößen und Zeitstempel ermitteln

Dateien mithilfe von Glob-Mustern finden

Überprüfung der Pfadgültigkeit

Lesen und Schreiben von Dateien

Dateien öffnen

Den Inhalt von Dateien lesen

In Dateien schreiben

Verwendung von with-Anweisungen

Variablen mit dem shelve-Modul speichern

Projekt 4: Erstellen Sie zufällige Quiz-Dateien

Schritt 1: Speichern der Quizdaten in einem Dictionary

Schritt 2: Erstellen der Quiz-Datei

Schritt 3: Die Antwortoptionen erstellen

Schritt 4: Schreiben des Inhalts der Dateien

Zusammenfassung

Übungsaufgaben

Übungsprogramme

Mad Libs

Regex-Suche

11 Dateien organisieren

Das Modul shutil

Dateien und Ordner kopieren

Verschieben und Umbenennen von Dateien und Ordnern

Dateien und Ordner dauerhaft löschen

Löschen in den Papierkorb

Einen Verzeichnisbaum durchlaufen

Dateien komprimieren mit dem Modul zipfile

ZIP-Dateien erstellen und hinzufügen

ZIP-Dateien lesen

Extrahieren aus ZIP-Dateien

Projekt 5: Sichern Sie einen Ordner in einer ZIP-Datei

Schritt 1: Den Namen der ZIP-Datei festlegen

Schritt 2: Erstellen Sie die neue ZIP-Datei

Schritt 3: Durchlaufen Sie den Verzeichnisbaum

Ideen für weitere Programme

- Zusammenfassung

- Übungsaufgaben

- Übungsprogramme

Selektives Kopieren

Unnötige Dateien löschen

Dateien umnummerieren

Amerikanische Datumsangaben ins europäische Format umwandeln

- 12 Kommandozeilenprogramme entwickeln und bereitstellen

 - Ist es das Programm, das alles wirkt und schafft?

 - Verwendung des Terminals

Die Kommandos cd, pwd, dir und ls

Die Umgebungsvariable PATH

PFAD-Bearbeitung

Die Kommandos »which« und »where«

- Virtuelle Umgebungen

- Python-Pakete mit pip installieren

- Selbsterkennende Python-Programme

- Textbasiertes Programmdesign

Kurze Kommandonamen

Kommandozeilenargumente

Zwischenablage-I/O

Farbiger Text mit Bext

Terminal leeren

Signal- und Textbenachrichtigung

- Ein kurzes Programm: Schneesturm

- Pop-up-Nachrichtenboxen mit PyMsgBox

- Python-Programme bereitstellen

Windows

macOS

Ubuntu Linux

Ein kurzes Programm: Das aktuelle Arbeitsverzeichnis kopieren

Windows

macOS

Ubuntu Linux

Ein kurzes Programm: Zwischenablage-Recorder

Windows

macOS

Ubuntu Linux

Kompilieren von Python-Programmen mit PyInstaller

Zusammenfassung

Übungsaufgaben

Übungsprogramm: Machen Sie Ihre Programme bereit für den Einsatz

13 Web Scraping

HTTP und HTTPS

Projekt 6: Starten Sie ein Programm mit dem browser-Modul

Schritt 1: Ermitteln der URL

Schritt 2: Kommandozeilenargumente verarbeiten

Schritt 3: Inhalt der Zwischenablage abrufen

Ideen für ähnliche Programme

Dateien mit »requests« aus dem Web laden

Webseiten herunterladen

Fehlerprüfung

Heruntergeladene Dateien auf dem Laufwerk speichern

Zugriff auf eine Wetter-API

Abrufen von Breitengrad und Längengrad

Das aktuelle Wetter abrufen

Eine Wettervorhersage abrufen

APIs erkunden

HTML verstehen

Das Format erkunden

Anzeigen des Quelltexts einer Webseite

Die Entwicklertools Ihres Browsers öffnen

HTML-Elemente finden

HTML mit BeautifulSoup parsen

Ein BeautifulSoup-Objekt erstellen

Daten aus den Attributen eines Elements abrufen

Projekt 7: Alle Suchergebnisse öffnen

Schritt 1: Die Suchseite abrufen

Schritt 2: Alle Ergebnisse finden

Schritt 3: Browser für jedes Ergebnis öffnen

Ideen für ähnliche Programme

Projekt 8: XKCD-Comics herunterladen

Schritt 1: Das Programm entwerfen

Schritt 2: Die Webseite herunterladen

Schritt 3: Finden und Herunterladen des Comic-Bilds

Schritt 4: Das Bild speichern und den vorherigen Comic suchen

Ideen für ähnliche Programme

Den Browser mit Selenium steuern

Start eines von Selenium gesteuerten Browsers

Browser-Schaltflächen anklicken

Elemente auf der Seite finden

Elemente auf der Seite anklicken

Formulare ausfüllen und absenden

Sondertasten senden

Den Browser mit Playwright steuern

Starten eines von Playwright gesteuerten Browsers

Browser-Schaltflächen anklicken

Elemente auf der Seite finden

Elemente auf der Seite anklicken

Formulare ausfüllen und absenden

Sondertasten senden

Zusammenfassung

Übungsaufgaben

Übungsprogramme

Bildseiten-Downloader

2048

Link-Verifikation

14 Excel-Tabellen

Excel-Dateien lesen

Eine Arbeitsmappe öffnen

Lesen von Tabellen aus der Arbeitsmappe

Zellen aus Tabellen lesen

Umwandlung zwischen Spaltenbuchstaben und Spaltennummern

Zeilen und Spalten holen

Projekt 9: Zensus-Statistiken erfassen

Schritt 1: Die Daten der Tabellenkalkulation einlesen

Schritt 2: Ausfüllen der Datenstruktur

Schritt 3: Speichern Sie die Ergebnisse in einer Datei

Ideen für vergleichbare Programme

Excel-Dokumente erstellen

Excel-Dateien erstellen und speichern

Erstellen und Entfernen von Tabellen

Werte in Zellen schreiben

Projekt 10: Aktualisieren einer Tabellenkalkulation

Schritt 1: Eine Datenstruktur mit den aktualisierten Informationen anlegen

Schritt 2: Alle Zeilen überprüfen und fehlerhafte Preise anpassen

Ideen für vergleichbare Programme

Die Schriftart von Zellen festlegen

Formeln

Anpassen von Zeilen und Spalten

Festlegen von Zeilenhöhe und Spaltenbreite

Zellen zusammenführen und aufteilen

Fenster fixieren

Diagramme

Zusammenfassung

Übungsfragen

Übungsprogramme

Multiplikationstabellenersteller

Leere-Zeilen-Einfüger

15 Google Sheets

EZSheets installieren und einrichten

Ein neues Google Cloud-Projekt erstellen

Aktivieren der Sheets- und Drive-APIs

Konfiguration des OAuth-Zustimmungsbildschirms

Anmeldedaten erstellen

Anmelden mit der Anmeldedatendatei

Die Anmeldedatendatei widerrufen

Google Sheets-Tabellen

Erstellen, Hochladen und Auflisten von Spreadsheets

Zugriff auf Attribute von Spreadsheet

Herunterladen und Hochladen von Spreadsheets

Spreadsheets löschen

Worksheet-Objekte

Daten lesen und schreiben

Erstellen, Verschieben und Löschen von Tabellen

Tabellenblätter kopieren

Google Formulare

Projekt 11: Schein-Blockchain-Kryptowährungsbetrug

Schritt 1: Überprüfung der fiktiven Blockchain

Schritt 2: Transaktionen durchführen

Arbeiten mit Anfragelimits in Google Sheets

Zusammenfassung

Übungsfragen

Übungsprogramme

Google Forms-Daten herunterladen

Spreadsheets in andere Formate umwandeln

Fehler in einem Spreadsheet erkennen

16 SQLite-Datenbanken

Tabellenkalkulationen vs. Datenbanken

SQLite im Vergleich zu anderen SQL-Datenbanken

Datenbanken und Tabellen erstellen

Verbindung zu Datenbanken herstellen

Tabellen erstellen

Datentypen definieren

Tabellen und Spalten auflisten

CRUD-Datenbankoperationen

Einfügen von Daten in die Datenbank

Daten aus der Datenbank auslesen

Daten in der Datenbank aktualisieren

Daten aus der Datenbank löschen

Transaktionen zurückrollen

Datenbanken fürs Backup sichern

Tabellen ändern und löschen

Mehrere Tabellen mit Fremdschlüsseln verbinden

In-Memory-Datenbanken und Backups

Datenbanken kopieren

SQLite-Anwendungen

Zusammenfassung

Übungsfragen

Übungsprogramme

Katzen-Impfstatus-Prüfer

Datenbank der Gerichtzutaten

17 PDFs und Word-Dokumente

PDF-Dokumente

Text extrahieren

Nachbearbeitung mit KI

Bilder extrahieren

PDFs aus anderen Seiten erstellen

Projekt 12: Ausgewählte Seiten aus vielen PDFs kombinieren

Schritt 1: Alle PDF-Dateien ermitteln

Schritt 2: Jede PDF-Datei öffnen

Schritt 3: Speichern Sie die Ergebnisse

Ideen für vergleichbare Programme

Word-Dokumente

Word-Dokumente lesen

Den vollständigen Text aus einer .docx-Datei extrahieren

Formatierung von Paragraph- und Run-Objekten

Word-Dokumente schreiben

Überschriften hinzufügen

Zeilen- und Seitenumbrüche einfügen

Bilder einfügen

- Zusammenfassung

- Übungsfragen

- Übungsprogramme

PDF-Paranoia

Individuelle Einladungen

- 18 CSV-, JSON- und XML-Dateien

- Das CSV-Format

CSV-Dateien lesen

Datenzugriff in einer for-Schleife

CSV-Dateien schreiben

Verwendung von Tabs anstelle von Kommas

Verarbeitung von Kopfzeilen

- Projekt 13: Entfernen der Kopfzeile aus CSV-Dateien

Schritt 1: Jede Datei durchlaufen

Schritt 2: Lesen der Datei

Schritt 3: Die neue CSV-Datei schreiben

Ideen für vergleichbare Programme

- Vielseitige Klartextformate

Json

Xml

- Zusammenfassung

- Übungsfragen

Übungsprogramm: Excel-zu-CSV-Konverter

19 Zeit messen, Aufgaben planen und Programme starten

Das Modul time

Den Epoch-Timestamp abrufen

Programme anhalten

Projekt 14: Super-Stoppuhr

Schritt 1: Konfigurieren Sie das Programm zur Zeiterfassung

Schritt 2: Rundenzeiten erfassen und ausgeben

Ideen für vergleichbare Programme

Das Modul datetime

Zeitdauer darstellen

Pausieren bis zu einem bestimmten Datum

Umwandlung von datetime-Objekten in Strings

Konvertieren von Strings in datetime-Objekte

Andere Programme aus Python heraus starten

Kommandozeilenargumente an Prozesse übergeben

Ausgabe der gestarteten Kommandos lesen

Ausführen von Task Scheduler, launchd und cron

Dateien mit Standardanwendungen öffnen

Projekt 15: Einfacher Countdown

Schritt 1: Herunterzählen

Schritt 2: Die Sounddatei abspielen

Ideen für vergleichbare Programme

Zusammenfassung

Übungsfragen

Übungsprogramme

Verschönerte Stoppuhr

Freitag-der-13.-Finder

20 Versand von E-Mail, Text und Push-Benachrichtigungen

Die Gmail-API

API aktivieren

E-Mails versenden

E-Mails lesen

Suche nach E-Mails

Anhänge herunterladen

SMS-E-Mail-Gateways

Push-Benachrichtigungen

Benachrichtigungen versenden

Metadaten übertragen

Benachrichtigungen empfangen

Zusammenfassung

Übungsfragen

Übungsprogramme

Regenschirm-Erinnerung

Automatischer Abmelder

Computersteuerung über E-Mail

21 Diagramme erstellen und Bilder manipulieren

Computerbilder Grundlagen

Farben und RGBA-Werte

Koordinaten und Box-Tupel

Bildmanipulation mit Pillow

Arbeiten mit dem Image-Datentyp

Bilder zuschneiden

Bilder in andere Bilder einfügen

Bilder skalieren

Bilder drehen und spiegeln

Einzelne Pixel ändern

Projekt 16: Ein Logo hinzufügen

Schritt 1: Das Logo-Bild öffnen

Schritt 2: Über alle Dateien iterieren

Schritt 3: Die Bilder skalieren

Schritt 4: Logo hinzufügen und Änderungen speichern

Ideen für ähnliche Programme

Zeichnen auf Bildern

Formen

Text

Bilder in die Zwischenablage kopieren und daraus einfügen

Diagramme mit Matplotlib erstellen

Liniendiagramme und Streudiagramme

Balkendiagramme und Torten- oder Kreisdiagramme

Zusätzliche Komponenten

Zusammenfassung

Übungsfragen

Übungsprogramme

Fliesenleger

Foto-Ordner auf der Festplatte identifizieren

Individuelle Sitzkarten erstellen

22 Texterkennung in Bildern

Tesseract und PyTesseract installieren

Windows

macOS

Linux

PyTesseract

Grundlagen der OCR

Vorverarbeitung eines Bildes

Fehlerkorrektur mit großen Sprachmodellen

Texterkennung in nichtenglischen Sprachen

Die NAPS2-Scanner-Anwendung

NAPS2 installieren und einrichten

NAPS2 aus Python heraus ausführen

Eingabe spezifizieren

Zusammenfassung

Übungsfragen

Übungsprogramm: Browser-Text-Scraper

23 Steuerung von Tastatur und Maus

Einrichten von Bedienungshilfen-Apps in macOS

Den Kurs beibehalten

Pausen und Sicherheitsmechanismen

Abmeldungen

Steuerung der Mausbewegung

Den Mauszeiger bewegen

Die aktuelle Position ermitteln

Steuerung der Mausinteraktion

Klicken

Ziehen

Scrollen

Planen Sie Ihre Mausbewegungen

Screenshots aufnehmen

Bilderkennung

Fensterinformationen abrufen

Das aktive Fenster ermitteln

Fenster mit anderen Funktionen ermitteln

Fenster manipulieren

Die Tastatur steuern

Tastatureingabe-Strings senden

Tastennamen angeben

Drücken und Loslassen von Tasten

Tastenkombinationen ausführen

Einrichtung von GUI-Automatisierungsskripten

Anzeigen von Pop-up-Nachrichtefeldern

Zusammenfassung

Übungsfragen

Übungsprogramme

Beschäftigt wirken

Textfelder mit der Zwischenablage auslesen

Einen spielenden Bot entwickeln

24 Text-zu-Sprache- und Spracherkennungses

Text-zu-Sprache-Engine

Sprachausgabe erzeugen

Sprachausgabe als WAV-Dateien speichern

Spracherkennung

Erstellung von Untertiteldateien

Herunterladen von Videos von Websites

Zusammenfassung

Übungsfragen

Übungsprogramme

Sprachausgabe zum Zahlenratespiel hinzufügen

»99 Bottles of Beer« singen

YouTube-Transkriptor

Anhang

A Installation von Drittanbieter-Paketen

pip installieren

pip auffinden

Pip aus virtuellen Umgebungen ausführen

Installation der in diesem Buch verwendeten Pakete

B Antworten auf Übungsfragen

Kapitel 1

Kapitel 2

Kapitel 3

Kapitel 4

Kapitel 5

Kapitel 6

Kapitel 7

Kapitel 8

Kapitel 9

Kapitel 10

Kapitel 11

Kapitel 12

Kapitel 13

Kapitel 14

Kapitel 15

Kapitel 16

Kapitel 17

Kapitel 18

Kapitel 19

Kapitel 20

Kapitel 21

Kapitel 22

Kapitel 23

Kapitel 24

Stichwortverzeichnis

4

Funktionen



Eine Funktion ist wie ein Mini-Programm innerhalb eines Programms. Python stellt verschiedene eingebaute Funktionen bereit, wie etwa die Funktionen `print()`, `input()` und `len()` aus den vorherigen Kapiteln. Sie können jedoch auch eigene Funktionen schreiben. In diesem Kapitel werden Sie eigene Funktionen erstellen, den Call Stack untersuchen, der die Ausführungsreihenfolge der Funktionen im Programm festlegt, und mehr über den Gültigkeitsbereich von Variablen innerhalb und außerhalb von Funktionen erfahren.

Funktionen erstellen

Um besser zu verstehen, wie Funktionen funktionieren, erstellen wir eine solche. Geben Sie dieses Programm in den Dateieditor ein und speichern Sie es unter *helloFunc.py*:

```
def hallo():  
    # Gibt drei Grüße aus  
    print('Guten Morgen!')  
    print('Guten Tag!')  
    print('Guten Abend!')  
hallo()  
hallo()
```

```
print('NOCH EINMAL!')
hallo()
```

Die erste Zeile ist eine `def`-Anweisung, die eine Funktion mit dem Namen `hallo()` definiert. Der Code im Block, der auf die `def`-Anweisung folgt, bildet den Körper der Funktion. Dieser Code wird ausgeführt, wenn die Funktion aufgerufen wird, nicht bei der ersten Definition der Funktion.

Die `hallo()`-Zeilen nach der Funktion sind Funktionsaufrufe. Im Code ist ein Funktionsaufruf einfach der Name der Funktion, gefolgt von Klammern, wobei zwischen den Klammern verschiedene Argumente stehen können. Wenn die Programmausführung diese Aufrufe erreicht, springt sie zur ersten Zeile der Funktion und beginnt dort mit der Ausführung des Codes. Am Ende der Funktion kehrt die Ausführung zu der Zeile zurück, welche die Funktion aufgerufen hat, und fährt anschließend wie gewohnt fort.

Da dieses Programm `hallo()` dreimal aufruft, wird der Code in der Funktion `hallo()` ebenfalls dreimal ausgeführt. Wenn Sie dieses Programm ausführen, sieht die Ausgabe folgendermaßen aus:

```
Guten Morgen!
Guten Tag!
Guten Abend!
Guten Morgen!
Guten Tag!
Guten Abend!
NOCH EINMAL!
Guten Morgen!
Guten Tag!
Guten Abend!
```

Ein Hauptzweck von Funktionen besteht darin, Code zu bündeln, der mehrfach ausgeführt wird. Ohne eine definierte Funktion müssten Sie diesen Code jedes Mal kopieren und einfügen, wenn Sie ihn ausführen möchten; das Programm würde dann so aussehen:

```
print('Guten Morgen!')
print('Guten Tag!')
```

```
print('Guten Abend!')
print('Guten Morgen!')
print('Guten Tag!')
print('Guten Abend!')
print('NOCH EINMAL!')
print('Guten Morgen!')
print('Guten Tag!')
print('Guten Abend!')
```

Generell sollten Sie Codeduplizierungen vermeiden, da Sie bei einer späteren Änderung des Codes – etwa um einen Fehler zu beheben – daran denken müssen, den Code an allen kopierten Stellen zu aktualisieren.

Mit wachsender Programmiererfahrung werden Sie oft damit beschäftigt sein, *Duplizierungen zu entfernen*, also kopierten und eingefügten Code zu beseitigen. Die Entfernung von Duplizierungen macht Ihre Programme kürzer, besser lesbar und einfacher zu aktualisieren.

Argumente und Parameter

Wenn Sie die Funktion `print()` oder `len()` aufrufen, übergeben Sie ihr Werte, sogenannte *Argumente*, indem Sie diese in die Klammern eintragen. Sie können auch eigene Funktionen definieren, die Argumente annehmen. Geben Sie dieses Beispiel in den Dateieditor ein und speichern Sie es als *helloFunc2.py*:

```
def sage_hallo_zu(name): ❶
    # Gibt drei Grüße für den angegebenen Namen aus
    print('Guten Morgen, ' + name) ❷
    print('Guten Tag, ' + name)
    print('Guten Abend, ' + name)
sage_hallo_zu('Alice') ❸
sage_hallo_zu('Bob')
```

Wenn Sie dieses Programm ausführen, sieht die Ausgabe folgendermaßen aus:

```
Guten Morgen, Alice
Guten Tag, Alice
```

Guten Abend, Alice

Guten Morgen, Bob

Guten Tag, Bob

Guten Abend, Bob

Die Definition der `sage_hallo_zu()`-Funktion besitzt einen Parameter namens `name` ❶. Parameter sind Variablen, die Argumente enthalten. Wenn eine Funktion mit Argumenten aufgerufen wird, werden diese Argumente in den Parametern gespeichert. Beim ersten Aufruf der Funktion `sage_hallo_zu()` wird ihr das Argument `'Alice'` ❷ übergeben. Die Programmausführung tritt in die Funktion ein und der Parameter `name` wird automatisch auf `'Alice'` gesetzt, was die `print()`-Anweisung ausgibt ❸. Sie sollten Parameter in Ihrer Funktion verwenden, wenn sie abhängig von den Werten, die Sie beim Funktionsaufruf übergeben, leicht unterschiedliche Anweisungen ausführen soll.

Eine Besonderheit von Parametern ist, dass der im Parameter gespeicherte Wert vergessen wird, sobald die Funktion zurückkehrt. Wenn Sie beispielsweise `print(name)` nach `sage_hallo_zu('Bob')` im vorherigen Programm hinzufügen, würde das Programm einen Fehler anzeigen, da es keine Variable mit dem Namen `name` gibt. Diese Variable wird nach dem Funktionsaufruf `sage_hallo_zu('Bob')` zerstört, sodass sich `print(name)` auf eine `name`-Variable bezieht, die nicht existiert.

Die Begriffe *definieren*, *aufrufen*, *übergeben*, *Argument* und *Parameter* können verwirrend sein. Um deren Bedeutungen zu rekapitulieren, betrachten Sie folgendes Codebeispiel:

```
def sage_hallo_zu(name): ❶
    # Gibt drei Grüße für den angegebenen Namen aus
    print('Guten Morgen, ' + name)
    print('Guten Tag, ' + name)
    print('Guten Abend, ' + name)
sage_hallo_zu('Al') ❷
```

Eine Funktion zu *definieren* bedeutet, sie zu erstellen, ebenso wie eine Zuweisungsanweisung wie `spam = 42` die Variable `spam` erzeugt. Die `def`-Anweisung definiert die Funktion `sage_hallo_zu()` ❶. Der *Aufruf* der erstellten Funktion geschieht mit der Zeile `sage_hallo_zu('Al')` ❷. Das

übergibt die Ausführung an den Beginn des Funktionscodes. Mit diesem Funktionsaufruf wird der Funktion der String 'Al' übergeben. Ein Wert, der in einem Funktionsaufruf übergeben wird, ist ein *Argument*. Argumente werden lokalen Variablen zugewiesen, die *Parameter* genannt werden. Das Argument 'Al' wird dem Parameter name zugewiesen.

Diese Begriffe kann man leicht verwechseln. Wenn Sie sie jedoch auseinanderhalten können, wissen Sie genau, was mit dem Text in diesem Kapitel gemeint ist.

Rückgabewerte und return-Anweisungen

Wenn Sie die Funktion `len()` aufrufen und ihr ein Argument wie 'Hello' übergeben, ergibt der Funktionsaufruf den Ganzzahlwert 5, was der Länge des Strings entspricht, den Sie übergeben haben. Allgemein bezeichnet man den Wert, zu dem ein Funktionsaufruf ausgewertet wird, als den *Rückgabewert* der Funktion.

Wenn Sie eine Funktion mit der `def`-Anweisung erstellen, können Sie den Rückgabewert mit einer `return`-Anweisung festlegen, die aus Folgendem besteht:

- dem `return`-Schlüsselwort
- dem Wert oder Ausdruck, den die Funktion zurückgeben soll

Handelt es sich um einen Ausdruck, ist der Rückgabewert das Ergebnis, zu dem dieser Ausdruck ausgewertet wird. Das folgende Programm definiert beispielsweise eine Funktion, die – abhängig von der als Argument übergebenen Zahl – einen unterschiedlichen String zurückgibt. Geben Sie diesen Code in den Dateieditor ein und speichern Sie ihn als *magic8Ball.py*:

```
import random ❶
def hole_antwort(antwort_zahl): ❷
    # Weissagt eine Antwort basierend auf dem int
    antwort_zahl, 1 bis 9
    if antwort_zahl == 1: ❸
        return 'Es ist sicher'
    elif antwort_zahl == 2:
        return 'Es ist eindeutig so'
    elif antwort_zahl == 3:
```

```

        return 'Ja'
    elif antwort_zahl == 4:
        return 'Antwort unklar, versuche es erneut'
    elif antwort_zahl == 5:
        return 'Frage später erneut'
    elif antwort_zahl == 6:
        return 'Konzentriere dich und frage erneut'
    elif antwort_zahl == 7:
        return 'Meine Antwort ist nein'
    elif antwort_zahl == 8:
        return 'Aussichten nicht so gut'
    elif antwort_zahl == 9:
        return 'Sehr zweifelhaft'
print('Stelle eine Ja-oder-Nein-Frage:')
input('>')
r = random.randint(1, 9) ❹
weissagung = hole_antwort(r) ❺
print(weissagung) ❻

```

Wenn das Programm startet, importiert Python zunächst das `random`-Modul ❶. Anschließend erfolgt die Definition der `hole_antwort()`-Funktion ❷. Da die Funktion nicht sofort aufgerufen wird, wird der Code darin nicht ausgeführt. Als Nächstes ruft das Programm die `random.randint()`-Funktion mit zwei Argumenten auf: 1 und 9 ❸. Diese Funktion ermittelt eine zufällige Ganzzahl zwischen 1 und 9 (einschließlich 1 und 9) und speichert sie in einer Variablen namens `r`.

Jetzt ruft das Programm die `hole_antwort()`-Funktion mit `r` als Argument ❹ auf. Die Programmausführung springt dann an den Beginn dieser Funktion ❸ und legt den Wert von `r` in einem Parameter mit dem Namen `antwort_zahl` ab. Dann gibt die Funktion abhängig vom Wert in `antwort_zahl` einen von vielen möglichen Strings zurück. Die Ausführung kehrt zu der Zeile am Ende des Programms zurück, die ursprünglich `hole_antwort()` ❹ aufgerufen hat, und weist den zurückgegebenen String

einer Variablen namens `weissagung` zu, die anschließend an einen `print()`-Aufruf ⑥ übergeben und auf dem Bildschirm ausgegeben wird.

Beachten Sie, dass Sie Rückgabewerte als Argumente an andere Funktionsaufrufe übergeben können. Dadurch können Sie diese drei Zeilen abkürzen:

```
r = random.randint(1, 9)
weissagung = hole_antwort(r)
print(weissagung)
```

zu dieser einzelnen, aber gleichwertigen Zeile:

```
print(hole_antwort(random.randint(1, 9)))
```

Denken Sie daran, dass Ausdrücke aus Werten und Operatoren bestehen; Sie können einen Funktionsaufruf in einem Ausdruck verwenden, da der Aufruf zu seinem Rückgabewert ausgewertet wird.

Der Wert »None«

In Python steht der Wert `None` für das Fehlen eines Werts. `None` ist der einzige Wert des Datentyps `NoneType`. (Andere Programmiersprachen könnten diesen Wert `null`, `nil` oder `undefined` nennen.) Genauso wie die booleschen Werte `True` und `False` müssen Sie `None` stets mit einem großen `N` schreiben.

Dieser »Wert ohne Wert« kann hilfreich sein, wenn Sie etwas speichern müssen, das in einer Variablen nicht mit einem echten Wert verwechselt werden soll. Ein Anwendungsfall für `None` ist der Rückgabewert von `print()`. Die Funktion `print()` gibt Text auf dem Bildschirm aus und muss, anders als etwa `len()` oder `input()`, keinen Wert zurückgeben. Da jedoch alle Funktionsaufrufe einen Rückgabewert liefern müssen, gibt `print()` `None` zurück. Um dies in der Praxis zu sehen, geben Sie bitte Folgendes in die interaktive Shell ein:

```
>>> spam = print('Hallo!')
Hallo!
>>> None == spam
True
```

Im Hintergrund fügt Python `return None` ans Ende jeder Funktionsdefinition ohne `return`-Anweisung hinzu. Dieses Verhalten ähnelt der Art und Weise, wie

eine `while`- oder `for`-Schleife implizit mit einer `continue`-Anweisung endet. Funktionen geben ebenfalls `None` zurück, wenn Sie eine `return`-Anweisung ohne Wert verwenden (also nur das Schlüsselwort `return` allein).

Benannte Parameter

Python identifiziert die meisten Argumente anhand ihrer Position im Funktionsaufruf. Zum Beispiel ist `random.randint(1, 10)` etwas anderes als `random.randint(10, 1)`. Der erste Aufruf gibt eine zufällige Ganzzahl zwischen 1 und 10 zurück, da das erste Argument das untere Ende des Bereichs bestimmt und das nächste Argument das obere Ende festlegt, während der zweite Funktionsaufruf einen Fehler verursacht.

Andererseits erkennt Python *benannte Parameter* am Namen, der ihnen im Funktionsaufruf vorangestellt wird. Sie werden auch hören, dass benannte Parameter als Schlüsselwortparameter oder Schlüsselwortargumente bezeichnet werden (*keyword arguments*), obwohl sie nichts mit den Python-Schlüsselwörtern, also »keywords«, zu tun haben. Programmierer verwenden benannte Parameter häufig, um optionale Argumente bereitzustellen. Beispielsweise verwendet die Funktion `print()` die optionalen Parameter `end` und `sep`, um festzulegen, welches Trennzeichen am Ende beziehungsweise zwischen den Argumenten ausgegeben wird. Wenn Sie ein Programm ohne diese Argumente ausführen,

```
print('Hallo')
print('World')
```

würde die Ausgabe folgendermaßen aussehen:

```
Hallo
World
```

Die beiden Strings erscheinen auf getrennten Zeilen, da die Funktion `print()` automatisch ein Neue-Zeile-Zeichen '`\n`' an das Ende des Strings anhängt. Sie können jedoch den benannten Parameter `end` setzen, um das neue Zeilenendezeichen durch einen anderen String zu ersetzen. Wenn der Code beispielsweise so aussieht:

```
print('Hallo', end='')
print('World')
```

würde die Ausgabe folgendermaßen aussehen:

HalloWorld

Die Ausgabe erscheint in einer einzigen Zeile, weil Python nach 'Hallo' keine neue Zeile mehr ausgibt. Stattdessen wird ein leerer String ausgegeben. Dies ist nützlich, wenn Sie den Umbruch in eine neue Zeile deaktivieren möchten, der am Ende jedes Funktionsaufrufs von `print()` passiert. Angenommen, Sie möchten die Ergebnisse einer Serie von Münzwürfen – Kopf oder Zahl – ausgeben. Wenn Sie die Ausgabe in einer einzigen Zeile darstellen, wirkt das Ergebnis ansprechender, wie in diesem *coinflip.py*-Programm:

```
import random
for i in range(100): # Führe 100 Münzwürfe aus.
    if random.randint(0, 1) == 0:
        print('K', end=' ')
    else:
        print('Z', end=' ')
print() # Gib am Ende eine neue Zeile aus.
```

Dieses Programm zeigt die K- und Z-Ergebnisse kompakt in einer einzigen Zeile an, anstatt jedes K- oder Z-Ergebnis einzeln pro Zeile auszugeben:

```
Z K Z Z Z K K Z Z Z Z K K K K Z K K Z Z Z Z Z K Z
Z Z Z Z K Z Z Z Z Z K Z K Z K K K Z Z K Z Z Z Z K
Z K K K Z K K Z K Z Z Z Z Z K Z Z K Z Z Z Z K Z K
K K Z Z Z Z K Z Z Z Z K K K Z K Z K K K K Z K K Z
```

Wenn Sie ebenso mehrere String-Werte an `print()` übergeben, trennt die Funktion diese automatisch durch ein einzelnes Leerzeichen. Um dieses Verhalten zu beobachten, geben Sie Folgendes in die interaktive Shell ein:

```
>>> print('Katzen', 'Hunde', 'Mäuse')
Katzen Hunde Mäuse
```

Sie können das standardmäßige Trennzeichen ersetzen, indem Sie dem benannten Parameter `sep` einen anderen String zuweisen. Geben Sie Folgendes in die interaktive Shell ein:

```
>>> print('Katzen', 'Hunde', 'Mäuse', sep=',')
Katzen,Hunde,Mäuse
```

Sie können benannte Parameter auch zu den Funktionen hinzufügen, die Sie selbst schreiben. Dafür müssen Sie jedoch zunächst die Liste und den Dictionary-Datentyp in [Kapitel 6](#) und [Kapitel 7](#) kennenlernen. Merken Sie sich zunächst, dass einige Funktionen optionale benannte Parameter besitzen, die Sie beim Funktionsaufruf angeben können.

Der Call Stack

Stellen Sie sich vor, Sie führen ein ausschweifendes Gespräch mit jemandem. Sie sprechen über Ihre Freundin Alice, was Sie wiederum an eine Geschichte über Ihren Kollegen Bob erinnert. Zuvor müssen Sie jedoch noch etwas zu Ihrer Cousine Carol erklären. Sie beenden die Geschichte über Carol und kehren dann zu Bob zurück. Wenn Sie Ihre Ausführungen zu Bob abgeschlossen haben, sprechen Sie wieder über Alice. Dann werden Sie an Ihren Bruder David erinnert, erzählen eine Geschichte über ihn und kehren anschließend nochmals zur ursprünglichen Geschichte über Alice zurück. Ihr Gespräch folgte einer Stapel-ähnlichen Struktur (engl. *stack*), wie in [Abbildung 4.1](#). In einem *Stack* werden Elemente ausschließlich oben hinzugefügt oder entfernt, und das aktuelle Thema befindet sich stets an der Spitze des Stacks.

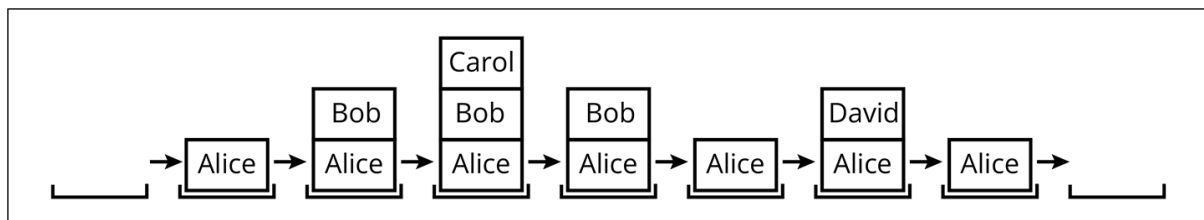


Abb. 4.1 Ihr verschachtelter Gesprächs-Stack

Wie bei Ihrem verschachtelten Gespräch führt ein Funktionsaufruf nicht dazu, dass die Ausführung auf direktem Weg nur an die Spitze einer Funktion gelangt. Python merkt sich, welche Codezeile die Funktion aufgerufen hat, sodass die Ausführung dorthin zurückkehren kann, wenn ein `return`-Statement erreicht wird. Falls die ursprüngliche Funktion weitere Funktionen aufgerufen hat, kehrt die Ausführung zunächst zu *diesen* Funktionsaufrufen zurück, bevor sie aus dem ursprünglichen Funktionsaufruf zurückkehrt. Der Funktionsaufruf an der Spitze des Stacks ist die aktuelle Position der Ausführung.

Öffnen Sie ein Dateieditor-Fenster und geben Sie den folgenden Code ein. Speichern Sie ihn anschließend als `abcdCallStack.py`:

```
def a():  
    print('a() startet')  
    b() ❶
```

```

    d() ❷
    print('a() kehrt zurück')
def b():
    print('b() startet')
    c() ❸
    print('b() kehrt zurück')
def c():
    print('c() startet') ❹
    print('c() kehrt zurück')
def d():
    print('d() startet')
    print('d() kehrt zurück')
a() ❺

```

Wenn Sie dieses Programm ausführen, wird die Ausgabe wie folgt aussehen:

```

a() startet
b() startet
c() startet
c() kehrt zurück
b() kehrt zurück
d() startet
d() kehrt zurück
a() kehrt zurück

```

Wenn `a()` aufgerufen wird ❺, ruft die Funktion `b()` ❶ auf, die wiederum `c()` ❸ aufruft. Die Funktion `c()` ruft nichts weiter auf, sie gibt lediglich `c() startet` ❹ und `c() kehrt zurück` aus, bevor sie zur Aufrufstelle in `b()` ❸ zurückkehrt. Sobald die Ausführung zu dem Code in `b()` zurückkehrt, der `c()` aufgerufen hat, kehrt sie in `a()` dorthin zurück, wo `b()` aufgerufen wurde ❶. Die Ausführung wird in der nächsten Zeile der Funktion `a()` fortgesetzt ❷, die einen Aufruf von `d()` darstellt. Wie die Funktion `c()` ruft auch `d()` nichts weiter auf. Sie gibt lediglich `d() startet` und `d() kehrt zurück` aus, bevor sie zur Aufrufstelle in `a()` zurückkehrt. Da `d()` keinen weiteren Code

enthält, kehrt die Ausführung zu der Stelle in `a()` zurück, an der `d()` aufgerufen wurde ②. Die letzte Zeile in `a()` gibt `a()` kehrt zurück aus, bevor am Ende des Programms die Ausführung zum ursprünglichen Aufruf von `a()` ⑤ zurückkehrt.

Der Call Stack ermöglicht es Python, nach jedem Funktionsaufruf zu wissen, wohin die Ausführung zurückkehren soll. Der Call Stack wird nicht in einer Variable Ihres Programms gespeichert, vielmehr ist er ein Bereich im Arbeitsspeicher Ihres Computers, den Python automatisch im Hintergrund verwaltet. Wenn Ihr Programm eine Funktion aufruft, legt Python ein *Stackframe* (oder *frame object*) oben auf dem Call Stack an. Stackframes speichern die Zeilennummer des ursprünglichen Funktionsaufrufs, damit Python weiß, wohin es zurückkehren soll. Wenn das Programm einen weiteren Funktionsaufruf ausführt, fügt Python ein weiteres Stackframe über dem vorherigen im Call Stack hinzu.

Wenn ein Funktionsaufruf zurückkehrt, entfernt Python ein Stackframe vom oberen Ende des Stacks und setzt die Ausführung an der darin gespeicherten Zeilennummer fort. Beachten Sie, dass Stackframes immer am oberen Ende des Stacks hinzugefügt und entfernt werden und niemals an einer anderen Stelle. [Abbildung 4.2](#) veranschaulicht den Zustand des Call Stacks in `abcdCallStack.py`, während jede Funktion aufgerufen wird und zurückkehrt.

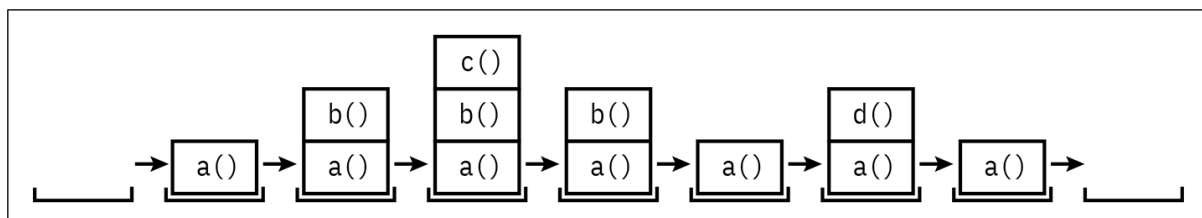


Abb. 4.2 Die Frame-Objekte des Call Stacks, während `abcdCallStack.py` Funktionen aufruft und verlässt.

Das oberste Element des Call Stacks ist die aktuell ausgeführte Funktion. Ist der Call Stack leer, befindet sich die Ausführung auf einer Zeile außerhalb aller Funktionen.

Den Call Stack müssen Sie nicht zwingend kennen, um Programme zu schreiben – er ist ein technisches Detail. Es genügt zu wissen, dass Funktionsaufrufe zur Zeilennummer zurückkehren, von der sie aufgerufen wurden. Ein Verständnis des Call Stacks erleichtert es jedoch, den lokalen und globalen Geltungsbereich zu verstehen, wie im nächsten Abschnitt beschrieben.

Lokaler und globaler Geltungsbereich

Nur Code innerhalb einer aufgerufenen Funktion kann auf die in dieser Funktion zugewiesenen Parameter und Variablen zugreifen. Diese Variablen befinden sich im *lokalen Gültigkeitsbereich* (engl. *local scope*) der Funktion. Im Gegensatz dazu kann Code an beliebiger Stelle im Programm auf Variablen zugreifen, die außerhalb aller Funktionen definiert wurden. Diese Variablen existieren im globalen Gültigkeitsbereich (engl. *global scope*). Eine Variable, die in einem lokalen Gültigkeitsbereich existiert, nennt man *lokale Variable*, während eine Variable im globalen Gültigkeitsbereich als *globale Variable* bezeichnet wird. Eine Variable muss entweder das eine oder das andere sein, sie kann nicht zugleich lokal und global sein.

Stellen Sie sich einen Gültigkeitsbereich als einen Behälter für Variablen vor. Es gibt nur einen globalen Gültigkeitsbereich, der beim Start Ihres Programms angelegt wird. Beendet Ihr Programm die Ausführung, wird der globale Gültigkeitsbereich zerstört und alle darin enthaltenen Variablen werden verworfen. Ein neuer lokaler Gültigkeitsbereich entsteht, sobald ein Programm eine Funktion aufruft. Alle in der Funktion zugewiesenen Variablen existieren im lokalen Gültigkeitsbereich dieser Funktion. Wenn die Funktion zurückkehrt, wird der lokale Gültigkeitsbereich zusammen mit diesen Variablen aufgehoben.

Python verwendet Gültigkeitsbereiche, da so eine Funktion ihre eigenen Variablen anpassen kann und dennoch ausschließlich über ihre Parameter und ihren Rückgabewert mit dem restlichen Programm kommuniziert. Dadurch reduziert sich die Anzahl der Codezeilen, die möglicherweise einen Fehler verursachen. Wenn Ihr Programm ausschließlich aus globalen Variablen bestünde und ein Fehler durch eine Variable mit einem falschen Wert verursacht würde, hätten Sie möglicherweise Schwierigkeiten, die Quelle dieses fehlerhaften Wertes zu ermitteln. Sie könnte von überall im Programm gesetzt worden sein, das Hunderte oder Tausende Zeilen lang sein kann! Tritt der Fehler jedoch in einer lokalen Variablen auf, können Sie Ihre Suche auf eine einzelne Funktion beschränken.

Aus diesem Grund ist es zwar in kleinen Programmen unproblematisch, globale Variablen zu verwenden, doch es ist eine schlechte Angewohnheit, sich bei immer umfangreicheren Programmen auf globale Variablen zu verlassen.

Regeln zum Gültigkeitsbereich

Beachten Sie beim Arbeiten mit lokalen und globalen Variablen die folgenden Regeln:

- Code, der sich im globalen Gültigkeitsbereich außerhalb aller Funktionen befindet, kann keine lokalen Variablen verwenden.

- Code, der sich im lokalen Gültigkeitsbereich einer Funktion befindet, kann keine Variablen aus anderen lokalen Gültigkeitsbereichen verwenden.
- Code in einem lokalen Gültigkeitsbereich kann auf globale Variablen zugreifen.
- Sie können denselben Namen für verschiedene Variablen verwenden, sofern sie sich in unterschiedlichen Gültigkeitsbereichen befinden. Das heißt, es kann eine lokale Variable mit dem Namen spam und eine globale Variable ebenfalls mit dem Namen spam geben.

Lassen Sie uns diese Regeln anhand von Beispielen nachvollziehen.

Code im globalen Gültigkeitsbereich kann keine lokalen Variablen verwenden.

Betrachten Sie den folgenden Code, der beim Ausführen zu einem Fehler führt:

```
def spam():
    eggs = 'sss' ❶
spam()
print(eggs)
```

Die Ausgabe des Programms wird wie folgt aussehen:

```
Traceback (most recent call last):
  Datei "C:/test1.py", line 4, in
    print(eggs)
NameError: name 'eggs' is not defined
```

Der Fehler tritt auf, weil die Variable eggs nur im lokalen Gültigkeitsbereich existiert, der beim Aufruf von spam() erzeugt wird ❶. Sobald die Programmausführung aus spam() zurückkehrt, wird dieser lokale Gültigkeitsbereich zerstört und die Variable eggs existiert nicht mehr. Wenn Ihr Programm dann versucht, print(eggs) auszuführen, gibt Ihnen Python einen Fehler zurück, der besagt, dass eggs nicht definiert ist. Das ist nachvollziehbar, wenn Sie darüber nachdenken: Befindet sich die Programmausführung im globalen Gültigkeitsbereich, existieren keine lokalen Gültigkeitsbereiche, sodass keine lokalen Variablen vorhanden sein können. Deshalb können Sie globale Variablen nur im globalen Gültigkeitsbereich referenzieren.

Code, der sich in einem lokalen Gültigkeitsbereich befindet, kann keine Variablen aus anderen lokalen Gültigkeitsbereichen verwenden.

Python erstellt jedes Mal einen neuen lokalen Gültigkeitsbereich, wenn ein Programm eine Funktion aufruft – auch dann, wenn diese Funktion von einer anderen Funktion aufgerufen wird. Betrachten Sie folgendes Programm:

```
def spam():
    eggs = 'SPAMSPAM'
    bacon() ❶
    print(eggs) # Gibt 'SPAMSPAM' aus ❷
def bacon():
    ham = 'hamham'
    eggs = 'BACONBACON'
    spam() ❸
```

Zuerst ruft das Programm `spam()` auf ❸. Innerhalb dieser Funktion entsteht ein neuer Gültigkeitsbereich mit einer eigenen Variablen `eggs = 'SPAMSPAM'`. Danach wird `bacon()` ❶ aufgerufen. Auch hier legt Python einen neuen Gültigkeitsbereich an. In diesem gibt es zwei Variablen: `ham = 'hamham'` und eine neue, eigene `eggs = 'BACONBACON'`.

Beide `eggs`-Variablen existieren gleichzeitig, aber in verschiedenen Bereichen. Wenn `bacon()` endet, verschwindet seine lokale `eggs`-Variable wieder. Zurück in `spam()` existiert nur noch die dortige `eggs`-Variable mit dem Wert `'SPAMSPAM'`, die ausgegeben wird ❷.

Code, der sich im lokalen Geltungsbereich befindet, kann globale Variablen verwenden.

Bisher habe ich gezeigt, dass Code im globalen Geltungsbereich nicht auf Variablen im lokalen Geltungsbereich zugreifen kann; auch Code in einem anderen lokalen Geltungsbereich kann dies nicht. Betrachten Sie nun das folgende Programm:

```
def spam():
    print(eggs) # Gibt 'GLOBALGLOBAL' aus
eggs = 'GLOBALGLOBAL'
spam()
print(eggs)
```

Da die `spam()`-Funktion keinen Parameter namens `eggs` hat und auch kein Code existiert, der `eggs` einen Wert zuweist, betrachtet Python die Verwendung von `eggs` in dieser Funktion als Referenz auf die globale Variable `eggs`. Deshalb gibt das Programm beim Ausführen `'GLOBALGLOBAL'` aus.

Lokale und globale Variablen können denselben Namen haben.

Technisch gesehen ist es völlig zulässig, denselben Variablennamen sowohl für eine globale Variable als auch für lokale Variablen in unterschiedlichen Gültigkeitsbereichen zu verwenden. Wenn Sie sich das Leben erleichtern wollen, sollten Sie dies jedoch vermeiden. Um zu sehen, was passieren kann, geben Sie den folgenden Code in den Dateieditor ein und speichern Sie diesen als `localGlobalSameName.py`:

```
def spam():
    eggs = 'spam local' ❶
    print(eggs) # Gibt 'spam local' aus
def bacon():
    eggs = 'bacon local' ❷
    print(eggs) # Gibt 'bacon local' aus
    spam()
    print(eggs) # Gibt 'bacon local' aus
eggs = 'global' ❸
bacon()
print(eggs) # Gibt 'global' aus
```

Wenn Sie dieses Programm ausführen, erscheint die folgende Ausgabe:

```
bacon local
spam local
bacon local
global
```

Dieses Programm enthält tatsächlich drei verschiedene Variablen, die jedoch alle `eggs` heißen, was zu Verwirrung führen kann. Die Variablen lauten wie folgt:

- eine Variable namens `eggs`, die im lokalen Gültigkeitsbereich existiert, wenn `spam()` aufgerufen wird ❶
- eine Variable namens `eggs`, die im lokalen Gültigkeitsbereich existiert, wenn `bacon()` aufgerufen wird ❷
- eine Variable namens `eggs`, die im globalen Gültigkeitsbereich existiert ❸

Da diese drei separaten Variablen denselben Namen tragen, kann es schwierig sein, jeweils nachzuvollziehen, welche gerade verwendet wird. Vergeben Sie stattdessen für alle Variablen eindeutige Namen, selbst wenn sie in verschiedenen Gültigkeitsbereichen auftreten.

Die global-Anweisung

Wenn Sie eine globale Variable innerhalb einer Funktion verändern müssen, verwenden Sie die Anweisung `global`. Wenn Sie zu Beginn einer Funktion eine Zeile wie `global eggs` einfügen, weist dies Python an: »In dieser Funktion bezieht sich `eggs` auf die globale Variable, daher soll keine lokale Variable mit diesem Namen erstellt werden.« Geben Sie zum Beispiel den folgenden Code in den Dateieditor ein und speichern Sie ihn als `globalStatement.py`:

```
def spam():
    global eggs ❶
    eggs = 'spam' ❷
eggs = 'global'
spam()
print(eggs) # Gibt 'spam' aus
```

Wenn Sie dieses Programm ausführen, gibt der abschließende `print()`-Aufruf Folgendes aus:

```
spam
```

Da `eggs` am Anfang von `spam()` als `global` deklariert wird ❶, bewirkt die Zuweisung von `eggs` auf `'spam'` ❷ eine Änderung des Werts der globalen Variable `eggs`. Es wird niemals eine lokale `eggs`-Variable erstellt.

Identifikation des Gültigkeitsbereichs

Nutzen Sie diese vier Regeln, um festzustellen, ob eine Variable zum lokalen oder zum globalen Gültigkeitsbereich gehört:

1. Eine Variable im globalen Gültigkeitsbereich (also außerhalb aller Funktionen) ist immer eine globale Variable.
2. Eine Variable in einer Funktion mit einer `global`-Anweisung ist in dieser Funktion immer eine globale Variable.
3. Andernfalls gilt: Wenn eine Funktion eine Variable in einer Zuweisungsanweisung verwendet, handelt es sich um eine lokale Variable.
4. Wenn die Funktion jedoch eine Variable verwendet, jedoch nie in einer Zuweisungsanweisung, handelt es sich um eine globale Variable.

Um diese Regeln besser zu verstehen, finden Sie hier ein Beispielprogramm. Geben Sie den folgenden Code in den Dateieditor ein und speichern Sie ihn unter *sameNameLocalGlobal.py*:

```
def spam():
    global eggs ❶
    eggs = 'spam' # Dies ist die globale Variable.
def bacon():
    eggs = 'bacon' # Dies ist eine lokale Variable.
    ❷
def ham():
    print(eggs) # Dies ist die globale Variable. ❸
eggs = 'global' # Dies ist die globale Variable.
spam()
print(eggs)
```

In der Funktion `spam()` bezieht sich `eggs` auf die globale Variable `eggs`, da die Funktion eine `global`-Anweisung dafür enthält ❶. In `bacon()` ist `eggs` eine lokale Variable, weil die Funktion eine Zuweisungsanweisung für sie enthält ❷. In `ham()` ❸ ist `eggs` die globale Variable, da die Funktion weder eine Zuweisungsanweisung noch eine `global`-Anweisung für diese Variable enthält. Wenn Sie *sameNameLocalGlobal.py* ausführen, wird die Ausgabe wie folgt aussehen:

```
spam
```

Wenn Sie versuchen, in einer Funktion eine lokale Variable zu verwenden, bevor Sie ihr einen Wert zuweisen, wie im folgenden Programm, gibt Python Ihnen

einen Fehler aus. Um dies zu veranschaulichen, geben Sie Folgendes in den Dateieditor ein und speichern Sie es als `sameNameError.py`:

```
def spam():
    print(eggs) # FEHLER!
    eggs = 'spam local' ❶
eggs = 'global' ❷
spam()
```

Wenn Sie das vorherige Programm ausführen, erhalten Sie eine Fehlermeldung:

```
Traceback (most recent call last):
  File "C:/sameNameError.py", line 6, in
    spam()
  File "C:/sameNameError.py", line 2, in spam
    print(eggs) # FEHLER!
UnboundLocalError: local variable 'eggs' referenced
before assignment
```

Dieser Fehler tritt auf, weil Python erkennt, dass es in der `spam()`-Funktion eine Zuweisungsanweisung für `eggs` gibt ❶ und daher jede Erwähnung der Variable `eggs` in `spam()` als lokale Variable betrachtet. Da jedoch `print(eggs)` ausgeführt wird, bevor `eggs` ein Wert zugewiesen wurde, existiert die lokale Variable `eggs` nicht. Python greift in diesem Fall nicht auf die globale `eggs`-Variable zurück ❷.

Der Name des Fehlers, `UnboundLocalError`, kann etwas verwirrend sein. In Python bedeutet *binden* (engl. *binding*) so viel wie zuweisen. Dieser Fehler weist daher darauf hin, dass im Programm eine lokale Variable verwendet wurde, bevor ihr ein Wert zugewiesen war.

Funktionen als »Black Boxes«

Oft müssen Sie bei einer Funktion lediglich deren Eingaben (Parameter) und ihren Ausgabewert kennen; es ist nicht immer notwendig, sich mit den Details des Funktionscodes zu beschäftigen. Wenn Sie auf diese Weise abstrahiert über Funktionen nachdenken, spricht man oft davon, eine Funktion als »Black Box« zu behandeln.

Dieses Konzept ist grundlegend für die moderne Programmierung. In späteren Kapiteln dieses Buches werden Ihnen verschiedene Module mit Funktionen vorgestellt, die von anderen entwickelt wurden. Auch wenn Sie aus Interesse den Quellcode einsehen können, müssen Sie nicht verstehen, wie diese Funktionen intern arbeiten, um sie zu nutzen. Da das Schreiben von Funktionen ohne globale Variablen ratsam ist, müssen Sie sich in der Regel keine Sorgen machen, dass der Funktionscode mit dem restlichen Programm interagiert.

Ausnahmebehandlung

Wenn aktuell ein Fehler oder eine *Ausnahme* (engl. *exception*) in Ihrem Python-Programm auftritt, stürzt das gesamte Programm ab – es »crasht«. Das möchten Sie in realen Programmen vermeiden. Stattdessen soll das Programm Fehler erkennen, sie behandeln und anschließend weiterlaufen.

Betrachten Sie hierzu das folgende Programm, das einen Fehler durch Division durch null enthält. Öffnen Sie ein Dateieditor-Fenster und geben Sie den folgenden Code ein. Speichern Sie die Datei als *zeroDivide.py*:

```
def spam(teile_durch):
    return 42 / teile_durch
print(spam(2))
print(spam(12))
print(spam(0))
print(spam(1))
```

Wir haben eine Funktion namens `spam` definiert, ihr einen Parameter übergeben und anschließend den Wert dieser Funktion mit verschiedenen Parametern ausgegeben, um das Verhalten zu beobachten. Das ist die Ausgabe, wenn Sie den vorangegangenen Code ausführen:

```
21.0
```

```
3.5
```

```
Traceback (most recent call last):
```

```
File "C:/zeroDivide.py", line 6, in
    print(spam(0))
```

```
File "C:/zeroDivide.py", line 2, in spam
```

```
return 42 / teile_durch
```

```
ZeroDivisionError: division by zero
```

Ein `ZeroDivisionError` tritt immer dann auf, wenn Sie versuchen, eine Zahl durch null zu teilen. Anhand der in der Fehlermeldung angegebenen Zeilennummer erkennen Sie, dass die `return`-Anweisung in `spam()` einen Fehler verursacht. In den meisten Fällen weisen Ausnahmen auf einen Bug in Ihrem Code hin, den Sie beheben müssen. Manchmal kann man jedoch Ausnahmen voraussehen und sie behandeln. Beispielsweise lernen Sie in [Kapitel 10](#), wie Sie Text aus Dateien lesen. Wenn Sie einen Dateinamen angeben, der nicht existiert, löst Python eine `FileNotFoundError`-Ausnahme aus. Sie können diese Ausnahme so behandeln, dass Sie den Benutzer bitten, den Dateinamen erneut einzugeben, anstatt das Programm aufgrund einer unbehandelten Ausnahme sofort abstürzen zu lassen.

Fehler lassen sich mit `try`- und `except`-Anweisungen behandeln. Der Code, der potenziell einen Fehler verursachen könnte, wird in einen `try`-Block gesetzt. Die Ausführung des Programms springt zum Beginn des nachfolgenden `except`-Blocks, wenn ein Fehler auftritt.

Sie können den zuvor gezeigten Code zur Division durch null in einen `try`-Block setzen und den Code zur Fehlerbehandlung in einen `except`-Block auslagern:

```
def spam(divide_by):
    try:
        # ZeroDivisionError-Code in diesem Block,
        # führt nicht zum
    Crash:
        return 42 / divide_by
    except ZeroDivisionError:
        # Bei einem ZeroDivisionError wird der Code
        # in diesem Block ausgeführt:
        print('Fehler: Ungültiges Argument.')
print(spam(2))
print(spam(12))
print(spam(0))
print(spam(1))
```

Wenn der Code in einem `try`-Block einen Fehler verursacht, springt die Programmausführung sofort zum Code in den `except`-Block. Nach der Ausführung dieses Codes läuft das Programm wie gewohnt weiter. Tritt im `try`-Block keine Ausnahme auf, wird der Code im `except`-Block übersprungen. Die Ausgabe des vorherigen Programms lautet wie folgt:

```
21.0
3.5
Fehler: Ungültiges Argument.
None
42.0
```

Beachten Sie, dass auch alle Fehler, die bei Funktionsaufrufen innerhalb des `try`-Blocks auftreten, abgefangen werden. Schauen Sie sich das folgende Programm mit `spam()`-Aufrufen im `try`-Block an:

```
def spam(divide_by):
    return 42 / divide_by
try:
    print(spam(2))
    print(spam(12))
    print(spam(0))
    print(spam(1))
except ZeroDivisionError:
    print('Fehler: Ungültiges Argument.')
```

Wenn Sie dieses Programm ausführen, sieht die Ausgabe wie folgt aus:

```
21.0
3.5
Fehler: Ungültiges Argument.
```

Der Grund, warum `print(spam(1))` nie ausgeführt wird, liegt darin, dass das Programm nach dem Wechsel in den Code des `except`-Blocks nicht mehr zum `try`-Block zurückkehrt. Stattdessen fährt es regulär mit dem weiteren Programmablauf nach dem `except`-Block fort.

Ein kurzes Programm: Zigzag

Nutzen wir die bisherigen Programmierkonzepte, um ein kleines Animationsprogramm zu erstellen. Dieses Programm erzeugt ein Zickzack-Muster, das sich hin- und herbewegt, bis Sie es durch Drücken der Stopp-Schaltfläche im Mu-Editor oder durch Betätigen von `Strg-C` anhalten. Beim Ausführen des Programms erscheint die folgende Ausgabe:

```
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
```

Geben Sie den folgenden Quellcode in den Dateieditor ein und speichern Sie die Datei unter *zigzag.py*:

```
import time, sys
einzug = 0 # Wie viele Leerzeichen einrücken
einzug_erhöhen = True # Ob die Einrückung zunimmt
oder nicht
try:
    while True: # Die Hauptprogrammschleife
        print(' ' * einzug, end='')
        print('*****')
        time.sleep(0.1) # Pause für 1/10s.
        if einzug_erhöhen:
            # Erhöhe die Anzahl der Leerzeichen:
            einzug = einzug + 1
            if einzug == 20:
```

```

        # Richtung wechseln:
        einzug_erhöhen = False
    else:
        # Verringern der Anzahl der Leerzeichen:
        einzug = einzug - 1
        if einzug == 0:
            # Richtung wechseln:
            einzug_erhöhen = True
except KeyboardInterrupt:
    sys.exit()

```

Analysieren wir diesen Code Zeile für Zeile, beginnend ganz oben:

```

import time, sys
einzug = 0 # Wie viele Leerzeichen einrücken
einzug_erhöhen = True # Ob die Einrückung zunimmt
oder nicht

```

Zunächst importieren wir die Module `time` und `sys`. Unser Programm verwendet zwei Variablen: Die Variable `einzug` zeichnet auf, wie viele Leerzeichen als Einrückung vor dem Block aus acht Sternchen stehen. Die Variable `einzug_erhöhen` enthält einen booleschen Wert, der angibt, ob die Einrückung zunimmt oder abnimmt:

```

try:
    while True: # Die Hauptprogrammschleife
        print(' ' * einzug, end='')
        print('*****')
        time.sleep(0.1) # Pause für 1/10s.

```

Anschließend fügen wir den restlichen Programmcode in eine `try-Anweisung` ein. Wenn man während der Ausführung eines Python-Programms `Strg-C` drückt, löst Python die Ausnahme `KeyboardInterrupt` aus. Ist keine `try-except-Anweisung` vorhanden, um diese Ausnahme abzufangen, stürzt das Programm mit einer unschönen Fehlermeldung ab. Wir möchten jedoch, dass unser Programm die Ausnahme `KeyboardInterrupt` sauber behandelt,

indem es `sys. exit()` aufruft. (Den Code, der dies ermöglicht, finden Sie in der `except`-Anweisung am Ende des Programms.)

Die `while True`-Endlosschleife wiederholt die Anweisungen im Programm unendlich. Dabei verwenden wir `' ' * einzug`, um die richtige Anzahl von Leerzeichen für die Einrückung auszugeben. Damit nach diesen Leerzeichen nicht automatisch eine neue Zeile ausgegeben wird, übergeben wir auch `end=''` beim ersten `print()`-Aufruf. Ein zweiter `print()`-Aufruf gibt das Band aus Sternchen aus. Wir haben die `time.sleep()`-Funktion bisher noch nicht besprochen; hier reicht es zu wissen, dass sie eine Pause von einer Zehntelsekunde in das Programm einfügt:

```
if einzug_erhöhen:
    # Erhöhe die Anzahl der Leerzeichen
    einzug = einzug + 1
    if einzug == 20:
        # Richtung wechseln:
        einzug_erhöhen = False
```

Als Nächstes passen wir die Anzahl der Einrückungen an, die beim nächsten Ausgeben der Sternchen verwendet wird. Ist `einzug_erhöhen` auf `True`, erhöhen wir `einzug` um 1. Sobald jedoch die Einrückung 20 erreicht, verringern wir die Einrückung:

```
else:
    # Verringern der Anzahl der Leerzeichen:
    einzug = einzug - 1
    if einzug == 0:
        # Richtung wechseln:
        einzug_erhöhen = True
```

Ist `einzug_erhöhen` gerade `False`, möchten wir eins von `einzug` subtrahieren. Sobald die Einrückung 0 erreicht, soll die Einrückung wieder erhöht werden. In jedem Fall springt die Programmausführung zurück an den Anfang der Hauptprogrammschleife, um die Sternchen erneut auszugeben.

Drückt man zu irgendeinem Zeitpunkt, während sich die Programmausführung im `try`-Block befindet, `Strg-C`, löst diese `except`-Anweisung die `KeyboardInterrupt`-Ausnahme aus und verarbeitet sie:

```
except KeyboardInterrupt:  
    sys.exit()
```

Die Programmausführung wechselt in den `except`-Block, führt `sys.exit()` aus und beendet das Programm. So hat man, obwohl die Hauptprogrammschleife unendlich ist, eine Möglichkeit, das Programm zu beenden.

Ein kurzes Programm: Spike

Lassen Sie uns ein weiteres Scroll-Animationsprogramm erstellen. Dieses Programm verwendet String-Replikation und verschachtelte Schleifen, um Zacken (engl. *spikes*) zu zeichnen. Öffnen Sie eine neue Datei in Ihrem Code-Editor, geben Sie den folgenden Code ein und speichern Sie diese als *spike.py*:

```
import time, sys  
try:  
    while True: # Die Hauptprogrammschleife  
        # Zeichnen von Linien mit zunehmender Länge:  
        for i in range(1, 9):  
            print('-' * (i * i))  
            time.sleep(0.1)  
        # Zeichnen von Linien mit abnehmender Länge:  
        for i in range(7, 1, -1):  
            print('-' * (i * i))  
            time.sleep(0.1)  
except KeyboardInterrupt:  
    sys.exit()
```

Wenn Sie dieses Programm ausführen, zeichnet es wiederholt die folgende Zacke:

```
-  
----  
-----  
-----
```



```
print('-' * (i * i))
time.sleep(0.1)
```

Sie können die Werte 9 und 7 in den beiden `for`-Schleifen anpassen, wenn Sie die Breite der Spitze verändern möchten. Der übrige Code funktioniert mit diesen neuen Werten weiterhin einwandfrei.

Zusammenfassung

Funktionen sind das wichtigste Mittel, um Ihren Code in logische Gruppen zu gliedern. Da Variablen in Funktionen in ihren eigenen lokalen Gültigkeitsbereichen existieren, kann der Code einer Funktion die Werte lokaler Variablen in anderen Funktionen nicht direkt beeinflussen. Dies begrenzt die Codeteile, die die Werte Ihrer Variablen ändern können, was beim Debuggen hilfreich ist.

Funktionen sind ein hervorragendes Werkzeug, um Ihren Code zu organisieren. Sie können sich Funktionen wie Blackboxen vorstellen: Sie haben Eingaben in Form von Parametern und Ausgaben in Form von Rückgabewerten, und der Code in ihnen beeinflusst keine Variablen in anderen Funktionen.

In den vorherigen Kapiteln konnte ein einziger Fehler dazu führen, dass Ihr Programm abstürzt. In diesem Kapitel haben Sie `try`- und `except`-Anweisungen kennengelernt, mit denen Code ausgeführt werden kann, wenn ein Fehler erkannt wurde. Dadurch werden Ihre Programme widerstandsfähiger gegenüber häufig auftretenden Fehlern.

Übungsfragen

1. Warum sind Funktionen in Ihren Programmen vorteilhaft?
2. Wann wird der Code in einer Funktion ausgeführt: wenn die Funktion definiert wird oder wenn sie aufgerufen wird?
3. Mit welcher Anweisung wird eine Funktion erstellt?
4. Was ist der Unterschied zwischen einer Funktion und einem Funktionsaufruf?
5. Wie viele globale Gültigkeitsbereiche hat ein Python-Programm? Wie viele lokale Gültigkeitsbereiche gibt es?
6. Was geschieht mit Variablen in einem lokalen Gültigkeitsbereich, wenn der Funktionsaufruf beendet wird?

7. Was ist ein Rückgabewert? Kann ein Rückgabewert Teil eines Ausdrucks sein?
8. Wenn eine Funktion keine `return`-Anweisung enthält, welcher Rückgabewert ergibt sich beim Aufruf dieser Funktion?
9. Wie können Sie eine Variable innerhalb einer Funktion dazu bringen, sich auf die globale Variable zu beziehen?
10. Welcher Datentyp besitzt `None`?
11. Was bewirkt die Anweisung `import heissenalleihrehaustiereeric`?
12. Wenn Sie im Modul `spam` eine Funktion namens `bacon()` haben, wie würden Sie diese nach dem Import von `spam` aufrufen?
13. Wie können Sie verhindern, dass ein Programm beim Auftreten eines Fehlers abstürzt?
14. Was gehört in den `try`-Block? Was gehört in den `except`-Block?
15. Schreiben Sie das folgende Programm in eine Datei mit dem Namen `notrandomdice.py` und führen Sie es aus. Warum gibt jeder Funktionsaufruf dieselbe Zahl zurück?

```
import random
zufallszahl = random.randint(1, 6)
def hole_zufallswurf():
    # Gibt eine zufällige Ganzzahl von 1 bis 6
    zurück
    return zufallszahl
print(hole_zufallswurf())
print(hole_zufallswurf())
print(hole_zufallswurf())
print(hole_zufallswurf())
```

Übungsprogramme

Schreiben Sie zur Übung Programme, die die folgenden Aufgaben ausführen.

Die Collatz-Sequenz

Schreiben Sie eine Funktion mit dem Namen `collatz()`, die einen Parameter mit dem Namen `zahl` besitzt. Ist `zahl` gerade, soll `collatz()` den Wert von `zahl // 2` ausgeben und zurückgeben. Ist `zahl` ungerade, soll `collatz()` das Ergebnis von `3 * zahl + 1` ausgeben und diesen Wert zurückgeben.

Schreiben Sie anschließend ein Programm, das den Benutzer eine Ganzzahl eingeben lässt und fortlaufend `collatz()` auf diese Zahl anwendet, bis die Funktion den Wert `1` zurückgibt. (Erstaunlicherweise funktioniert diese Folge tatsächlich für jede Ganzzahl; früher oder später gelangen Sie mit dieser Sequenz immer zur 1! Selbst Mathematiker wissen nicht genau, warum. Ihr Programm erforscht damit die sogenannte Collatz-Sequenz, die manchmal auch als »das einfachste Problem der Mathematik, das niemand lösen kann«, bezeichnet wird, siehe <https://youtu.be/094y1Z2wpJg>, Veritasium, 2021.)

Denken Sie daran, den Rückgabewert von `input()` mit der Funktion `int()` in eine Ganzzahl umzuwandeln; andernfalls handelt es sich um einen String-Wert. Um die Ausgabe übersichtlicher zu gestalten, sollten die `print()`-Aufrufe, die die Zahlen ausgeben, den benannten Parameter `sep=' '` verwenden, damit alle Werte in einer Zeile erscheinen.

Die Ausgabe dieses Programms könnte beispielsweise wie folgt aussehen:

```
Bitte gib eine Zahl ein:
```

```
3
```

```
3 10 5 16 8 4 2 1
```

Hinweis

Eine Ganzzahl `zahl` ist gerade, wenn `zahl % 2 == 0`, und ungerade, wenn `zahl % 2 == 1`.

Eingabevalidierung

Fügen Sie dem vorherigen Projekt `try`- und `except`-Anweisungen hinzu, um zu überprüfen, ob man einen Nicht-Ganzzahl-String eingegeben hat. Normalerweise löst die Funktion `int()` einen `ValueError`-Fehler aus, wenn ihr ein Nicht-Ganzzahl-String übergeben wird, wie beispielsweise bei `int('lachen')`. Geben Sie im `except`-Block eine Nachricht für den Benutzer aus, dass eine Ganzzahl eingegeben werden muss.

Stichwortverzeichnis

Symbole

?

Glob-Muster [315](#)

Nicht gieriger Mustervergleich [282](#)

.

Aktuelles Verzeichnis [309](#)

.. [309](#)

* [279](#)

()

Gruppen in regulären Ausdrücken [271](#)

[]

Arbeitsblätter [446](#)

Listendefinition [175](#)

Listenelemente abrufen [176](#)

Stringindices und -slices [239](#)

Zeichenklassen [299](#)

{}

Reguläre Ausdrücke [278](#)

*

Glob-Muster [315](#)

Multiplikation [57](#)

Reguläre Ausdrücke [278](#)

Stringwiederholung [57](#)

/

Pfadverkettung [306](#)

[239](#)

+

Addition [57](#)

Listenverkettung [180](#)

Reguläre Ausdrücke [280](#)

Stringverkettung [57](#)

+= [186](#)

*= [186](#)

= [81](#)

== [81](#)
| [273](#), [289](#)
200 [397](#)
404 [397](#)
2048 [441](#)
.docx [557](#)
__file__ [363](#)
.ttf [668](#)

A

Ablaufdiagramme [79](#)
Ablaufsteuerung
 else [89](#)
 if [88](#)
abs() [71](#)
abspath() [309](#)
ACID (Transaktion) [517](#)
activate()
 PyAutoGUI [716](#)
add_break() [567](#)
add_heading() [566](#)
Additionsoperator [57](#)
add_paragraph() [564](#)
add_picture() [568](#)
add_run() [564](#)
Aktuelles Arbeitsverzeichnis [307](#), [354](#)
alert() [724](#)
Alphawert [644](#)
anchor [312](#)
and [83](#)
Anführungszeichen
 Doppelt [236](#)
 Dreifach [238](#), [288](#)
Anhängemodus [321](#)
Anker [312](#)
Anwendungsprogrammierschnittstellen [587](#)
APIs [587](#)
append() [188](#)
Arbeitsblätter
 Aktives Arbeitsblatt [457](#)

- Auf Zellen zugreifen [446](#)
- Aus Arbeitsmappe abrufen [446](#)
- Ausschnitte [449](#)
- Bereiche fixieren [469](#)
- CSV-Dateien [573](#)
- Erstellen [458](#)
- Größe bestimmen [446](#)
- Index [458](#)
- Werte in Zellen schreiben [459](#)
- Zeilenhöhe [467](#)
- Zellen verbinden [468](#)
- Arbeitsmappen
 - Einführung [444](#)
 - Speichern [457](#)
- Archive [341](#)
- Argumente
 - Befehlszeilenargumente [394](#)
 - compile() [289](#)
 - Funktionsaufruf [129](#)
 - Funktionsdefinitionen [135](#)
 - Listen [175](#)
 - Position [135](#)
 - Rückgabewerte als Argumente für andere Funktionen [132](#)
- argv [366](#)
- ASCII-Sortierung [190](#)
- assert [160](#)
- Assertions [160](#)
- attachments [630](#)
- AttributeError [188](#)
- attrs [413](#)
- Aufrufstack [137](#)
- Ausdrücke [52](#)
- Ausnahmen
 - AssertionError [160](#)
 - Auslösen [158](#)
 - Einführung [146](#)
 - Google Tabellen [501](#)
 - Unterschied zu Assertions [160](#)
- automateboringstuff3 [745](#)

B

- Backslash
 - Maskierung [236](#)
 - Pfade [305](#)
- BarChart() [471](#)
- basename() [312](#)
- basicConfig() [162](#), [164](#)
- Beautiful Soup
 - Zweck [411](#)
- Bedingungen [86](#)
- Beendigungscode [614](#)
- Benutzereingaben
 - Groß- und Kleinschreibung [243](#)
 - isX-Methoden [245](#)
 - Strings in Integer umwandeln [67](#)
 - Validieren [155](#)
- Benutzerverzeichnis [309](#)
- bext [368](#)
- Bilder
 - Beschneiden [650](#)
 - Drehen [655](#)
 - Einzelne Pixel ändern [658](#)
 - Formen zeichnen [666](#)
 - Größe [649](#)
 - Größe ändern [655](#)
 - In Word-Dateien einfügen [568](#)
 - Kacheln [651](#)
 - Koordinatensystem [646](#)
 - Kopieren [651](#)
 - Punkte zeichnen [666](#)
 - Spiralen zeichnen [704](#)
- Binärdateien
 - Binärer Schreibmodus [547](#)
 - PDFs [541](#)
 - Shelf-Dateien [323](#)
- Binärzahl [72](#)
- Binnenmajuskel [62](#)
- BitTorrent [640](#)
- Blöcke [86](#), [129](#)
- Bohnenzählamt [504](#)
- Boolesche Operatoren
 - Kurzschlussauswertung [192](#)

Boolesche Werte
 Einführung [80](#)
Box-Objekte [709](#)
break [108](#)
Breakpoint [170](#)
Browser
 Klicks auf Browserschaltflächen simulieren [427](#)
 Mit Selenium steuern [425](#)
 Tabs öffnen [418](#)
Brute-Force-Angriff [572](#)
bs4 [411](#)

C

calc.exe [614](#)
CamelCase [62](#)
Captchas [717](#)
cd [354](#)
cell() [446](#)
Cell-Objekte [449](#)
center() [248](#)
chdir() [307](#)
choice() [185](#)
chr() [250](#)
clear() [495](#)
click()
 PyAutoGUI [703](#)
 Selenium [430](#)
close() [321](#)
 PyAutoGUI [716](#)
Codeduplizierung [129](#)
Collatz-Folge [154](#)
column_index_from_string() [448](#)
compile() [268](#), [289](#)
confirm() [724](#)
connect (SQLite) [512](#)
continue [110](#)
convertAddress() [489](#)
copy() [201](#), [251](#), [334](#)
copy (Modul) [201](#)
copyTo() [495](#)

copytree() [334](#)
Countdown [620](#)
countdown() (PyAutoGUI) [722](#)
create_sheet() [458](#)
createSheet() [493](#)
createSpreadsheet() [493](#)
CREATE TABLE (SQLite) [512](#)
cron [619](#)
crop() [650](#)
CRUD (Idiom) [516](#)
CSS-Selektoren [413](#)
CSV-Dateien
 Aufbau [575](#)
 Auf Google Tabellen hochladen [483](#)
 Einführung [573](#)
 Google-Tabellen im CSV-Format herunterladen [485](#)
 Kopfzeilen [580](#)
 Lesen [575](#)
 Schreiben [578](#)
 Trennzeichen [579](#)
 Umfangreiche Dateien in einer Schleife lesen [577](#)
 Unterschiede zu Excel-Dateien [574](#)
 Vorteile [574](#)
csv (Modul)
 reader-Objekte [575](#)
cuda [747](#)
cwd() [307](#)
CWD [354](#)

D

Dateieditor [62](#)
Dateien
 Binärdateien [317](#)
 Binärer Schreibmodus [399](#)
 CSV-Dateien [573](#)
 CSV-Dateien schreiben [578](#)
 Dateinamen [305](#)
 Heruntergeladene Dateien speichern [399](#)
 Herunterladen [397](#)
 In ZIP-Dateien sichern [344](#)

- Komprimieren [341](#)
- Lesemodus [320](#)
- Lesen [317](#), [320](#)
- Löschen [336](#)
- PDFs [541](#)
- Pfade [303](#)
- Prüfen, ob ein Pfad eine Datei ist [303](#)
- Schreiben [321](#)
- Shelf-Dateien [323](#)
- Standardanwendungen [619](#)
- Textdateien [317](#)
- Umbenennen [334](#)
- Word-Dateien [557](#)
- .xlsx [443](#)
- ZIP-Dateien [341](#)
- Datentypen
 - Boolesche Werte [80](#)
 - datetime [607](#)
 - Dictionaries [211](#)
 - Image [649](#)
 - Listen [175](#)
 - Methoden [187](#)
 - NoneType [134](#)
 - Sequenziell [194](#)
 - timedelta [608](#)
 - Truthy/falsy [113](#)
 - Tupel [197](#)
 - Umwandeln [198](#)
 - Veränderbar/unveränderbar [195](#)
- Datentypen (SQLite) [513](#)
- datetime
 - Datentyp [607](#)
 - datetime() [607](#)
 - datetime-Objekte in Strings umwandeln [611](#)
 - now() [607](#)
 - Strings in datetime-Objekte umwandeln [612](#)
 - timedelta() [608](#)
- Datum (SQLite) [514](#)
- debug() [162](#)
- Debugger [166](#)
- Debugging

- Assertions deaktivieren [160](#)
- Logging [164](#)
- print() [164](#)
- decrypt() [552](#)
- deepcopy() [201](#)
- def [129](#), [135](#)
- Deflate-Algorithmus [341](#)
- del [180](#)
- delete()
 - EZSheets [486](#), [495](#)
- DELETE (SQLite) [516](#)
- delimiter [579](#)
- Dezimal [72](#)
- Dialogfelder [724](#)
- Dictionaries
 - copy() [201](#)
 - DictReader/DictWriter-Objekte [580](#)
 - Echte Kopien [201](#)
 - Einführung [211](#)
 - JSON [589](#)
 - Methoden [211](#)
 - Sortierung [229](#)
 - Vergleich mit Listen [229](#)
 - Verschachtelt [229](#)
 - Vorhandensein von Schlüsseln und Werten prüfen [217](#)
 - Werte abrufen [229](#)
- Dictionary
 - Standardwerte bei fehlendem Schlüssel [217](#)
- DictReader-Objekte [580](#)
- DictWriter-Objekte [580](#)
- dir [354](#)
- dirname() [312](#)
- disable() [164](#), [166](#)
- Division durch null [146](#)
- Doctorow, Cory [265](#)
- Document() [558](#)
- Document-Objekte [558](#)
- Doppelklick [723](#)
- DOTALL [284](#)
- doubleClick() [703](#)
- downloadAs...() [504](#)

downloadAttachment() [630](#)
downloadFolder [630](#)
dragTo()/drag() [703](#)
Draw() [666](#)
drive [312](#)
dumps() [590](#)

E

Eingabeaufforderung
 Mu [44](#)
Eingabevalidierung
 ValueError [155](#)
Einrückung
 Blöcke [86](#)
elif [91](#)
ellipse() [667](#)
Ellipsen [667](#)
else [91](#)
Elternordner [309](#), [312](#)
E-Mail
 Attribute von Nachrichten [628](#)
 Gmail [626](#)
 Kopien [627](#)
 Lesen [628](#)
 Senden [627](#)
EMAIL_ADDRESS [627](#)
encrypt() [552](#)
Endlosschleife [108](#), [110](#)
endswith() [246](#)
Entwicklertools [408](#)
enumerate() [185](#)
Epochen-Zeitstempel [602](#)
Erweiterte Zuweisungsoperatoren [186](#)
Escapen
 Reguläre Ausdrücke [271](#)
Escapezeichen
 CSV-Dateien [575](#)
Excel
 Dateien öffnen [446](#)
 Excel-Dateien schreiben [457](#)

- Excel-Tabellen auf Google Tabellen hochladen [493](#)
- Formeln [465](#)
- Google-Tabellen im Excel-Format herunterladen [485](#)
- Schrift [463](#)
- Spaltenbezeichnungen [446](#), [448](#)
- Zellen aufteilen [468](#)
- except [146](#), [158](#)
- Exception() [158](#)
- Exception-Objekte [158](#)
- Exceptions [146](#)
- exists() [316](#), [317](#)
- expand [655](#)
- extract() [342](#)
- extractall() [343](#)
- extract_text() [542](#)
- EZGmail
 - Installieren [626](#)
 - Textnachrichten über SMS-E-Mail-Gateways senden [631](#)
 - Zweck [626](#)
- EZSheets
 - Installieren [477](#)
 - Komplette Spalten und Zeilen lesen und schreiben [490](#)
 - Tabellen aktualisieren [484](#)
 - Tokendateien [480](#)
 - Zweck [477](#)

F

- FailSafeException [699](#)
- Fakultät [162](#)
- False [80](#), [110](#)
- Falsy [113](#)
- Farben
 - von GUI-Elementen ermitteln [706](#)
 - RGBA-Werte [644](#)
- Farbiger Text [368](#)
- Fehlermeldungen
 - Eigene [158](#)
 - Hilfe finden [45](#)
 - Statuscodes [398](#)
 - SyntaxError [52](#)

- Verhindern [146](#)
- Fenster
 - Aktiv [704](#)
 - Aktives Fenster ermitteln [713](#)
 - Aktivieren [715](#), [718](#)
 - Attribute [713](#)
 - Größe ändern [715](#)
 - Maximieren/minimieren [715](#)
 - Status [715](#)
 - Titel [714](#)
- fetchall (SQLite) [520](#)
- FileNotFoundError [307](#)
- File-Objekte
 - Zurückgeben [320](#)
- findall() [276](#), [292](#)
- find_element(s)_* [428](#)
- Fließkommazahlen
 - Abrunden [67](#)
 - In Strings umwandeln [67](#)
 - Runden [606](#)
- FLIP_LEFT_RIGHT/_TOP_BOTTOM [657](#)
- float() [67](#)
- Flussdiagramme *siehe Ablaufdiagramme* [79](#)
- Flusssteuerung
 - Blockierung [602](#)
- Fokus [704](#)
- Foreign Key (SQLite) [532](#)
- Formatierung
 - Attribute von Run-Objekten [560](#)
 - Eigene Formate [557](#)
 - Formatvorlagen [560](#)
 - Run-Objekte [557](#)
 - Zeitangaben [611](#)
- Formulare
 - Automatisch ausfüllen [718](#)
 - Google Formulare [503](#)
 - Mit Selenium ausfüllen [430](#)
 - Senden [718](#)
 - Tabulator [718](#)
 - Textfelder lesen [727](#)
- for-Schleifen

- Einführung [114](#)
- Gleichwertige while-Schleife [117](#)
- Iteration über Dictionaries [215](#)
- Listen als Bereich [182](#)
- Frameobjekte [137](#)
- freeze_panes [469](#)
- Fremdschlüssel (SQLite) [532](#)
- from import [118](#)
- fromtimestamp() [607](#)
- F-Strings [242](#)
- Füllzeichen [248](#)
- Funktionen
 - Aufrufen [64](#), [129](#)
 - Blackbox [143](#)
 - Debugging [166](#)
 - Definieren [129](#)
 - Funktionen aus Modulen nutzen [118](#)
 - Globale Variablen ändern [143](#)
 - Gültigkeitsbereiche [139](#)
 - Methoden [187](#)
 - Ohne explizite return-Anweisung [134](#)
 - Rückgabewerte [132](#)
 - Rumpf [129](#)
 - Zweck [129](#)

G

- Gauß, Carl Friedrich [114](#)
- Gecko [426](#)
- get() [215](#)
 - Dateien herunterladen [397](#)
 - Tag-Objekte [415](#)
- getActiveWindow() [713](#)
- getAllTitles() [714](#)
- getAllWindows() [714](#)
- getColor() [644](#)
- getColumn() [489](#)
- get_column_letter() [448](#)
- getColumnLetterOf() [489](#)
- getColumnNumberOf() [489](#)
- getcwd() [307](#)

- getinfo() [342](#)
- getpixel() [658](#)
- getRows() [492](#)
- getsize() [303](#)
- getText() [413](#), [559](#)
- getWindowsAt() [714](#)
- getWindowsWithTitle() [714](#)
- Gleichheitsoperator [81](#)
- glob() [315](#)
- global [143](#)
- Globaler Gültigkeitsbereich [139](#)
- Glob-Muster [315](#)
- Gmail
 - API aktivieren [626](#)
 - Einschränkungen [627](#)
 - EZGmail [626](#)
- Goethe, Johann Wolfgang von [699](#)
- Google Tabellen
 - Ganzes Tabellenblatt aktualisieren [489](#)
 - Grenzwerte [489](#)
 - Leistung [489](#)
 - Limits [501](#)
 - Spaltenbezeichnungen [489](#)
 - Spreadsheet-Objekte [487](#)
 - Tabelle anlegen [493](#)
 - Tabellenattribute [495](#)
 - Tabellen auflisten [493](#)
 - Tabellenblätter [487](#)
 - Tabellen herunterladen [485](#)
 - Tabellen löschen [486](#)
 - Zellenadressierung [489](#)
 - Zugriff aktivieren [480](#)
 - Zugriff widerrufen [481](#)
- GPU [747](#)
- Groß- und Kleinschreibung
 - Reguläre Ausdrücke [286](#)
 - Schreibung von Strings ändern [243](#)
 - Variablennamen [61](#)
- group() [268](#), [271](#)
- groups() [271](#)
- GUI-Automatisierung

Einführung [697](#)
Probleme vermeiden [722](#)
Skript abbrechen [699](#)
Ziehen mit der Maus [703](#)
Zugriff auf macOS [698](#)
Gültigkeitsbereiche
Funktionen [139](#)

H

herunterladen, Videos [737](#)
Hintergrundfarbe [649](#)
home() [309](#)
hotkey() [721](#)
HTML
Attribute [406](#), [413](#)
Beautiful-Soup-Objekt erstellen [412](#)
CSS-Selektoren [413](#)
Durchsuchen [411](#)
Elemente finden [409](#)
Google-Tabellen im HTML-Format herunterladen [485](#)
Grundlagen [406](#)
Quellcode anzeigen [407](#)
Reguläre Ausdrücke [408](#)
Tags [406](#)

I

IDLE [43](#)
if-Anweisung
Block [88](#)
Garantierte Ausführung eines Blocks [91](#)
Reihenfolge von elif-Anweisungen [91](#)
IGNORECASE [286](#)
Image [647](#)
ImageColor [645](#)
ImageDraw [666](#)
ImageFont [668](#)
ImageNotFoundException [709](#)
Image-Objekte [649](#)
Image.open() [688](#)
import [118](#)

in [183](#), [194](#)
index() [188](#)
IndexError [176](#)
Index (SQLite) [525](#)
Indizes
 Listenelemente [176](#)
 Negative Indizes [178](#)
 Tabellenblätter [487](#)
init()
 EZGmail [626](#)
input() [65](#)
insert() [188](#)
INSERT (SQLite) [516](#)
int() [67](#), [155](#)
Integer
 Unveränderbarkeit [198](#)
Interaktive Shell [44](#), [51](#)
is_absolute() [309](#)
isActive [715](#)
is_dir() [316](#)
isdir() [317](#)
isEncrypted [552](#)
is_file() [316](#)
isfile() [317](#)
islower() [243](#)
isMaximized [715](#)
isMinimized [715](#)
isupper() [243](#)
isX-Stringmethoden [245](#)
items() [215](#)
iter_content() [399](#)

J

join() [247](#)
Jokerzeichen [274](#)
JSON
 Einführung [573](#), [587](#)
 Strings [589](#)
 Übersetzen in Python-Werte [589](#)
json (Modul) [587](#)

K

Kalenderarithmetik [datetime](#) [607](#)

Kedelkloppersprook [255](#)

KeyboardInterrupt [148](#), [622](#)

KEYBOARD_KEYS [719](#)

keyDown()/keyUp() [721](#)

KeyError [229](#)

keys() [215](#)

Klammern

 Dictionaries [211](#)

 F-Strings [242](#)

 Funktionen [64](#)

 Funktionsaufruf [129](#)

 Indizes [176](#)

 Rangfolge von Operatoren [52](#)

 Reguläre Ausdrücke gruppieren [271](#)

 Slices [178](#)

 Tupel [197](#)

 Werte aus Dictionaries abrufen [229](#)

Klangdateien [621](#)

Kommandozeile [353](#)

Kommentare

 Auskommentieren [64](#)

 Einführung [64](#)

 Mehrzeilig [239](#)

 TODO-Kommentare [251](#)

Komprimierungstyp [341](#)

Konversationsthreads [628](#)

Koordinatensystem [700](#)

Kreise [667](#)

Kurzschlussauswertung [192](#)

L

Ländercodes [587](#)

Laufwerksangabe [312](#)

launchd [619](#)

Leerraum Entfernen [249](#)

len() , [66](#)

LIMIT (SQLite) [524](#)

line() [666](#)

LineChart() [471](#)
line_num [577](#)
lineterminator [579](#)
Linien [666](#)
list() [198](#)
listdir() [303](#)
Listen
 Bereich für for-Schleifen [182](#)
 Direkte Änderung [175](#)
 Echte Kopien [201](#)
 Einführung [175](#)
 Einrückung [191](#)
 Elemente abrufen [176](#)
 Elemente entfernen [180](#), [189](#)
 Elemente hinzufügen [188](#)
 Länge abrufen [179](#)
 Methoden [188](#)
 Reihenfolge der Einträge umkehren [191](#)
 Sortieren [190](#)
 Veränderbarkeit [195](#), [198](#)
 Verketteten [180](#)
 Verschachtelte Listen [176](#)
 Verwendung [188](#)
 Vorhandensein von Elementen prüfen [183](#)
 Werte ändern [179](#)
listSpreadsheets() [484](#)
ljust() [248](#)
loads() [589](#)
load_workbook() [446](#)
locateOnScreen() [709](#)
Logging
 Datei [164](#)
 deaktivieren [164](#)
 Deaktivieren [166](#)
 Logfunktionen [164](#)
 Loglevel [164](#)
 Zweck [162](#)
Logo [663](#)
Logos [549](#)
LogRecord-Objekte [162](#)
Lokaler Gültigkeitsbereich [139](#), [141](#)

ls [354](#)
lstrip() [249](#)

M

Magisches Orakel [193](#)
makedirs() [310](#)
Maskierung [236](#), [271](#)
 Strings [236](#)
Match-Objekte [268](#)
Maus
 Klicks [703](#)
 Koordinaten von GUI-Elementen ermitteln [706](#)
 Mauszeiger bewegen [701](#)
 Position abrufen [702](#)
max() [418](#)
max_column [447](#)
maximize() [715](#)
maxResult [628](#)
max_row [447](#)
Mehrfachzuweisung [184](#)
merge_cells() [468](#)
mergePage() [549](#)
messages [628](#)
Methoden
 Einführung [187](#)
 Selenium [428](#)
 Tabellenblätter [489](#)
 Verketten [243](#), [655](#)
middleClick() [703](#)
min() [418](#)
minimize() [715](#)
MMS [631](#)
Module
 Ersatz für os.path [305](#)
 Importieren [118](#)
 Namen [118](#)
 Präfix [120](#)
 random [118](#)
 Variablen als Modul speichern [323](#)
ModuleNotFoundError [364](#)

mouseDown() [703](#)
mouseInfo() [706](#)
mouseUp() [703](#)
move() [335](#)
 PyAutoGUI [701](#)
moveTo() [701](#)
Mu
 Debugger [166](#)
 Installieren [42](#)
Multiplikationsoperator [57](#)

N

NameError [180](#)
namelist() [342](#)
NAPS2 [691](#)
new()
 Pillow [649](#)
newline [578](#)
None [134](#)
NoSuchElement [428](#)
not [83](#)
not in [183](#), [194](#)
NOT NULL (SQLite) [515](#)
now() [607](#)
ntfy [634](#)
numpy [747](#)
NVIDIA [747](#)

O

OCR [684](#)
open()
 Dateien [319](#)
 Pillow [647](#)
OpenPyXL
 Diagramme [471](#)
 Font() [463](#)
 Installieren [444](#)
 Zweck [443](#)
OpenWeatherMap
 Registrierung [587](#)

Operatoren

* [57](#)

/ [306](#)

+ , [57](#)

Arithmetisch [52](#)

Auswertungsreihenfolge [85](#)

Binär [83](#)

Boolesche Operatoren [83](#)

Einführung [52](#)

in [240](#)

Listen [183](#)

Listenwiederholung [186](#)

Rangfolge [52](#)

Stringverkettung [186](#)

Unär [83](#)

Vorhandensein von Schlüsseln und Werten in Dictionaries prüfen [217](#)

or [83](#)

ord() [250](#)

ORDER BY (SQLite) [523](#)

OrderedDict-Objekte [580](#)

Ordner

Einführung [303](#)

Größe ermitteln [314](#)

Inhalt auflisten [307](#)

Kopieren [334](#)

Löschen [336](#)

Neue Ordner anlegen [310](#)

Nicht vorhandene Ordner [307](#)

Verzeichnisbaum durchlaufen [338](#)

os (Modul)

basename() [312](#)

Bestandteile eines Pfads zurückgeben [312](#)

Dateien und Ordner löschen [336](#)

dirname() [312](#)

Ersatz für os.path [303](#)

Existenz und Art von Pfaden prüfen [316](#)

getcwd() [307](#)

getsize() [303](#)

isfile [317](#)

listdir() [303](#)

makedirs() [310](#)

Ordnergrößen [303](#)

P

Page-Objekte [542](#)

pages (pypdf) [542](#)

Papierkorb [337](#)

Paragraph-Objekte [557](#), [558](#)

Parameter

 Einführung [135](#)

 Optional [135](#)

parent [312](#)

parents [312](#)

Parser [412](#)

password()

 PyAutoGUI [724](#)

Passwörter

 Eingabe [724](#)

 Hartcodiert [430](#)

 PDFs [552](#)

 Starke Passwörter [301](#)

 Validierung [245](#)

paste() [251](#)

 Pillow [651](#)

 Transparente Pixel [663](#)

Path() [303](#)

PATH [356](#)

pathlib

 Importieren [303](#)

 Inhalt einer Textdatei zurückgeben [317](#)

 Path() [303](#)

 Pfadverkettung [306](#)

 Prüfen, ob Pfad absolut oder relativ ist [309](#)

 String als Textdatei schreiben [317](#)

Path-Objekte

 Anzeige in der Shell [303](#)

 Attribute [312](#)

 Mit Strings verketteten [306](#)

 name [312](#)

 Verzeichnis erstellen [310](#)

PAUSE [699](#)

PDF

Google-Tabellen im PDF-Format herunterladen [485](#)

PdfFileReader-Objekte [542](#)

PdfReader() [542](#)

PDFs

Aus Word-Dateien erstellen [569](#)

Einführung [541](#)

Entschlüsseln [552](#)

Erstellen [547](#)

Kombinieren [547](#)

Probleme bei der Verarbeitung [541](#)

Seiten drehen [549](#)

Text entnehmen [542](#)

Verschlüsseln [552](#)

PDF Texterkennung [691](#)

PEP 8 [61](#)

Pfade Absolut [309](#)

Absoluten Pfad zurückgeben [309](#)

Bestandteile als Strings abrufen [312](#)

Gültigkeit prüfen [303](#)

In String umwandeln [303](#)

Prüfen, ob Pfad absolut ist [309](#)

Relativ [309](#)

Relativen Pfad zurückgeben [309](#)

. und .. [309](#)

Zerlegen [312](#)

PieChart() [471](#)

Pig Latin [255](#)

PIL [647](#), [688](#)

Pillow [688](#)

Bilder einfügen mit transparentem Hintergrund [663](#)

ImageDraw [666](#)

ImageFont [668](#)

Modulname [647](#)

pip [361](#), [744](#)

Pipe [273](#), [289](#)

pixel()

PyAutoGUI [708](#)

pixelMatchesColor() [708](#)

Plausibilitätsprüfung *siehe Assertions* [160](#)

playsound3 [370](#)

- Playwright [746](#)
- point() [666](#)
- poll() [616](#)
- polygon() [667](#)
- Polygone [667](#)
- Popen()
 - Andere Python-Skripte ausführen [617](#)
 - Dateien in der Standardanwendung öffnen [619](#)
 - Dateien unmittelbar in einem Programm öffnen [617](#)
 - shell=True [619](#)
 - Zweck [616](#)
- position()
 - PyAutoGUI [702](#)
- pprint() [230](#)
- press() [721](#)
- print()
 - End- und Trennzeichen [135](#)
 - Rückgabewert [134](#)
 - Stringausgabe [64](#)
 - Testlauf [336](#)
- Programmierspiele [208](#)
- prompt()
 - PyAutoGUI [724](#)
- Prozesse
 - Befehlszeilenargumente übergeben [617](#)
 - Einführung [614](#)
 - Popen() [616](#)
 - Threads [614](#)
- Punkte (Zeichnen) [666](#)
- Punkt (Maßeinheit) [463](#)
- putpixel() [658](#)
- PyAutoGUI
 - Bilderkennung [709](#)
 - Dialogfelder [724](#)
 - Einschränkungen [703](#)
 - Fenster abrufen [714](#)
 - Funktionen [697](#)
 - Installieren [698](#)
 - Mauszeiger steuern [700](#)
 - Notfallsicherung [699](#)
 - Tabulator [718](#)

- Tastatur steuern [718](#)
- PyInstaller [386](#)
- PyMsgBox [373](#)
- PyPDF2
 - Seiten kopieren [547](#)
 - Zweck [541](#)
- pyperclip [251](#), [291](#), [367](#)
- PyPI [744](#)
- pytesseract [746](#)
- PyTesseract [684](#)
- Python
 - Aufrufstack [137](#)
 - Eigene Skripte importieren [323](#)
 - Einführung [34](#)
 - Einrückung [191](#)
 - Externe Programme starten [614](#)
 - Hilfe [45](#)
 - Installieren [42](#)
 - Module importieren [118](#)
 - os.path und pathlib [303](#)
 - Programm abbrechen [110](#)
 - Programme ausführen [62](#)
 - Python-Skripte aus einem Python-Skript heraus starten [617](#)
 - Stilrichtlinie [61](#)
 - Zeitfunktionen [602](#)
- Python-Docx
 - Importieren [557](#)
 - Installieren [557](#)
 - Objekte [557](#)
 - Word-Dateien lesen [558](#)
 - Zweck [557](#)
- Python Image Library [647](#)
- pyttsx [730](#)

Q

- qBittorrent [640](#)

R

- raise [158](#)
- raise_for_status() [398](#)

randint() [118](#)
random
 choice() [185](#)
 randint() [118](#)
 shuffle() [185](#)
range() , [114](#), [117](#)
Rangfolge von Operatoren [52](#)
read() [320](#)
reader() [575](#)
reader-Objekte [575](#)
readlines() [320](#)
recent() [629](#)
Rechner-App [614](#)
Rechtecke [667](#)
Rechtecktuplel [646](#)
Rechtsklick [703](#)
rectangle() [667](#)
Reference() [471](#)
Referenz [198](#)
Referenz (SQLite) [532](#)
refresh()
 EZSheets [484](#)
Reguläre [268](#)
Reguläre Ausdrücke
 ? [278](#)
 Alle Übereinstimmungen finden [292](#)
 Eigene Zeichenklassen [274](#)
 Einführung [265](#)
 E-Mail-Adressen [292](#)
 findall() [292](#)
 Gierige/nicht gierige Suche [282](#)
 Jokerzeichen [274](#)
 Klammern [271](#)
 Nach Übereinstimmung am Ende suchen [284](#)
 Symbolübersicht [270](#)
 Zeichenklassen [299](#)
re (Modul) [268](#)
REPL [44](#), [51](#)
requests [397](#)
resize() [655](#)
Response-Objekte [397](#)

- restore()
 - PyAutoGUI [715](#)
- return [132](#)
- reverse [190](#)
- reverse() [191](#)
- RGBA-Werte [644](#)
- rightClick() [703](#)
- rjust() [248](#)
- rmdir() [336](#)
- rmtree() [336](#)
- Robotic Process Automation [697](#)
- Rohstrings [237](#)
- rollback() (SQLite) [528](#)
- round() [71](#), [606](#)
- rowCount [492](#)
- row_dimensions [467](#)
- RPA [697](#)
- rstrip() [249](#)
- Rückgabewerte [132](#)
- run()
 - cProfile [603](#)
- Runden (Fließkommazahlen) [67](#)
- Run-Objekte [557](#), [558](#)

S

- save()
 - OpenPyXL [457](#)
 - Pillow [647](#)
- ScatterChart() [471](#)
- Schachnotation [219](#)
- Schleifen for [182](#)
- Schlüssel
 - Fehlender Schlüssel [229](#)
 - Schlüssel-Wert-Paare [211](#)
- Schlüsselwortargumente [135](#)
- Schrägstrich
 - Ordnertrennzeichen [305](#)
- Schreibmodus [321](#)
- screenshot() [708](#)
- Screenshots [708](#)

- scroll() [706](#)
- Scrollen [706](#)
- search() [268](#)
 - EZGmail [629](#)
- select() [413](#)
- SELECT (SQLite) [516](#)
- Selenium
 - Importieren [426](#)
 - Klicks simulieren [430](#)
 - Kompatibilität [426](#)
 - Sondertasten simulieren [431](#)
 - Zweck [425](#)
- Semikolon [718](#)
- send()
 - EZGmail [627](#)
- send2trash [337](#)
- send_keys() [430](#)
- Separatorstrings [247](#)
- setdefault() [217](#)
- sheetnames [446](#)
- Sheet-Objekte [487](#)
- sheets [487](#)
- sheetTitles [487](#)
- shelve [323](#)
- shutil
 - Einführung [334](#)
- size()
 - PyAutoGUI [700](#)
- sleep() [602](#), [610](#)
 - PyAutoGUI [722](#)
- Slices
 - Strings [194](#), [239](#)
- SMS [631](#)
- SMTP
 - Zugriffseinschränkungen [628](#)
- smtplib [631](#)
- sort() [190](#)
- Sortieren (SQLite) [523](#)
- Sortierung
 - Alphabetisch [190](#)
 - ASCII [190](#)

- shuffle() [185](#)
- Sound [370](#)
- split() [247](#), [312](#), [575](#)
- Spracherkennung [733](#)
- Spreadsheet-Objekte
 - Attribute [495](#)
 - Erstellen [493](#)
 - ID [493](#)
 - Liste der Sheet-Objekte [495](#)
- SQL-Injection [518](#)
- SQLite [506](#)
- sqlite_schema (SQLite) [515](#)
- SRT-Datei [736](#)
- Stammordner [303](#)
- Standardbibliothek [118](#)
- Standardfarbnamen [644](#)
- startswith() [246](#)
- Statuscodes [397](#)
- Stein,Schere,Papier [123](#)
- stem [312](#)
- Stimme [732](#)
- Stoppuhr [602](#)
- str() [67](#)
- strftime() [611](#)
- Strings
 - Als Listen [239](#)
 - Ändern [175](#), [195](#)
 - Auf Bildschirm ausgeben [64](#)
 - Auf und Groß- oder Kleinschreibung prüfen [243](#)
 - Direktiven für Uhrzeit- und Datumsangaben [611](#)
 - Ersetzen [287](#)
 - Erweiterter Verkettungsoperator [186](#)
 - Ineinander einfügen [251](#)
 - In Listen umwandeln [198](#)
 - Mehrzeilig [238](#)
 - Methoden [243](#)
 - %s [242](#)
 - Stringdarstellung einer Zahl in Integer umwandeln [67](#)
 - Stringinterpolation [242](#)
 - Tabellarische Ausgabe [248](#)

- Vergleiche [243](#)
- Zeichen am Anfang und Ende entfernen [249](#)
- Zeilenumbrüche [238](#)
- strip() [249](#)
- strptime() [612](#)
- subprocess [614](#)
- Subtitle [736](#)
- Suchoperatoren [630](#)
- suffix [312](#)
- summary() [628](#)
- Sushi Go Round [727](#)
- SyntaxError [52](#)
- sys.argv [366](#), [394](#)
- sys.exit() [120](#)
- sys.version [363](#)

T

- Tabelle erstellen (SQLite) [512](#)
- Tabellenblätter
 - Anzahl der Zeilen und Spalten bestimmen [492](#)
 - Daten lesen und schreiben [488](#)
 - Einführung [487](#)
 - Erstellen [495](#)
 - Gleichzeitige Bearbeitung [488](#)
 - Größe ändern [492](#)
 - Kopieren [495](#)
 - Löschen [493](#)
 - Reihenfolge [495](#)
 - Sheet-Objekte [487](#)
 - Umsortieren [495](#)
- Tabellenkalkulationsprogramme [443](#)
- Taskplaner [619](#)
- Tastatur
 - Tastenbetätigung automatisieren [721](#)
 - Tastennamen [719](#)
- Tastenkürzel
 - Automatisierung [719](#), [721](#)
- Teillisten [178](#)
- Teilstrings [239](#)
- Terminal [353](#)

- Tesseract [684](#)
- text()
 - Pillow [668](#)
- Text
 - Ausrichtung [248](#)
 - Eingabe durch virtuelle Tastendrücke [718](#)
 - In Bildern zeichnen [668](#)
 - Textbearbeitung [251](#)
- Texterkennung [684](#)
- Textnachrichten
 - Benachrichtigungen von Programmen [628](#)
 - Empfangen [634](#)
 - Senden [634](#)
 - SMS-E-Mail-Gateways [631](#)
- TEXT (SQLite) [513](#)
- Text-zu-Sprache [730](#)
- Threads
 - Argumente übergeben [617](#)
- Tic Tac Toe [223](#)
- time() [602](#)
- time (Modul)
 - ctime() [602](#)
 - time() [602](#)
- toggeln [98](#)
- Transaktion [517](#)
- transcribe() (whisper) [733](#)
- Transparenz [644](#)
- transpose() [657](#)
- Trennzeichen
 - CSV-Dateien [579](#)
 - Pfade [305](#)
- True [80](#), [110](#)
- TrueType [668](#)
- Truthy [113](#)
- try [146](#), [158](#)
- TTS [730](#)
- Tupel
 - Benannte Tupel [700](#)
 - Einzelner Wert [197](#)
 - In Listen umwandeln [198](#)
 - Schreibweise [197](#)

Tupel Unpacking [184](#)
tuple() [198](#)
type() , [70](#)
TypeError [190](#), [197](#), [306](#)

U

Überschriften [566](#)
Unicode [250](#), [399](#)
Unix-Epoche [602](#)
unlink() [336](#)
unread() [628](#)
Untertiteldatei [736](#)
updateRow() [492](#)
updateRows() [492](#)
UPDATE (SQLite) [516](#)
upload()
 EZSheets [483](#), [504](#)
urllib2 [397](#)
User-Agent-String [425](#)

V

value
 Cell-Objekte [446](#)
ValueError [184](#), [188](#), [189](#)
values() [215](#)
Variablen
 Als Modul speichern [323](#)
 Einführung [58](#)
 Erweiterte Zuweisungsoperatoren [186](#)
 Global [139](#), [144](#)
 Initialisieren [59](#)
 Lokal [139](#), [141](#)
 Löschen [180](#)
 Namen [61](#), [142](#)
 Parameter [135](#)
 Überschreiben [59](#)
 Verweise [198](#)
venv [358](#), [745](#)
VERBOSE [288](#)
Vergleichsoperatoren [81](#), [85](#)

- Kalenderdaten [608](#)
- Verkettungsoperator
 - Strings , [57](#)
- version [363](#)
- Verweise
 - Echte Kopien statt Verweisen [201](#)
 - Einführung [198](#)
 - Übergeben [175](#)
- Verzeichnisse [303](#)
- Videos herunterladen [737](#)
- Virtuelle Umgebung [358](#)
- Voice [732](#)
- Volumes [303](#)
- VTT-Datei [736](#)

W

- Wahrheitswertetafel [83](#)
- wait() [616](#)
- walk() [338](#)
- Wartezeit [602](#)
- Wasserzeichen [551](#), [663](#)
- WD_BREAK.PAGE [567](#)
- webbrowser (Modul) [418](#)
- Websites
 - Dateien herunterladen [397](#)
 - JSON-Inhalte [587](#)
- Webtreiber [426](#)
- Weißraum
 - Operatoren [52](#)
- Werte [52](#)
- WHERE (SQLite) [520](#)
- while-Schleifen
 - Einführung [103](#)
- whisper [733](#)
- Wiederholungsoperator Strings [57](#)
- Window-Objekte [713](#)
- Word-Dateien
 - Kompletten Text abrufen [559](#)
 - Schreiben [564](#)
 - Struktur [557](#)

- Zeilenwechsel [567](#)
- Workbook() [457](#)
- Workbook-Objekte [446](#)
- write() [321](#)
 - PyAutoGUI [718](#)
 - pypdf [547](#)
- writeheader() [580](#)
- writer() [578](#)
- writer-Objekte [578](#)
- writerow() [578](#)
- Wurzelordner [303](#)

X

- XKCD [397](#)

Y

- Youtube [737](#)
- yt_dlp [737](#)

Z

- Zahlenratespiel [121](#)
- Zauberlehrling [699](#)
- Zeichenklassen [299](#)
- Zeilenumbruch
 - Reguläre Ausdrücke [284](#)
 - Strings an Zeilenumbrüchen trennen [247](#)
- Zeilenumbrüche
 - Zeilenendezeichen [579](#)
- Zeit
 - Aktuelle Uhrzeit [607](#)
 - Ausführung von Programmen planen [619](#)
 - Dauer der Ausführung eines Codeabschnitts [602](#)
 - Kalenderarithmetik [607](#), [608](#)
 - Zeitplanungswerkzeuge des Betriebssystem [619](#)
- Zeit (SQLite) [514](#)
- ZeroDivisionError [146](#)
- Zickzackmuster [148](#)
- ZIP-Dateien
 - Datensicherung [344](#)

- Entpacken [342](#)
- Erstellen [341](#)
- Google-Tabellen als ZIP-Archive herunterladen [485](#)
- Lesen [342](#)
- ZIP_DEFLATED [341](#)
- zipfile [341](#)
- ZipFile() [342](#)
- ZipFile-Objekte [342](#)
- Zirkumflex
 - Nach Übereinstimmung am Anfang suchen [284](#)
 - Negative Zeichenklassen [274](#)
- Zufallszahlen [118](#)
- Zuweisung
 - Listenelemente [179](#)
 - Variablen [59](#)
 - Verweise kopieren [198](#)
- Zuweisungsoperator [59](#), [81](#)
- Zwischenablage
 - Aktualisierbare Mehrfach-Zwischenablage [332](#)
 - Durchsuchen [292](#)
 - Selektorstrings kopieren [408](#)
 - Text abrufen [291](#)