A byte of Python deutsch

Release 2026

Horst JENS (Übersetzung), Swaroop CH (Original)

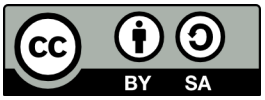
01.06.2026

Contents

“A Byte of Python deutsch” ist eine deutsche Übersetzung des englischsprachigen Buchs “A byte of Python” von Swaroop Chitlur. “A byte of Python” ist eine Einführung in die Programmiersprache Python.

Bemerkung

Dieses Übersetzungsprojekt wurde zusammen mit der ukrainischen Übersetzung 2025 von Netidee.at (https://www.netidee.at) gefördert.



Über diese Übersetzung

Diese Übersetzung beruht auf der englischen Originalausgabe von “A byte of Python” (https://python.swaroopch.com/, Stand Mai 2026), welche sich auf Python Version 3 bezieht.

Es wird versucht diese Übersetzung laufend aktuell zu halten, da sich auch das Originalprojekt (Github Link https://github.com/swaroopch/byte-of-python) gepflegt wird und sich ständig ändert.

Ich habe mir bei der Übersetzung Freiheiten erlaubt und nicht immer alles wortwörtlich übersetzt, z.B. wurden Textstellen die sich nur auf die veraltete Version 2 von Python beziehen gekürzt und an manchen Stellen Ergänzungen eingefüft.

Diese Übersetzung ist optimiert auf die html-Version, welche Sie direkt auf dieser Homepage lesen können:

https://spielend-programmieren.at/byte_of_python_deutsch

Darüberhinaus stehen folgende Formate zur Verfügung:

- Epub-Format für Ebook-Reader:

  https://spielend-programmieren.at/byte_of_python_deutsch/AbyteofPythondeutsch.epub

- Pdf-Format zum Ausdrucken:

  https://spielend-programmieren.at/byte_of_python_deutsch/AbyteofPythondeutsch.pdf

CHAPTER 1

Widmung

An Kalyan Varma und viele andere ältere Studierende am PESIT, die uns GNU/Linux und die Welt der Open-Source-Software nähergebracht haben.

Zum Gedenken an Atul Chitnis, einen Freund und Mentor, der sehr vermisst wird.

An die Pioniere, die das Internet möglich gemacht haben. Dieses Buch wurde erstmals im Jahr 2003 verfasst. Es ist nach wie vor beliebt, dank der Vision der Pioniere, Wissen im Internet zu teilen.

CHAPTER 2

Vorwort

Python ist wahrscheinlich eine der wenigen Programmiersprachen, die sowohl einfach als auch leistungsfähig ist. Dies ist gut für Anfänger ebenso wie für Experten und, noch wichtiger, sie macht Spaß beim Programmieren. Dieses Buch soll Ihnen helfen, diese wunderbare Sprache zu erlernen und zeigen, wie Sie Dinge schnell und mühelos erledigen können – im Grunde „das Gegengift zu Ihren Programmierproblemen".

2.1 Für wen dieses Buch ist

Dieses Buch dient als Leitfaden oder Tutorial für die Programmiersprache Python. Es richtet sich hauptsächlich an Neulinge. Es ist auch für erfahrene Programmierer nützlich.

Das Ziel ist, dass Sie Python aus diesem Buch lernen können, selbst wenn das Einzige, was Sie über Computer wissen, ist, wie man Textdateien speichert. Wenn Sie bereits Programmiererfahrung haben, können Sie Python ebenfalls aus diesem Buch lernen.

Wenn Sie bereits Programmiererfahrung haben, werden Sie sich für die Unterschiede zwischen Python und Ihrer Lieblingsprogrammiersprache interessieren – ich habe viele solcher Unterschiede hervorgehoben. Eine kleine Warnung jedoch: Python wird schon bald Ihre Lieblingsprogrammiersprache werden!

2.2 Offizielle Website

Die offizielle Website des Buches ist:

- Originalversion (englisch): https://python.swaroopch.com/
- deutsche Übersetzung: https://spielend-programmieren.at/byte_of_python_deutsch/

Dort gibt es auch Link zu Feedback und zu den Github-Repositories, um Korrekturen oder Verbesserungsvorschläge zu senden.

2.3 Ein Denkanstoß

> Es gibt zwei Arten, ein Software-Design zu konstruieren: Die eine besteht darin, es so einfach zu machen, dass offensichtlich keine Mängel vorhanden sind; die andere besteht darin, es so kompliziert zu machen, dass keine offensichtlichen Mängel vorhanden sind.
>
> – *C. A. R. Horare*

> Erfolg im Leben ist weniger eine Frage von Talent und Gelegenheit, sondern von Konzentration und Ausdauer.
>
> – *C. W. Wendte*

CHAPTER 3

Über Python

Python[1] gehört zu den wenigen Programmiersprachen, die sowohl einfach zu erlernen als auch leistungsstark sind. Python erlaubt es, sich auf die das Programmieren (die Problemlösung) zu konzentrieren, anstatt auf die Programmiersprache und deren Syntax und Struktur.

Die offizielle Einführung zu Python lautet:

> Python ist eine leicht zu erlernende, leistungsstarke Programmiersprache. Sie verfügt über effiziente, hochrangige Datenstrukturen und einen einfachen, aber effektiven Ansatz für die objektorientierte Programmierung. Pythons elegante Syntax und dynamische Typisierung machen es zusammen mit seiner interpretierten Natur zu einer idealen Sprache für Skripte und die schnelle Anwendungsentwicklung in vielen Bereichen auf den meisten Plattformen.

3.1 Die Geschichte hinter dem Namen

Guido van Rossum[2], der Schöpfer der Programmiersprache Python, benannte sie nach der BBC-Sendung „Monty Python's Flying Circus". Guido ist *kein* Freund von Würgeschlangen.

3.2 Eigenschaften von Python

3.2.1 Einfachheit

Python ist eine einfache und minimalistische Sprache. Ein gut geschriebenes Python-Programm liest sich fast wie (sehr vereinfachtes) Englisch. Diese Pseudocode[Pseudocode]-Natur von Python ist eine seiner größten Stärken. Sie ermöglicht es, sich auf die Problemlösung zu konzentrieren, anstatt auf die Sprache selbst.

[1] https://python.org
[2] https://gvanrossum.github.io/

3.2.2 Leicht zu erlernen

Der Einstieg in Python extrem leicht, besonders für Programmieranfänger. Python hat, wie bereits erwähnt, eine außerordentlich einfache Syntax.

3.2.3 Frei und Open Source

Python ist ein Beispiel für *FLOSS* (*F*ree, *L*ibre, *O*pen-*S*ource *S*oftware). Vereinfacht erklärt: Jeder darf diese Software frei verbreiten, jeder darf ihren Quellcode studieren, die Software verändern und Teile davon oder alles in neuen, freien Programmen verwenden. FLOSS basiert auf dem Konzept einer Gemeinschaft, die ihr Wissen teilt. Das ist einer der Gründe, warum Python so gut ist – es wurde von einer Gemeinschaft entwickelt, es "lebt" und wird ständig weiter verbessert.

3.2.4 Hochsprache (high-level)

Wenn man in Python programmiert, muss man sich nie um Hardware-nahe Details der Systemarchitektur wie die Speicherverwaltung kümmern.

3.2.5 Portabilität

Dank seiner Open-Source-Natur wurde Python auf viele Plattformen (und Betriebssysteme) portiert (d. h. angepasst, um darauf zu laufen). Alle Python-Programme laufen auf jeder dieser Plattformen ohne Änderungen, (außer Sie verwenden Betriebssystem-abhängige Funktionen die es auf anderen Plattformen nicht gibt).

Man kann Python unter GNU/Linux, Windows, FreeBSD, Macintosh, Solaris, OS/2, Amiga, AROS, AS/400, BeOS, OS/390, z/OS, Palm OS, QNX, VMS, Psion, Acorn RISC OS, VxWorks, PlayStation, Sharp Zaurus, Windows CE und PocketPC verwenden!

Man kann mittels Bibliotheken wie Kivy Apps in Python programmieren die sowohl auf dem Computer als auch auf dem Smartphone / Tablet unter iOS und Android funktionieren.

3.2.6 Interpretation

Dies bedarf einer kurzen Erklärung. Es gibt kompilierte und interpretierte Programmiersprachen. Python ist eine interpretierte Programmiersprache.

Ein in einer kompilierten Sprache wie C oder C++ geschriebenes Programm wird mithilfe eines Compilers mit verschiedenen Optionen und Flags aus der Quellsprache (C oder C++) in die Sprache Ihres Computers (Binärcode, d. h. Nullen und Einsen) übersetzt. Beim Ausführen des Programms kopiert der Linker/Loader das Programm von der Festplatte in den Arbeitsspeicher und startet die Ausführung.

Bei einer interpretierten Sprache (wieh Python oder Javascript) wird jede Programmzeile vom Quellcode direkt in Binärsprache übersetzt, man muß nicht auf den compiler warten sondern das Programm startet "sofort".

Python (eine interpretierte Sprache) benötigt keine Kompilierung in Binärcode. Jedes Python-Programm wird direkt aus dem Quellcode ausgeführt. Intern wandelt Python den Quellcode in eine Zwischenform, den sogenannten Bytecode, um, übersetzt diesen dann in die Sprache Ihres Computers und führt ihn anschließend aus. Dadurch wird die Verwendung von Python deutlich einfacher, da Sie sich nicht um das Kompilieren des Programms, das Verlinken und Laden der benötigten Bibliotheken usw. kümmern müssen. Python-Programme sind dadurch auch viel portabler, da man sie einfach auf einen anderen Computer kopieren kann und sie dort sofort funktionieren!

3.2.7 Objektorientiert

Python unterstützt sowohl prozedurale als auch objektorientierte Programmierung (OOP). In prozeduralen Sprachen basiert das Programm auf Prozeduren oder Funktionen, die wiederverwendbare Programmbausteine darstellen. In objektorientierten Sprachen hingegen basiert das Programm auf Objekten, die Daten und Funktionalität vereinen. Python bietet eine sehr leistungsstarke und gleichzeitig einfache Möglichkeit zur OOP, insbesondere im Vergleich zu großen Sprachen wie C++ oder Java.

3.2.8 Erweiterbar

Wenn man einen kritischen Codeabschnitt besonders schnell ausführen muss oder einen Algorithmus nicht offenlegen möchten, kann man diesen Teil des Programms in C oder C++ schreiben und ihn dann in im Python-Programm verwenden.

3.2.9 Einbettbar

Man kann Python in C/C+±Programme einbetten (embedding) , um den Benutzern Skripting-Funktionen in Pyhton zu bieten.

3.2.10 Umfangreiche Bibliotheken

Die Python-Standardbibliothek ist wirklich riesig. Sie unterstützt bei einer Vielzahl von Aufgaben, darunter reguläre Ausdrücke, Dokumentationsgenerierung, Unit-Tests, Threading, Datenbanken, Webbrowser, CGI, FTP, E-Mail, XML, XML-RPC, HTML, WAV-Dateien, Kryptografie, GUIs (grafische Benutzeroberflächen) und andere systemabhängige Funktionen. All dies ist immer verfügbar, wo auch immer Python installiert ist. Dies wird als „Batteries Included"-Philosophie ("Batterien mitgeliefert") von Python bezeichnet.

Neben der Standardbibliothek gibt es zahlreiche weitere hochwertige Bibliotheken, die man im Python Package Index finden kann.

3.2.11 Zusammenfassung

Python ist eine faszinierende und leistungsstarke Programmiersprache. Sie bietet die ideale Kombination aus Performance und Funktionen, die das Programmieren in Python sowohl erfreulich als auch einfach machen.

3.3 Python 3 versus 2

Dieses Buch basiert auf Python3. Ältere Versionen des Buchs (und z.B. die darauf basierende ekzellente deutsche Übersetzung auf https://cito.github.io/byte_of_python/) basieren auf Python2, welches seit Jänner 2020 nicht mehr offiziell unterstützt wird. Siehe auch https://peps.python.org/pep-0373/.

3.4 Was Programmierer sagen

Zitate bekannter Programmierer wie Eric S. Raymond (ESR) über Python:

- Eric S. Raymond ist der Autor von „The Cathedral and the Bazaar" und prägte den Begriff „Open Source". Er sagt, dass Python seine Lieblingsprogrammiersprache geworden ist (http://www.python.org/about/success/esr/). Dieser Artikel war die Inspiration für meine ersten Erfahrungen mit Python.
- Bruce Eckel ist der Autor der bekannten Bücher „Thinking in Java" und „Thinking in C++". Er sagt, dass ihn keine andere Sprache so produktiv gemacht hat wie Python. Python sei wohl die einzige Sprache, die sich darauf konzentriere, Programmierern die Arbeit zu erleichtern. Lesen Sie das vollständige Interview (http://www.artima.com/intv/aboutme.html), um mehr zu erfahren.

Peter Norvig ist ein bekannter Lisp-Autor und Leiter der Suchqualität bei Google (vielen Dank an Guido van Rossum für diesen Hinweis). Er sagt, dass das Schreiben von Python dem Schreiben von Pseudocode ähnelt. Er betont, dass Python schon immer ein integraler Bestandteil von Google war. Dies lässt sich auch auf der Google-Jobseite (http://www.google.com/jobs/index.html) überprüfen, wo Python-Kenntnisse als Voraussetzung für Softwareentwickler aufgeführt sind.

CHAPTER 4

Installation

Wenn wir in diesem Buch von **“Python 3”** sprechen, beziehen wir uns auf eine (aktuelle) Version von Python, die gleich oder höher als die Version Python 3.12 welche in diesem Buch verwendet wird.

4.1 Installation unter Windows

Besuchen Sie https://www.python.org/downloads/ und laden Sie die neueste Version herunter. Zum Zeitpunkt der Erstellung dieses Textes war es Python 3.12. Die Installation erfolgt wie bei jeder anderen Windows-Software.

Bemerkung

Falls Ihre Windows-Version älter als Vista ist, sollten Sie nur Python 3.4 herunterladen, da spätere Versionen neuere Windows-Versionen erfordern.

Achtung

Stellen Sie sicher, dass Sie die Option **`Add Python to PATH`** (Python zum Windows PATH hinzufügen) aktivieren.

Um den Installationsort zu ändern, klicken Sie auf `**Customize installation**`, dann auf `**Next**` und geben Sie `C:\python3` (oder einen anderen geeigneten Ort) als Installationsort ein.

Falls Sie die Option `**Add Python to PATH**` nicht aktiviert haben, aktivieren Sie `**Add Python to environment variables**`. Dies bewirkt dasselbe wie die Option auf dem ersten Installationsbildschirm.

Sie können wählen, ob Sie den **Launcher für alle Benutzer** installieren möchten oder nicht – das spielt keine große Rolle. Der Launcher wird verwendet, um zwischen verschiedenen installierten Python-Versionen zu wechseln.

Falls der **PATH** nicht korrekt gesetzt wurde (durch Aktivieren der Optionen `Add Python to PATH` oder `Add Python to environment variables`), folgen Sie den Schritten im nächsten Abschnitt (**Eingabeaufforderung (DOS)**), um dies zu korrigieren. Andernfalls gehen Sie direkt zum Abschnitt **Ausführen der Python-Eingabeaufforderung unter Windows** in diesem Dokument.

Hinweis für Personen mit Programmierkenntnissen

Falls Sie mit Docker vertraut sind, sehen Sie sich Python in Docker und Docker auf Windows an.

4.1.1 Eingabeaufforderung (DOS-Prompt)

Wenn Sie Python von der Windows-Eingabeaufforderung (DOS-Eingabeaufforderung, manchmal auch `cmd` genannt) aus verwenden möchten, müssen Sie die **PATH-Umgebungsvariable** entsprechend setzen.

Für Windows 2000, XP, 2003:

1. Klicken Sie auf **Systemsteuerung** → **System** → **Erweitert** → **Umgebungsvariablen**.
2. Klicken Sie auf die Variable namens `**PATH**` unter **Systemvariablen**.
3. Fügen Sie den Pfad zu Ihrem Python-Installationsverzeichnis hinzu, z. B. `C:\Python3\`.

Für Windows 7, 8, 10, 11:

1. Öffnen Sie die **Systemsteuerung** → **System und Sicherheit** → **System** → **Erweiterte Systemeinstellungen** (ganz rechts) → **Umgebungsvariablen** (unten) → (wählen Sie die Variable `**Path**` aus und klicken Sie auf **Bearbeiten**) → **Neu** → (geben Sie den Pfad zu Ihrem Python-Installationsverzeichnis ein, z. B. `C:\Python3\`).

4.1.2 Ausführen der Python-Eingabeaufforderung unter Windows

Für Windows-Benutzer können Sie den Interpreter in der Eingabeaufforderung ausführen, wenn Sie die PATH-Variable entsprechend gesetzt haben.

Um das Terminal unter Windows zu öffnen:

1. Klicken Sie auf die **Start-Schaltfläche** und dann auf **Ausführen**.
2. Geben Sie im Dialogfeld `cmd` ein und drücken Sie die **Eingabetaste**.

Geben Sie dann `python` ein und stellen Sie sicher, dass keine Fehler auftreten.

4.2 Installation unter Mac OS X

Für Mac OS X-Benutzer: Verwenden Sie Homebrew: `brew install python3`.

Um die Installation zu überprüfen:

1. Öffnen Sie das Terminal, indem Sie die Tastenkombination **[Befehl + Leertaste]** drücken (um die Spotlight-Suche zu öffnen).
2. Geben Sie `Terminal` ein und drücken Sie die **Eingabetaste**.
3. Führen Sie `python3` aus und stellen Sie sicher, dass keine Fehler auftreten.

4.3 Installation unter GNU/Linux

Für GNU/Linux-Benutzer: Verwenden Sie den Paketmanager Ihrer Distribution, um Python 3 zu installieren. Beispiel für Debian und Ubuntu:

```
sudo apt-get update && sudo apt-get install python3
```

Um die Installation zu überprüfen:

1. Öffnen Sie das Terminal, indem Sie die Anwendung **Terminal** starten oder durch Drücken von **[Alt + F2]** und Eingabe von `gnome-terminal`. Falls das nicht funktioniert, konsultieren Sie bitte die Dokumentation Ihrer spezifischen GNU/Linux-Distribution.
2. Führen Sie `python3` aus und stellen Sie sicher, dass keine Fehler auftreten.

Sie können die Python-Version auf dem Bildschirm anzeigen lassen, indem Sie Folgendes ausführen:

```
$ python3 -V
Python 3.12
```

Hinweis: `$` ist die Eingabeaufforderung der Shell. Diese kann je nach Einstellungen Ihres Betriebssystems auf Ihrem Computer anders aussehen. Daher werde ich die Eingabeaufforderung einfach mit dem Symbol `$` kennzeichnen.

Achtung: Die Ausgabe kann auf Ihrem Computer abweichen, abhängig von der auf Ihrem Computer installierten Python-Version.

4.4 Zusammenfassung

Von nun an gehen wir davon aus, dass Python auf Ihrem System installiert ist.

Als Nächstes schreiben wir unser erstes Python-Programm.

CHAPTER 5

Erste Schritte

Wir werden nun sehen, wie man ein traditionelles „Hello World“-Programm in Python ausführt. Dies wird Ihnen beibringen, wie Sie Python-Programme schreiben, speichern und ausführen.

Es gibt zwei Möglichkeiten, Python zur Ausführung eines Programms zu benutzen – über die interaktive Interpreter-Eingabeaufforderung oder über eine Quelldatei. Wir werden nun beide Methoden kennenlernen.

5.1 Verwendung der Interpreter-Eingabeaufforderung

Öffnen Sie das Terminal Ihres Betriebssystems (wie zuvor im Kapitel Installation besprochen) und öffnen Sie anschließend die Python-Eingabeaufforderung, indem Sie `python3` eingeben und die [Enter]-Taste drücken.

Sobald Sie Python gestartet haben, sollten Sie die Eingabeaufforderung (den “Prompt”) sehen:

>>>

Sie können hier direkt Dinge eintippen. Dies wird die Python-Interpreter-Eingabeaufforderung genannt.

Geben Sie an der Python-Eingabeaufforderung Folgendes ein:

```
print("Hallo Welt")
```

gefolgt von der [Enter]-Taste. Sie sollten die Wörter `Hallo Welt` auf dem Bildschirm erscheinen sehen.

Hier ist ein Beispiel dessen, was Sie sehen sollten, wenn Sie einen Mac-OS-X-Computer verwenden. Die Details über die Python-Software unterscheiden sich je nach Computer, aber der Teil ab der Eingabeaufforderung (d. h. ab >>>) sollte unabhängig vom verwendeten Betriebssystem gleich sein.

Anmerkung: Ihr Python Interpreter hat wahrscheinlich eine höhere Versionsnummer, z.B.

```
$ python3
Python 3.12.3 (main, Mar 23 2026, 19:04:32) [GCC 13.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hallo Welt")
Hallo Welt
>>>
```

Beachten Sie, dass Python Ihnen die Ausgabe der Zeile sofort präsentiert, sobald Sie die Eingabetaste (Enter-Taste) drücken!

Was Sie soeben eingegeben haben, ist eine einzelne Python-Anweisung. Wir verwenden `print`, um (wenig überraschend) jeden Wert auszugeben, den Sie ihm übergeben. Hier übergeben wir den Text `Hallo Welt`, und dieser wird unverzüglich auf dem Bildschirm ausgegeben.

5.1.1 Wie man die Interpreter-Eingabeaufforderung beendet

Wenn Sie eine GNU/Linux- oder OS-X-Shell verwenden, können Sie die Interpreter-Eingabeaufforderung verlassen, indem Sie `[ctrl + d]` drücken oder `exit()` eingeben gefolgt von der [Enter]-Taste.

Wenn Sie die Windows-Eingabeaufforderung verwenden, drücken Sie `[ctrl + z]` gefolgt von der [Enter]-Taste.

5.2 Auswahl eines Editors

Wir können unser Programm nicht jedes Mal an der Interpreter-Eingabeaufforderung eingeben, wenn wir etwas ausführen wollen, daher müssen wir es in Dateien speichern, sodass wir unsere Programme beliebig oft ausführen können.

Um unsere Python-Quelldateien zu erstellen, benötigen wir ein Editor-Programm (*code_editor*), in dem wir schreiben und speichern können. Ein guter Programmiereditor erleichtert Ihnen das Schreiben der Quelldateien erheblich. Daher ist die Wahl eines Editors tatsächlich entscheidend. Sie sollten einen Editor so auswählen, wie Sie ein Auto auswählen würden, das Sie kaufen möchten. Ein guter Editor hilft Ihnen dabei, Python-Programme einfach zu schreiben, macht Ihre Reise angenehmer und hilft Ihnen, Ihr Ziel (Ihr Lernziel) schneller und sicherer zu erreichen.

Eine der grundlegenden Anforderungen ist die Syntaxhervorhebung (*syntax heiglighting*), bei der die verschiedenen Teile Ihres Python-Programms farblich hervorgehoben werden, sodass Sie Ihr Programm sehen und seine Ausführung besser nachvollziehen können.

Wenn Sie keine Vorstellung davon haben, womit Sie beginnen sollen, empfehle ich Ihnen die Verwendung der Software PyCharm, die für Windows, Mac OS X und GNU/Linux verfügbar ist. Einzelheiten folgen im nächsten Abschnitt.

Wenn Sie Windows verwenden, benutzen Sie nicht Notepad – es ist zwar möglich damit Python-Programme zu schreiben, aber etwas anstrengend: Notepad bietet keine Syntaxhervorhebung und, was noch wichtiger ist, es unterstüzt keine automatischen Einrückungen. Diese sind für Python sehr wichtig.

Wenn Sie ein erfahrener Programmierer sind, verwenden Sie vermutlich bereits Vim oder Emacs. Diese beiden Editoren gehören zweifellos zu den mächtigsten, und Sie werden davon profitieren, sie für Ihre Python-Programme zu verwenden. Ich selbst verwende beide für die meisten meiner Programme und habe sogar ein komplettes Buch über Vim geschrieben.

Falls Sie bereit sind, sich die Zeit zu nehmen, Vim oder Emacs zu erlernen, empfehle ich Ihnen dringend, eines der beiden zu erlernen, da es sich langfristig für Sie auszahlen wird. Für Anfänger jedoch, wie bereits erwähnt, ist PyCharm ein guter Startpunkt, sodass Sie sich zunächst auf das Erlernen von Python konzentrieren können und nicht auf den Editor.

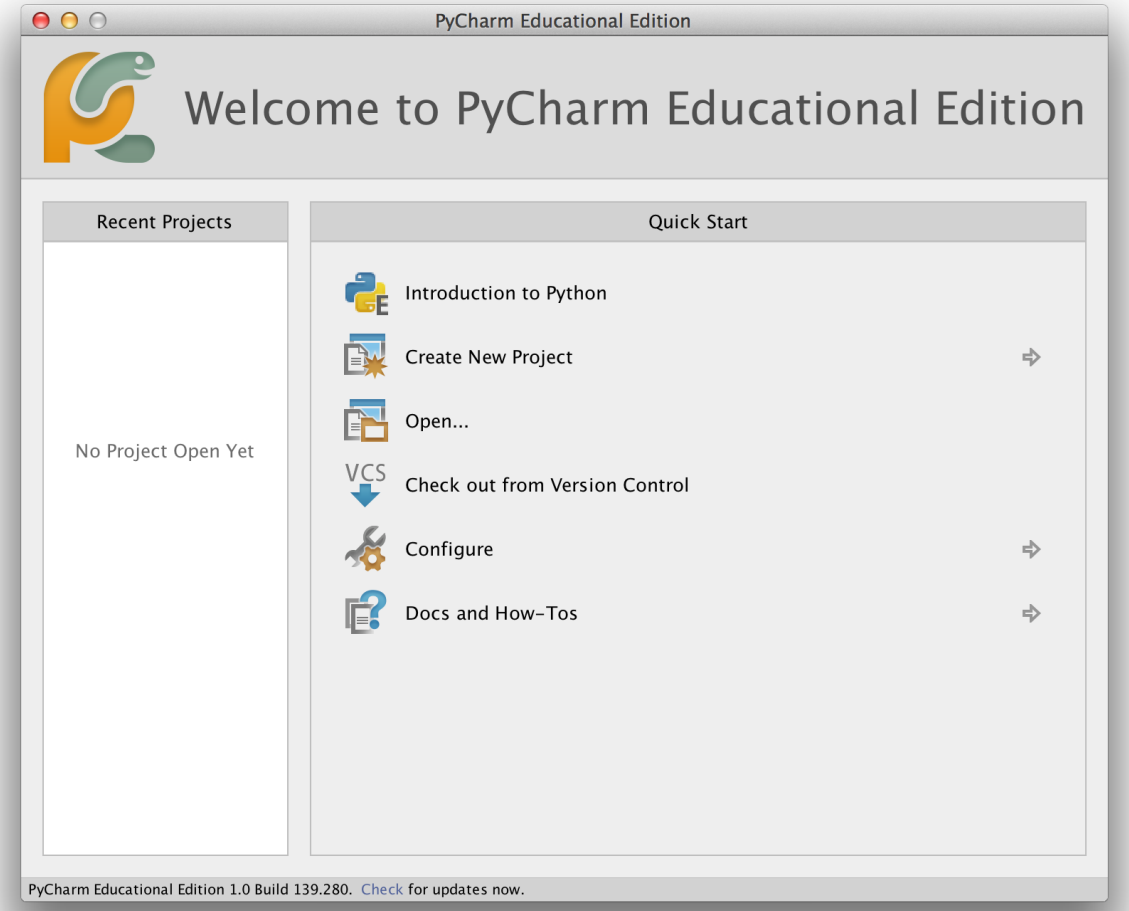
Um es nochmals zu betonen: Bitte wählen Sie einen geeigneten Editor – er kann das Schreiben von Python-Programmen angenehmer und einfacher machen.

Falls Sie an einer ausführlichen Diskussion zu diesem Thema interessiert sind, lesen Sie Finding the Perfect Python Code Editor.

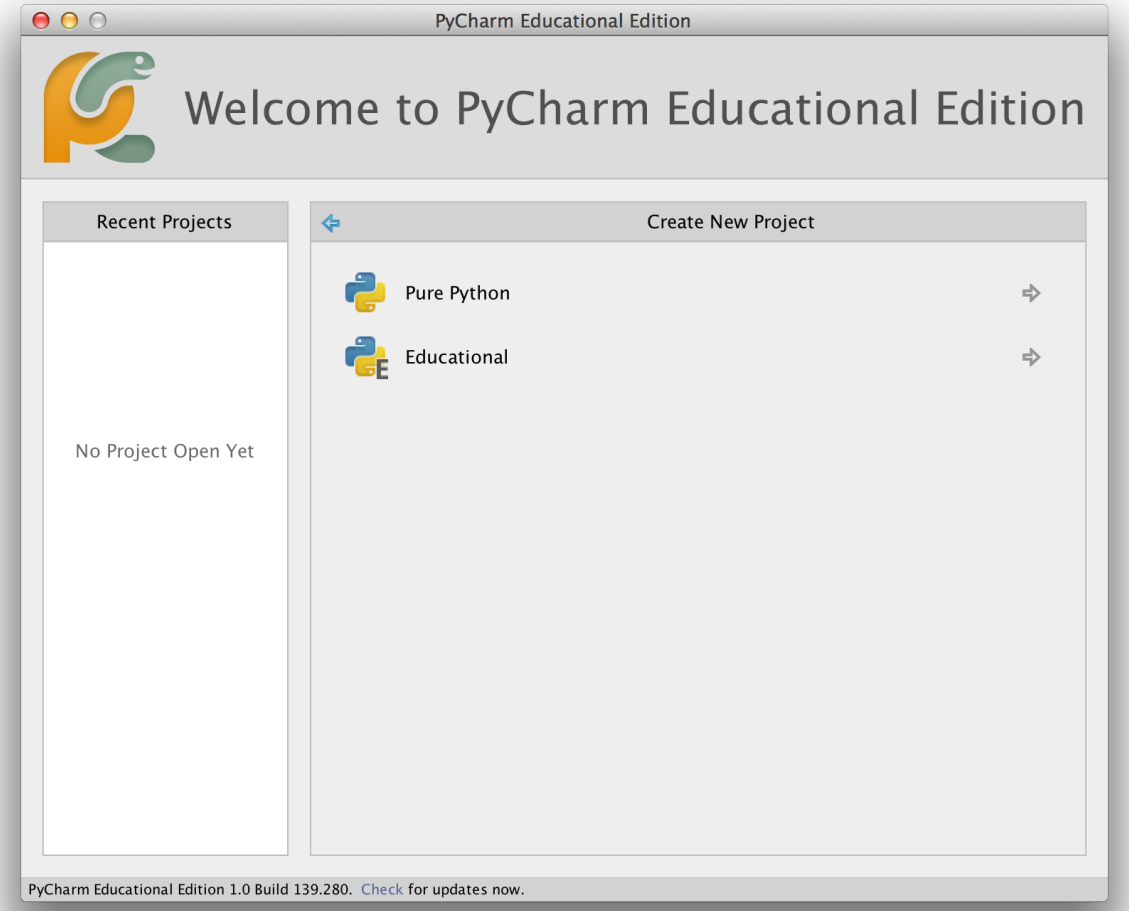
5.3 PyCharm

Die PyCharm ist ein kostenloser Editor, den Sie zum Schreiben von Python-Programmen verwenden können.

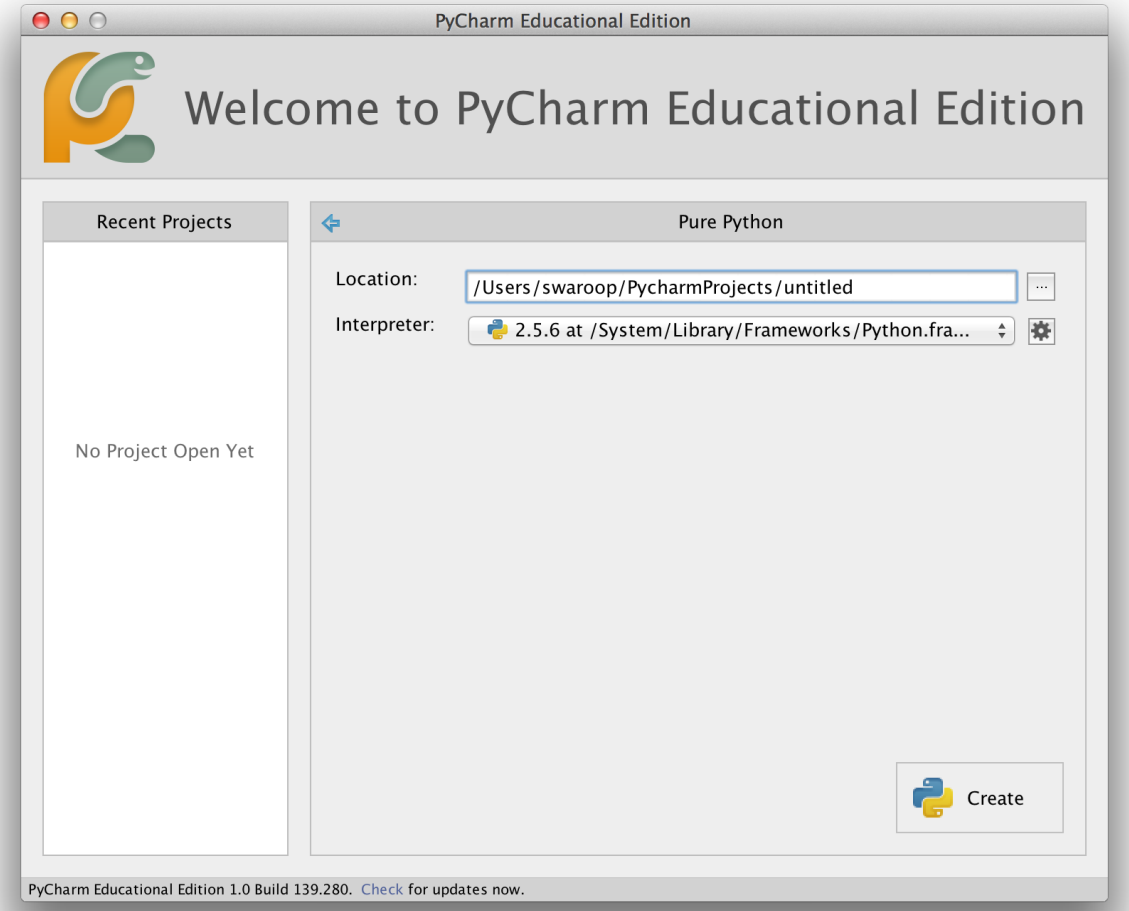
Wenn Sie PyCharm öffnen, sehen Sie dies. Klicken Sie auf `Create New Project`:



Wählen Sie `Pure Python`:

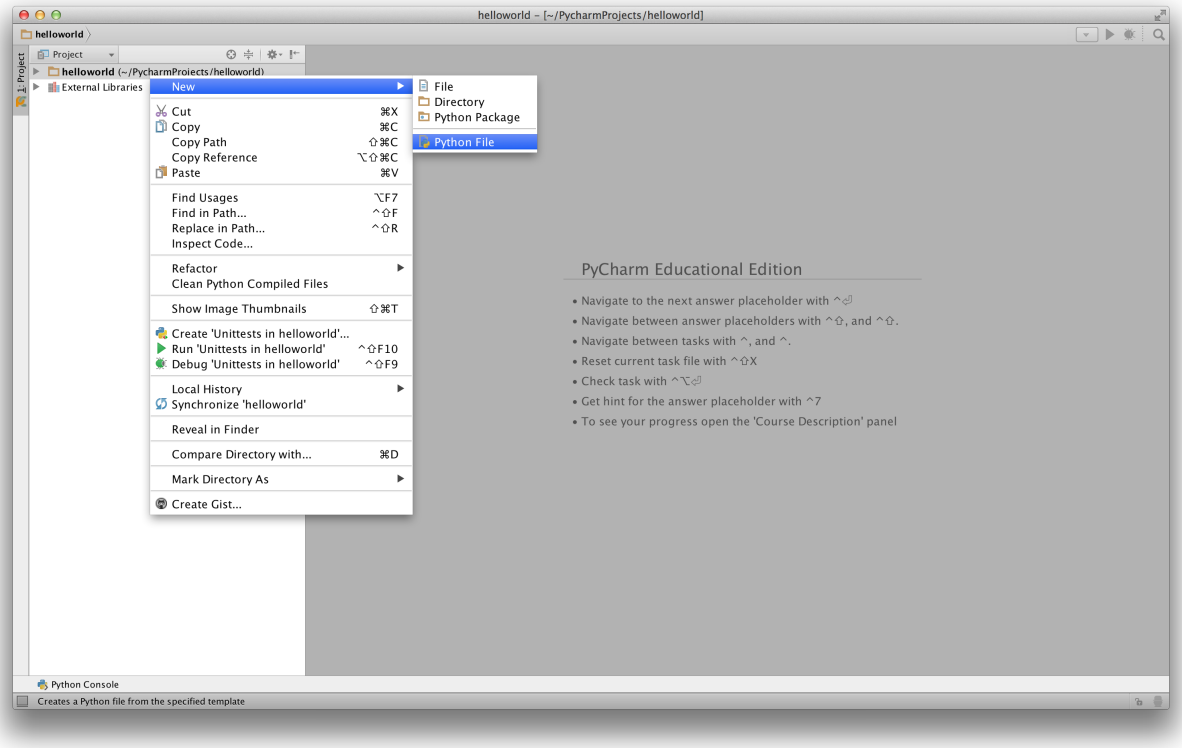


Ändern Sie `untitled` in `helloworld` als Speicherort des Projekts; Sie sollten ähnliche Details wie hier sehen:

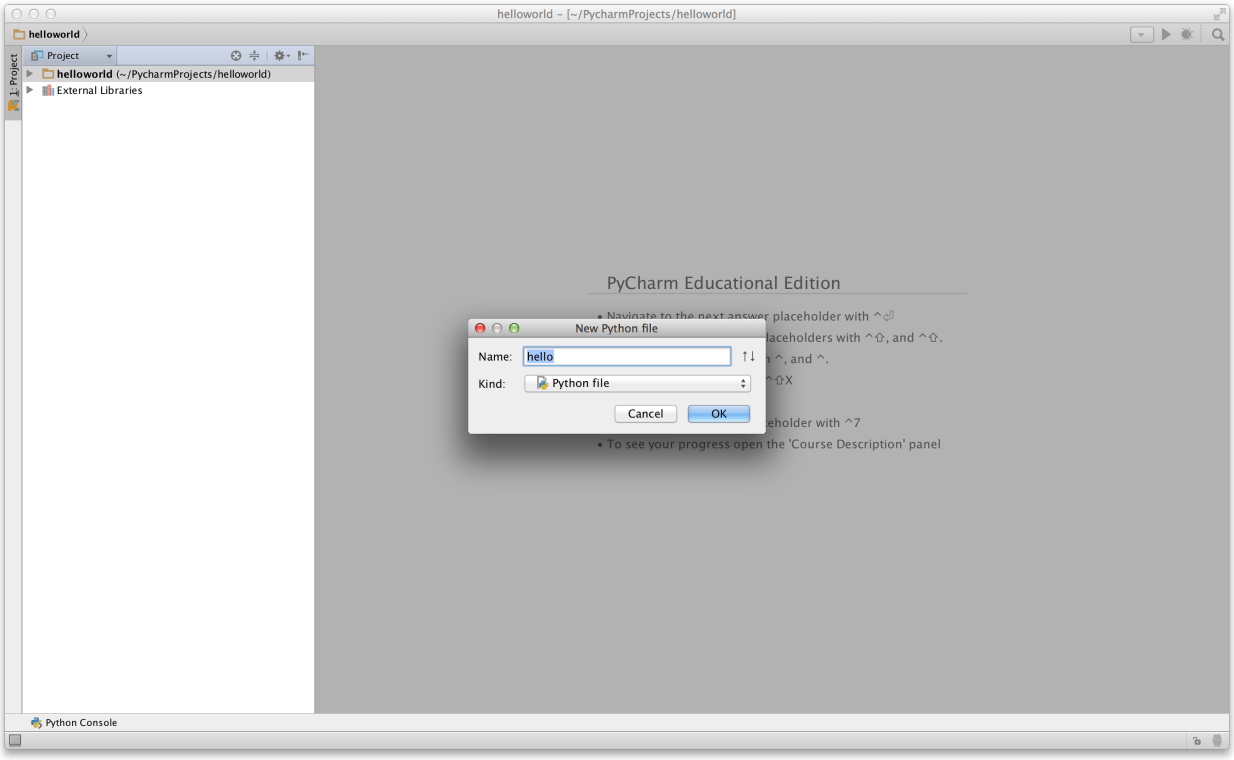


Klicken Sie auf die Schaltfläche `Create`.

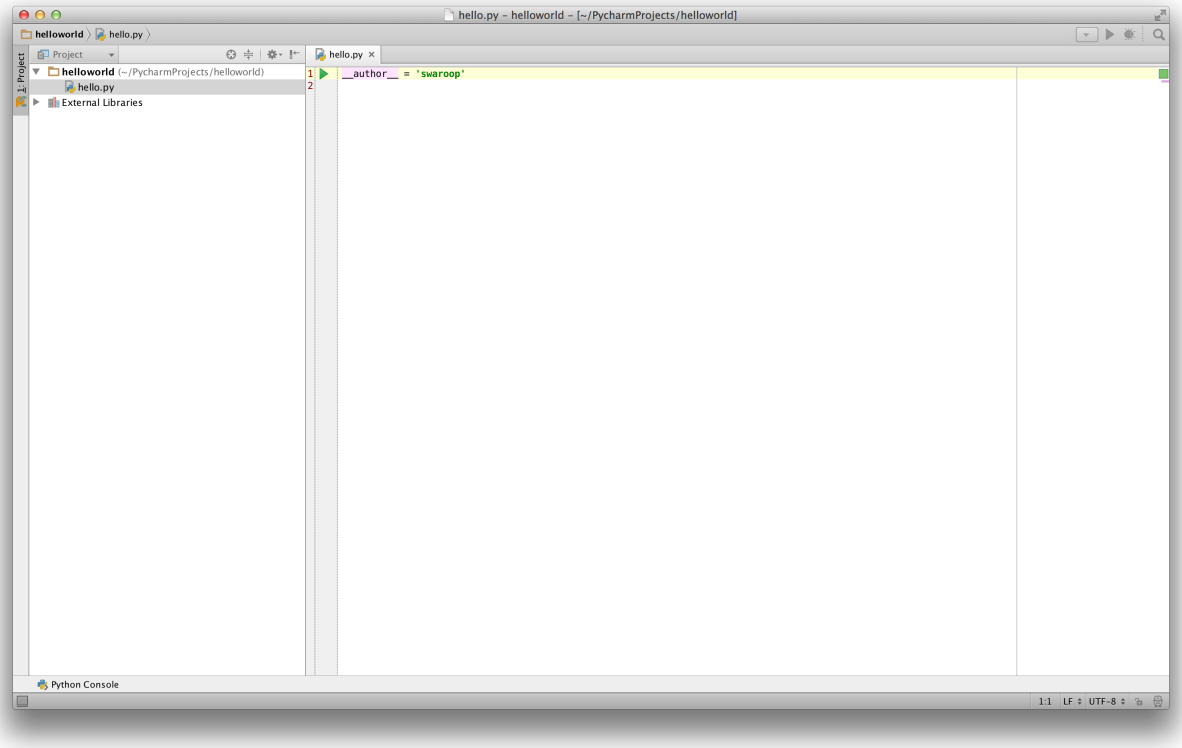
Klicken Sie mit der rechten Maustaste im Seitenbereich auf helloworld und wählen Sie `New` → `Python File`:



Sie werden aufgefordert, einen Namen einzugeben. Geben Sie ??`hello` ein:



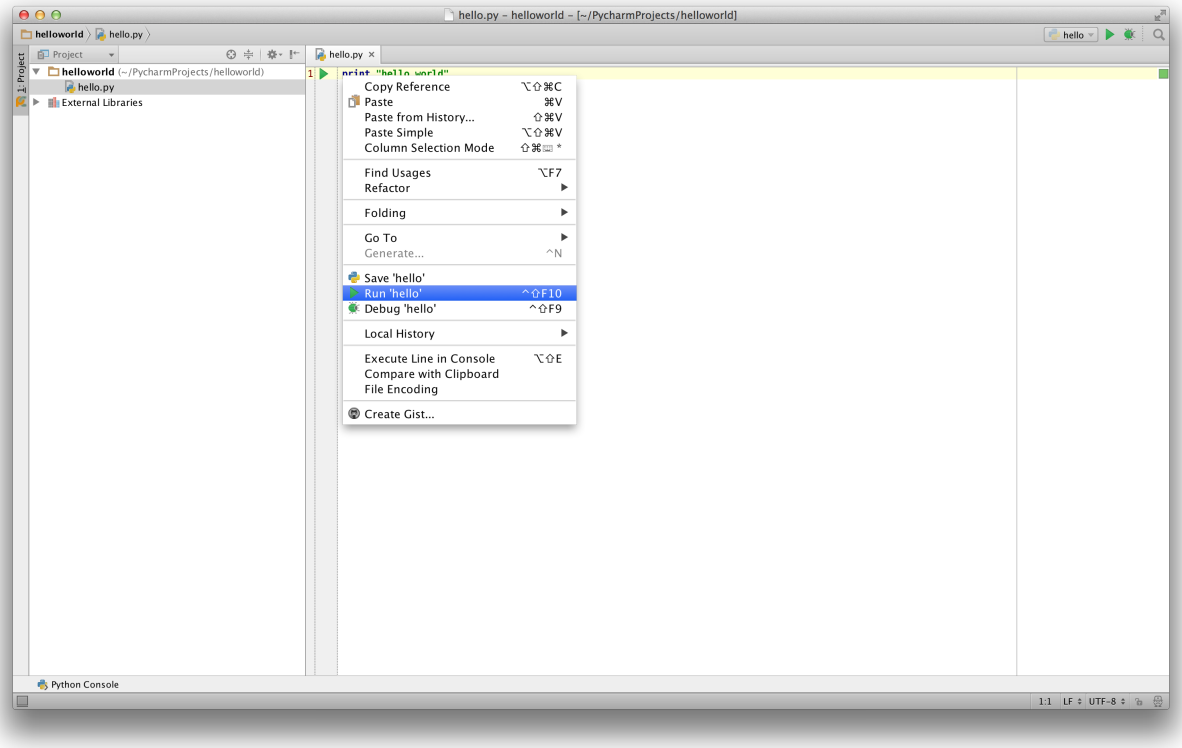
Nun sehen Sie eine geöffnete Datei:



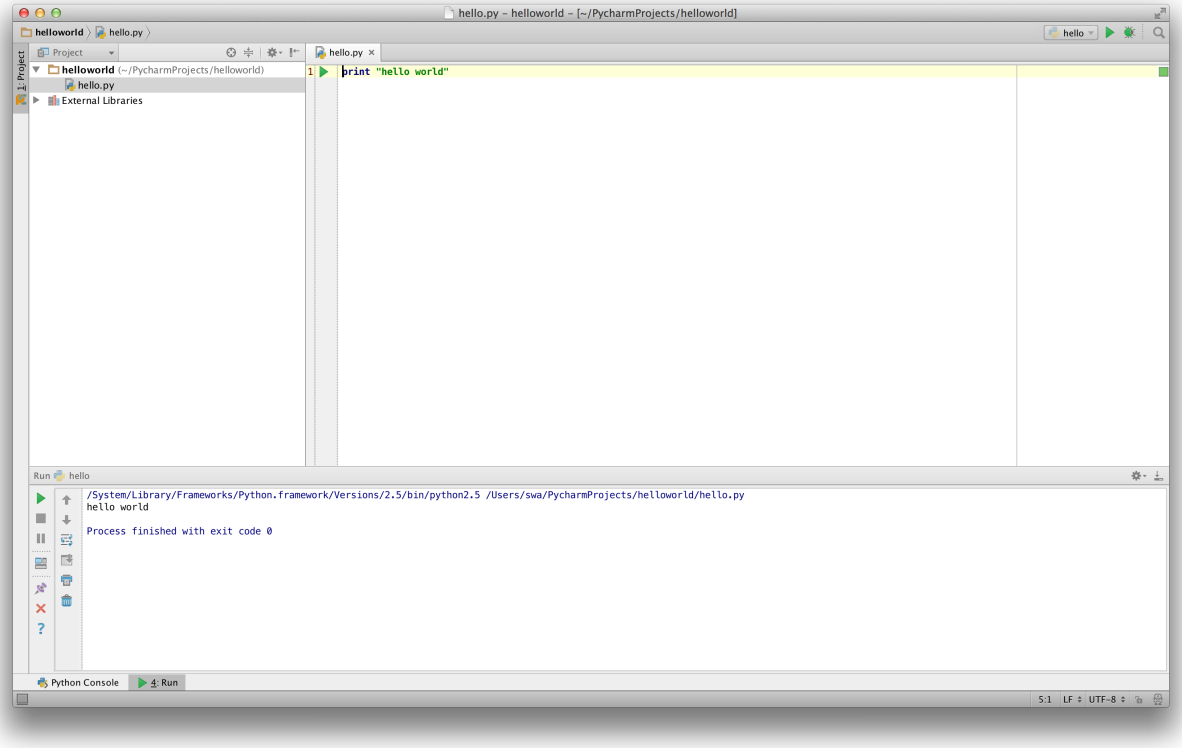
Löschen Sie die bereits vorhandenen Zeilen und geben Sie Folgendes ein:

```
print("Hallo Welt")
```

Klicken Sie nun mit der rechten Maustaste auf den eingegebenen Text (ohne ihn zu markieren) und wählen Sie Run `'hello'`.



Sie sollten nun die Ausgabe (also das, was Ihr Programm ausgibt) sehen:



Puh! Das waren einige Schritte zum Einstieg, aber von nun an: Jedes Mal, wenn Sie eine neue Datei erstellen sollen,

denken Sie daran, einfach im linken Bereich auf `helloworld` → `New` → `Python File` zu klicken und dieselben Schritte wie oben zum Schreiben und Ausführen zu befolgen.

Weitere Informationen zu PyCharm finden Sie auf der Seite PyCharm Quickstart.

5.4 Vim

1. Installieren Sie Vim
   - Mac-OS-X-Benutzer sollten das Paket `macvim` über HomeBrew installieren.
   - Windows-Benutzer sollten die „self-installing executable“ von der Vim website herunterladen.
   - GNU/Linux-Benutzer sollten Vim aus den Software-Repositories ihrer Distribution installieren, z. B. können Debian- und Ubuntu-Benutzer das Paket `vim` installieren.
2. Installieren Sie das Plugin jedi-vim
3. Installieren Sie das entsprechende `jedi`-Python-Paket: `pip install -U jedi`

5.5 Emacs

1. Installieren Sie Emacs 24+.

- Mac-OS-X-Benutzer sollten Emacs von http://emacsformacosx.com beziehen.
- Windows-Benutzer sollten Emacs von http://ftp.gnu.org/gnu/emacs/windows/ beziehen.
- GNU/Linux-Benutzer sollten Emacs aus den Repositories ihrer Distribution installieren, z. B. können Debian- und Ubuntu-Benutzer das Paket `emacs24` installieren.

2. Installieren Sie ELPY

5.6 Verwendung einer Quelldatei (Source file)

Kehren wir nun zum Programmieren zurück. Es ist Tradition, dass das erste Programm, das man schreibt und ausführt, wenn man eine neue Programmiersprache lernt, das „Hello World“-Programm ist – es gibt einfach nur „Hello World“ aus, wenn man es ausführt. Wie Simon Cozens[1] sagt, ist es die „traditionelle Beschwörung der Programmiergötter, um Ihnen zu helfen, die Sprache besser zu lernen“.

Starten Sie einen Editor Ihrer Wahl, geben Sie folgendes Programm ein und speichern Sie es als `hello.py`.

Wenn Sie PyCharm verwenden, haben wir bereits besprochen, wie man von einer Quelldatei aus ausführt.

Für andere Editoren öffnen Sie eine neue Datei `hello.py` und schreiben Folgendes hinein:

```
print("hello world")
```

Wo sollten Sie die Datei speichern? In einem beliebigen Ordner, dessen Speicherort Sie kennen. Wenn Sie nicht verstehen, was das bedeutet, erstellen Sie einen neuen Ordner und verwenden Sie diesen Ort, um dort alle Ihre Python-Programme zu speichern und auszuführen:

- `/tmp/py` unter Mac OS X
- `/tmp/py` unter GNU/Linux
- `C:\py` unter Windows

[1] Der Autor des großartigen Buchs “Beginning Perl”

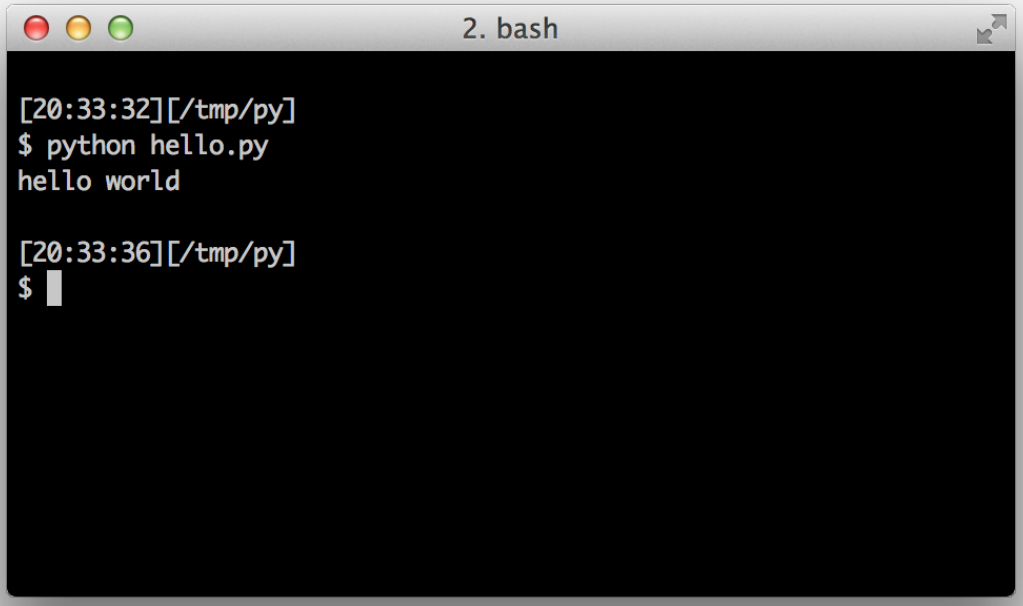
Um den oben genannten Ordner (für Ihr Betriebssystem) zu erstellen, verwenden Sie den Befehl `mkdir` im Terminal, z. B. `mkdir /tmp/py`.

WICHTIG: Achten Sie immer darauf, die Dateiendung `.py` zu verwenden, z. B. `foo.py`.

5.6.1 So führen Sie Ihr Python-Programm aus:

1. Öffnen Sie ein Terminalfenster
2. Wechseln Sie ins Verzeichnis (mittels dem *Change directory* Befehl: `cd`), in welchem Sie die Python-Datei gespeichert haben, z. B. `cd /tmp/py`.
3. Führen Sie das Programm aus, indem Sie den Befehl `python hello.py` eingeben. Die Ausgabe ist wie unten gezeigt.

```
$ python hello.py
Hallo Welt
```



Wenn Sie die Ausgabe wie oben gezeigt erhalten haben, herzlichen Glückwunsch! – Sie haben erfolgreich Ihr erstes Python-Programm ausgeführt. Sie haben den schwierigsten Teil des Programmierlernens gemeistert, nämlich den Einstieg mit Ihrem ersten Programm!

Falls ein Fehler auftrat, geben Sie das obige Programm genau wie gezeigt ein und führen Sie es erneut aus. Beachten Sie, dass Python zwischen Groß- und Kleinschreibung unterscheidet, d. h. `print` ist nicht dasselbe wie `Print` – beachten Sie das kleine `p` im ersten Fall und das große `P` im zweiten. Achten Sie außerdem darauf, dass keine Leerzeichen oder Tabulatoren vor dem ersten Zeichen jeder Zeile stehen – wir werden später sehen, warum dies wichtig ist.

Wie es funktioniert

Ein Python-Programm besteht aus Anweisungen. In unserem ersten Programm haben wir nur eine einzige Anweisung. In dieser Anweisung rufen wir die `print`-Anweisung auf und übergeben ihr den Text `"Hallo Welt"`.

5.7 Hilfe erhalten

Wenn Sie schnelle Informationen über eine Funktion oder Anweisung in Python benötigen, können Sie die eingebaute `help`-Funktion verwenden. Dies ist besonders nützlich, wenn Sie die Interpreter-Eingabeaufforderung verwenden. Führen Sie beispielsweise `help('len')` aus – dies zeigt die Hilfe für die Funktion `len` an, die verwendet wird, um die Anzahl der Elemente zu zählen.

TIPP: Drücken Sie `q`, um die Hilfe zu verlassen.

Auf ähnliche Weise können Sie Informationen über nahezu alles in Python abrufen. Verwenden Sie `help()`, um mehr über die Verwendung der Hilfe selbst zu erfahren!

Falls Sie Hilfe für Operatoren wie `return` benötigen, müssen Sie diese in Anführungszeichen setzen, etwa `help('return')`, damit Python nicht verwechselt, was wir tun wollen.

5.8 Zusammenfassung

Sie sollten nun in der Lage sein, Python-Programme problemlos zu schreiben, zu speichern und auszuführen.

Da Sie nun ein Python-Benutzer sind, wollen wir weitere Python-Konzepte erlernen.

CHAPTER 6

Grundlagen

Nur `hello world` auszugeben ist Ihnen nicht genug? Sie möchten mehr – Sie möchten Eingaben verarbeiten und ein Ergebnis erhalten. In Python erreichen wir dies mit Konstanten und Variablen. Weitere Konzepte dazu lernen wir in diesem Kapitel kennen.

6.1 Kommentare

( comments)

Kommentare sind Texte rechts neben dem #-Symbol (Auch *Raute* oder *Hashtag* genannt) und dienen hauptsächlich als Hinweise für den Leser des Programms. Python selbst ignoriert alles rechts vom # Zeichen.

Zum Beispiel:

```
print('hello world') # Beachte, dass print eine Funktion ist
```

oder:

```
# Beachte, dass print eine Funktion ist
print('hello world')
```

Verwenden Sie so viele hilfreiche Kommentare wie möglich in Ihrem Programm, um:

- Annahmen zu erläutern
- wichtige Entscheidungen zu erklären
- wichtige Details zu erläutern
- Probleme zu erläutern, die Sie lösen möchten
- Probleme zu erläutern, die Sie in Ihrem Programm überwinden möchten usw.

Dies ist für Leser Ihres Programms hilfreich, damit sie leicht verstehen können, was das Programm tut. Denken Sie daran: Jemand der Ihre Kommentare garantiert schätzen wird sind Sie selbst … ein paar Wochen in der Zukunft!

6.2 Literale Konstanten

( literal constants)

Ein Beispiel für eine literale Konstante ist eine Zahl wie `5`, `1,23` oder eine Zeichenkette wie „Dies ist eine Zeichenkette“ oder ‘Das ist eine Zeichenkette!’.

Sie wird als Literal bezeichnet, weil sie *literal* (wörtlich) ist – man verwendet ihren Wert wörtlich. Die Zahl `2` repräsentiert immer sich selbst und nichts anderes – sie ist eine *Konstante*, weil ihr Wert nicht verändert werden kann. Daher wird sie als *literal contstant* bezeichnet.

6.3 Zahlen

Zahlen lassen sich hauptsächlich in zwei Typen unterteilen: ganze Zahlen (Integer, `int`) und Gleitkommazahlen (Floats, `float`).

Beachten Sie: in Python muß immer der DEZIMALPUNKT verwendet werden anstatt dem Dezimalkomma!

Ein Beispiel für eine ganze Zahl ist `2`, eine natürliche Zahl.

Beispiele für Gleitkommazahlen (oder kurz: *float*) sind `3.23` und `52.3e-4`. Die Notation `e` steht für Zehnerpotenzen. In diesem Fall bedeutet `52.3e-4` `52.3 mal 10 hoch -4`.

Hinweis für erfahrene Programmierer

Es gibt keinen separaten Datentyp `long`. Der Datentyp `int` kann eine ganze Zahl beliebiger Größe sein.

6.4 Zeichenketten

( Strings)

Eine Zeichenkette ist eine Folge von Zeichen. Zeichenketten sind im Grunde genommen einfach eine Aneinanderreihung von Zeichen (wie z.B. Buchstaben, Ziffern, Sonderzeichen…).

Sie werden in fast jedem Python-Programm, das Sie schreiben, Zeichenketten (*strings*) verwenden. Beachten Sie daher den folgenden Abschnitt.

Strings müssen *immer* in Anführungszeichen eingesperrt sein.

6.4.1 Einfache Anführungszeichen

( single quotes)

Sie können Zeichenketten mithilfe einfacher Anführungszeichen angeben, z. B. `'Zitier mich'`.

Alle Leerzeichen und Tabulatoren innerhalb der Anführungszeichen bleiben erhalten.

6.4.2 Doppelte Anführungszeichen

( double quotes)

Zeichenketten in doppelten Anführungszeichen funktionieren genauso wie Zeichenketten in einfachen Anführungszeichen. Ein Beispiel ist `"Wie heißt Du?"`.

6.4.3 Dreifache Anführungszeichen

( triple quotes)

Mehrzeilige Zeichenketten können mit dreifachen Anführungszeichen (""" oder ''') angegeben werden. Innerhalb der dreifachen Anführungszeichen können einfache und doppelte Anführungszeichen beliebig verwendet werden. Ein Beispiel:

```
'''Dies ist eine mehrzeilige Zeichenkette. Dies ist die erste Zeile.
Dies ist die zweite Zeile.
"Wie heißt du?", fragte ich.
Er sagte: "Bond, James Bond."
'''
```

6.4.4 Zeichenketten sind unveränderlich

( strings are immutable)

Das bedeutet, dass eine einmal erstellte Zeichenkette nicht mehr verändert (mutiert) werden kann. Obwohl dies zunächst wie ein Nachteil erscheinen mag, ist es das eigentlich nicht. Wir werden später sehen, warum dies in den verschiedenen Programmen, die wir betrachten, keine Einschränkung darstellt.

Hinweis für C/C+±Programmierer

In Python gibt es keinen separaten Datentyp `char`. Er ist nicht notwendig, und Sie werden ihn sicher nicht vermissen.

Hinweis für Perl/PHP-Programmierer

Denken Sie daran, dass Strings in einfachen und doppelten Anführungszeichen identisch sind – sie unterscheiden sich in keiner Weise.

6.4.5 Die `format`-Methode

Manchmal möchten wir Strings aus anderen Informationen erstellen. Hier kommt die `format()`-Methode zum Einsatz.

Beispiel str_format_de.py

Quellcode

```
1 alter = 20
2 name = 'Swaroop'
3
4 print('{0} war {1} Jahre alt, als er dieses Buch schrieb'.format(name, alter))
5 print('Warum spielt {0} mit Python?'.format(name))
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
$ python3 str_format_de.py
Swaroop war 20 Jahre alt, als er dieses Buch schrieb
Warum spielt Swaroop mit Python?
```

So funktioniert es

Ein String kann bestimmte Spezifikationen haben. Anschließend kann die `format`-Methode aufgerufen werden, um diese Spezifikationen durch entsprechende Argumente zu ersetzen.

Betrachten wir die erste Verwendung, bei der wir `{0}` verwenden. Dies entspricht der Variable `name`, dem ersten Argument der `format`-Methode. Die zweite Spezifikation ist `{1}`, die `age` entspricht, dem zweiten Argument der `format`-Methode. Beachten Sie, dass Python bei 0 zu zählen beginnt. Das bedeutet, dass die erste Position Index 0, die zweite Position Index 1 usw. ist.

Wir hätten dasselbe auch durch Stringverkettung erreichen können:

```
name + ' ist ' + str(alter) + ' Jahre alt'
```

Das ist jedoch viel unschöner und fehleranfälliger. Außerdem würde die Umwandlung in einen String automatisch von der `format`-Methode durchgeführt, anstatt der in diesem Fall notwendigen expliziten Umwandlung. Drittens können wir mit der `format`-Methode die Nachricht ändern, ohne die verwendeten Variablen bearbeiten zu müssen, und umgekehrt.

Beachten Sie außerdem, dass die Zahlen optional sind. Sie hätten also auch Folgendes schreiben können:

```
alter = 20
name = 'Swaroop'

print('{} war {} Jahre alt, als er dieses Buch schrieb'.format(name, alter))
print('Warum spielt {} mit Python?'.format(name))
```

Dies liefert exakt dieselbe Ausgabe wie das vorherige Programm.

Wir können die Parameter auch benennen:

```
a = 20
n = 'Swaroop'

print('{name} war {alter} Jahre alt, als er dieses Buch schrieb'.format(name=n, alter=a))
print('Warum spielt {name} mit Python?'.format(name=n))
```

Dies liefert exakt dieselbe Ausgabe wie das vorherige Programm.

Python 3.6 führte eine kürzere Methode für benannte Parameter ein, sogenannte *f-Strings*:

f-strings

```
alter = 20
name = 'Swaroop'

print(f'{name} war {alter} Jahre alt, als er dieses Buch schrieb') # Beachten Sie das 'f'␣
↪vor dem String
print(f'Warum spielt {name} mit Python?') # Beachten Sie das 'f' vor dem String
```

Dies liefert exakt dieselbe Ausgabe wie das vorherige Programm.

Die `format`-Methode in Python ersetzt die Werte der Argumente durch die jeweilige Spezifikation. Es sind detailliertere Spezifikationen möglich, zum Beispiel:

Beispiel str_format2_de.py

Quellcode

```
# Dezimalzahlen (.) mit einer Genauigkeit von 3 für Gleitkommazahlen '0.333'
print('{0:.3f}'.format(1/3))

# Mit Unterstrichen (_) auffüllen und den Text zentrieren
#(^) mit einer Breite von 11 Zeichen '___hello___'
print('{0:_^11}'.format('hello'))

# Schlüsselwortbasiert
print('{name} schrieb {buch}'.format(name='Swaroop', buch='A Byte of Python'))
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
0.333
___hello___
Swaroop schrieb A Byte of Python
```

Da wir über Formatierung sprechen, beachten Sie, dass `print` immer mit einem unsichtbaren Zeilenumbruch (`\n`) endet, sodass wiederholte Aufrufe von `print` jeweils in einer separaten Zeile ausgegeben werden. Um zu verhindern, dass dieser Zeilenumbruch ausgegeben wird, können Sie festlegen, dass `print` mit einem Leerzeichen endet:

```
print('a', end='')
print('b', end='')
```

Ausgabe:

```
ab
```

Oder Sie können `print` mit einem Leerzeichen enden lassen:

```
print('a', end=' ')
print('b', end=' ')
print('c')
```

Ausgabe:

```
a b c
```

6.4.6 Escape-Sequenzen

( escape: *entwischen, entkommen*)

Angenommen, Sie möchten eine Zeichenkette mit einem einfachen Anführungszeichen (`'`) erstellen. Wie geben Sie diese Zeichenkette an? Beispielsweise lautet die Zeichenkette `"What's your name?"`. Sie können nicht einfach `'What's your name?'` angeben, da Python sonst nicht weiß, wo die Zeichenkette beginnt und endet. Daher müssen Sie angeben, dass das einfache Anführungszeichen nicht das Ende der Zeichenkette markiert. Dies geschieht mithilfe einer sogenannten Escape-Sequenz. Das einfache Anführungszeichen mitten im Textstring wird als `\'` eingegeben,

also ein Backslash gefolgt vom Anführungszeichen. Man sagt auch das Anführungszeichen wird (durch den Backslash) "*escaped*". Nun können Sie die Zeichenkette als `'What\'s your name?'` angeben.

Eine andere (etwas einfachere) Möglichkeit, diese Zeichenkette anzugeben, wäre die Verwendung von doppelten Anführungszeichen: `"What's your name?"`.

Escape-Sequencen braucht man auch um doppelte Anführungszeichen innerhalb eines *double_quote_strings* darzustellen:

```
"Ich kann schon \"Hello, World\" in Python programmieren"
```

Escape Sequenzen stammen aus der Zeit der mechanischen Schreibmaschienen und der ersten Drucker. Python interpretiert innerhalb eines *strings* folgende Escape-Sequenzen (übernommen von der Programmiersprache C):

Siehe auch

siehe auch https://docs.python.org/3/reference/lexical_analysis.html#escape-sequences

Code	Effekt	Beispiel	Ausgabe
`\ <newline>`	Ignoriert das Zeilenende	siehe Python Dokumentation	
`\\`	Rückwärts-Schrägstrich (Backslash)	`c:\\transaktionen`	`c:\transaktionen
`\'`	einfaches Anführungszeichen (Single quote)	`'what\'s up?'`	`what's up?`
`\"`	doppeltes Anführungszeichen (Double quote)	`"er sagt \"Hallo\" zu mir"`	`"er sagt "Hallo" zu mir"`
`\a`	Klingel (ASCII Bell (BEL))	`\b`	erzeugt einen Piepston
`\b`	Rücklöschetaste (ASCII Backspace (BS))		
`\f'`	Seite vor (ASCII Formfeed (FF))		
`\n`	Zeile vor (ASCII Linefeed (LF))	`"Zeile 1\nZeile 2"`	Zeile 1Zeile 2
`\r`	Druckkopf nach links (ASCII Carriage Return (CR))		
`\t`	Horizontaler Tab (ASCII Horizontal Tab (TAB))	`print("abc\tdef")`	abc def
`\v`	Vertikaler Tab (ASCII Vertical Tab (VT))		
`\ooo`	Octal character (bis zu 3 stellen)	`"\100"`	`@`
`\xhh`	Hexadecimal character (bis zu 2 stellen)	`\x40`	`@`
`\N{name}`	Named unicode char	`"\N{WHITE SMILING FACE}"`	``
`\Uxxxx`	Hexadecimal Unicode character (4 Stellen)	`"\u2614"`	``
`\ Uxxxxxxxx`	Hexadecimal Unicode character (8 Stellen)	`\U0001F060 „	``

Hinweis

In der Praxis sieht man obige Escape-Sequenzen eher selten, am ehesten noch `\n` um ein Zeilenende erzwingen und eventuell noch `\t` (für einen Tab). Man sollte sich jedoch bewusst sein daß z.B. `\t` eine Escape-Sequenz darstellt. Will man einen Backslash gefolgt von einem `t` im selben String darstellen, so muss man den Backslash *escapen* (also zwei Backslashs schreiben, um einen zu erhalten):

```
Code: `"Meine Dateien liegen auf c:\\transaktionen"`

Ausgabe: `Meine Dateien liegen auf c:\transaktionen`
```

Was aber, wenn Sie eine zweizeilige Zeichenkette angeben möchten? Eine Möglichkeit besteht darin, einen String in dreifache Anführungszeichen zu setzen, wie vorher gezeigt. Alternativ kann man eine Escape-Sequenz für das Zeilenumbruchzeichen `\n` verwenden, um den Beginn einer neuen Zeile zu kennzeichnen. Ein Beispiel:

```
'This is the first line\nThis is the second line'
```

Eine weitere nützliche Escape-Sequenz ist der Tabulator: `\t`. Es gibt noch viele weitere Escape-Sequenzen (siehe Tabelle oben)

Beachten Sie, dass ein einzelner Backslash am Ende einer Zeile in einem String anzeigt, dass der String in der nächsten Zeile fortgesetzt wird, ohne dass ein neuer Zeilenumbruch hinzugefügt wird. Zum Beispiel:

```
"This is the first sentence. \
This is the second sentence."
```

ist äquivalent zu:

```
"This is the first sentence. This is the second sentence."
```

6.4.7 Raw-strings

(raw: *roh, rau, original*)

Wenn Sie Zeichenketten angeben müssen, die nicht speziell verarbeitet werden (z. B. mit Escape-Sequenzen), verwenden Sie einen *raw*-string_ (wörtlich: Rohzeichenkette), indem Sie `r` oder `R` voranstellen. Beispiel:

```
r"Zeilenumbrüche werden durch \n gekennzeichnet"
```

Hinweis für Benutzer regulärer Ausdrücke

Verwenden Sie bei regulären Ausdrücken (*regex*) immer *raw_strings*. Andernfalls kann es zu vielen Rückwärtsreferenzen kommen. Rückwärtsreferenzen können beispielsweise mit `'\\1'` oder `r'\1'` referenziert werden.

6.5 Variable

Die Verwendung von Konstanten kann schnell langweilig werden. Man benötigt eine Möglichkeit, Informationen zu speichern und zu bearbeiten. Hier kommen *Variablen* ins Spiel. Variablen sind genau das, was der Name sagt: Ihr Wert kann variieren (sich ändern). Das heißt, Sie können beliebige Daten in einer Variable speichern. Variablen sind Speicherbereiche Ihres Computers, in denen Sie Informationen ablegen. Anders als bei Konstanten benötigen Sie eine Methode, um auf diese Variablen zuzugreifen, und geben ihnen daher Namen.

6.6 Benennung von Bezeichnern

(Identifier)

Variablen sind Beispiele für Bezeichner. *Bezeichner* sind Namen, die vergeben werden, um *etwas* zu identifizieren. Für die Benennung von Bezeichnern gelten folgende Regeln:

- Das erste Zeichen des Bezeichners *muss* ein Buchstabe des Alphabets sein (Großbuchstabe ASCII, Kleinbuchstabe ASCII oder Unicode-Zeichen) oder ein Unterstrich (_).
- Der Rest des Bezeichnernamens kann aus Buchstaben (Großbuchstabe ASCII, Kleinbuchstabe ASCII oder Unicode-Zeichen), Unterstrichen (_) oder Ziffern (0–9) bestehen.
- Bei Bezeichnernamen wird zwischen Groß- und Kleinschreibung unterschieden. Beispielsweise sind `myname` und `myName` nicht identisch. Beachten Sie das kleine `n` im ersten und das große `N` im zweiten. (Python ist *case_sensitive*)
- Beispiele für gültige Bezeichnernamen sind `i` und `name_2_3`. Beispiele für ungültige Bezeichnernamen sind `2things`, `this is spaced out`, `my-name` und `>a1b2_c3`.

6.7 Datentypen

Variablen können Werte verschiedener Typen, sogenannter Datentypen, speichern. Die grundlegenden Typen sind Zahlen und Zeichenketten, die wir bereits besprochen haben. In späteren Kapiteln werden wir sehen, wie man eigene Typen mithilfe von Klassen (siehe ./oop_de.md) erstellt.

6.8 Objekte

Python bezeichnet alles, was in einem Programm verwendet wird, als Objekt. Dies ist im allgemeinen Sinne gemeint. Anstatt „das Etwas“ zu sagen, sagen wir „das Objekt“.

Hinweis für Benutzer objektorientierter Programmierung

Python ist stark objektorientiert, da alles ein Objekt ist, einschließlich Zahlen, Zeichenketten und Funktionen.

Wir sehen uns nun an, wie man Variablen zusammen mit Literalkonstanten verwendet. Speichern Sie das folgende Beispiel und führen Sie das Programm aus.

6.9 Wie man Python-Programme schreibt

Die Standardprozedur zum Speichern und Ausführen eines Python-Programms ist fortan wie folgt:

1. Öffnen Sie Ihren bevorzugten Editor.
2. Geben Sie den im Beispiel angegebenen Programmcode ein.
3. Speichern Sie die Datei unter dem angegebenen Dateinamen, z.B. `program.py`.
4. Starten Sie den Python-Interpreter (im selben Verzeichnis in dem Sie ihre Datei gespeichert haben) mit dem Befehl `python program.py`, um das Programm auszuführen.

Manche Programmiereditoren bieten dazu eine eigene Schaltfläche oder einen Menübefehl.

6.9.1 Beispiel: Variablen und Literalkonstanten verwenden

Geben Sie das folgende Programm ein und führen Sie es aus:

Beispiel var_de.py

Quellcode

```
i = 5
print(i)
i = i + 1
print(i)

s = '''Dies ist ein mehrzeiliger String.
Dies ist die zweite Zeile.'''
print(s)
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
5
6
Dies ist ein mehrzeiliger String.
Dies ist die zweite Zeile.
```

Wie es funktioniert

So funktioniert dieses Programm: Zuerst weisen wir der Variablen `i` mit dem Zuweisungsoperator (=) den konstanten Wert `5` zu. Diese Zeile wird als Anweisung bezeichnet, da sie festlegt, dass etwas geschehen soll. In diesem Fall verknüpfen wir den Variablennamen `i` mit dem Wert `5`. Anschließend geben wir den Wert von `i` mit der `print`-Anweisung aus, die – wie der Name schon sagt – einfach den Wert der Variablen auf dem Bildschirm ausgibt.

Dann addieren wir `1` zum in `i` gespeicherten Wert und speichern das Ergebnis. Anschließend geben wir es aus und erhalten erwartungsgemäß den Wert `6`.

Analog weisen wir der Variablen `s` einen String zu und geben ihn aus.

Hinweis für Programmierer statischer Sprachen

Variablen werden verwendet, indem ihnen einfach ein Wert zugewiesen wird. Eine Deklaration oder Datentypdefinition ist nicht erforderlich.

6.10 Logische und physische Zeilen

Eine physische Zeile ist das, was Sie beim Schreiben des Programms sehen. Eine logische Zeile ist das, was Python als eine einzelne Anweisung interpretiert. Python geht implizit davon aus, dass jede physische Zeile einer logischen Zeile entspricht.

Ein Beispiel für eine logische Zeile ist die Anweisung `print('hello world')`. Wenn diese Zeile allein stünde (wie im Editor dargestellt), entspräche sie ebenfalls einer physischen Zeile.

Python fördert implizit die Verwendung einer einzelnen Anweisung pro Zeile, was den Code lesbarer macht.

Wenn Sie mehrere logische Zeilen in einer physischen Zeile definieren möchten, müssen Sie dies explizit mit einem Semikolon (`;`) kennzeichnen, das das Ende einer logischen Zeile bzw. Anweisung markiert. Zum Beispiel:

```
i = 5
print(i)
```

ist im Prinzip dasselbe wie

```
i = 5;
print(i);
```

was auch dasselbe ist wie

```
i = 5; print(i);
```

und dasselbe wie

```
i = 5; print(i)
```

Es wird dringend empfohlen nur *eine* logische Zeile pro physischer Zeile schreiben*. Verwenden Sie bitte *kein* Semikolon am Zeilenende.

Es gibt jedoch eine Situation, in der dieses Konzept sehr nützlich ist: Wenn Sie eine lange Codezeile haben, können Sie diese mithilfe des Backslashs in mehrere physische Zeilen aufteilen. Dies wird als explizite Zeilenverknüpfung bezeichnet:

```
s = 'Dies ist ein String. \
Dies setzt den String fort.'
print(s)
```

Ausgabe:

```
Dies ist ein String. Dies setzt den String fort.
```

Ebenso ist:

```
i = \
5
```

dasselbe wie

```
i = 5
```

Manchmal wird implizit angenommen, dass kein Backslash benötigt wird. Dies ist der Fall, wenn die logische Zeile eine öffnende Klammer, eckige Klammer oder geschweifte Klammer, aber keine schließende Klammer hat. Dies wird als *implizite Zeilenverknüpfung* bezeichnet. Sie können dies in späteren Kapiteln beim Programmieren mit *list* beobachten.

6.11 Einrückung

( indentation)

Leerzeichen sind in Python wichtig. Genauer gesagt: *Leerzeichen am Zeilenanfang sind wichtig*. Dies nennt man *Einrückung*. Führende Leerzeichen (Leerzeichen und Tabulatoren) am Anfang einer logischen Zeile bestimmen deren Einrückungsebene, die wiederum die Gruppierung von Anweisungen festlegt.

Das bedeutet, dass zusammengehörige Anweisungen *unbedingt* die gleiche Einrückung haben müssen. Jede solche Gruppe von Anweisungen wird als *Block* bezeichnet. Beispiele für die Bedeutung von Blöcken werden in späteren Kapiteln behandelt.

Wichtig ist, dass falsche Einrückungen zu Fehlern führen können. Zum Beispiel:

```
i = 5
# Fehler unten! Beachten Sie das einzelne Leerzeichen am Zeilenanfang:
 print('Der Wert ist', i)
print('Ich wiederhole, der Wert ist', i)
```

Beim Ausführen erhalten Sie folgenden Fehler:

```
Datei "whitespace.py", Zeile 3
 print('Der Wert ist', i)
^
Einrückungsfehler: Unerwartete Einrückung
```

Beachten Sie das einzelne Leerzeichen am Anfang der dritten Zeile. Der von Python angezeigte Fehler besagt, dass die Syntax des Programms ungültig ist, d. h. das Programm wurde nicht korrekt geschrieben. Dies bedeutet für Sie, dass Sie *nicht beliebig neue Anweisungsblöcke beginnen können* (mit Ausnahme des Standard-Hauptblocks, den Sie natürlich bisher verwendet haben). Fälle, in denen Sie neue Blöcke verwenden können, werden in späteren Kapiteln, wie z. B. dem *Kontrollfluss*, ausführlich beschrieben.

6.12 Einrücken

Verwenden Sie vier Leerzeichen für die Einrückung. Dies ist die offizielle Empfehlung der Python-Sprache. Gute Editoren erledigen das automatisch für Sie. Achten Sie darauf, eine einheitliche Anzahl von Leerzeichen für die Einrückung zu verwenden, da Ihr Programm sonst nicht ausgeführt wird oder unerwartetes Verhalten zeigt.

Hinweis für Programmierer statischer Sprachen

Python verwendet immer Einrückungen für Blöcke und niemals geschweifte Klammern. Führen Sie `from __future__ import braces` aus, um mehr zu erfahren.

6.13 Zusammenfassung

Nachdem wir nun viele Details besprochen haben, können wir uns interessanteren Themen wie Kontrollflussanweisungen zuwenden. Machen Sie sich mit dem Inhalt dieses Kapitels vertraut.

CHAPTER 7

Operatoren und Ausdrücke

- ( operators: *Operatoren*)
- ( expressions: *Ausdrücke*)

Die meisten Anweisungen (logische Zeilen), die Sie schreiben, enthalten *Ausdrücke*. Ein einfaches Beispiel für einen Ausdruck ist `2 + 3`. Ein Ausdruck kann in Operatoren und Operanden zerlegt werden.

Operatoren sind Funktionalitäten, die etwas tun, und können durch Symbole wie + oder durch spezielle Schlüsselwörter dargestellt werden. Operatoren benötigen einige Daten, auf die sie operieren, und solche Daten werden Operanden genannt. In diesem Fall sind `2` und `3` die Operanden.

7.1 Operatoren

Wir werden kurz einen Blick auf die Operatoren und ihre Verwendung werfen.

Beachten Sie, dass Sie die in den Beispielen gegebenen Ausdrücke interaktiv mit dem Interpreter auswerten können. Um zum Beispiel den Ausdruck `2 + 3` zu testen, verwenden Sie die Eingabeaufforderung des interaktiven Python-Interpreters:

```
>>> 2 + 3
5
>>> 3 * 5
15
>>>
```

Hier folgt ein schneller Überblick über die verfügbaren Operatoren:

- + (Plus)
  - Addiert zwei Objekte
  - `3 + 5` ergibt `8`. `'a' + 'b'` ergibt `'ab'`.
- - (Minus)

  - Gibt die Subtraktion einer Zahl von der anderen zurück; wenn der erste Operand fehlt, wird er als Null angenommen.
  - `-5.2` ergibt eine negative Zahl und `50 - 24` ergibt `26`.
- * (Multiplikation)
  - Gibt die Multiplikation der beiden Zahlen zurück oder liefert den String, der so oft wiederholt wird. `-2 * 3` ergibt `6`. `'la' * 3` ergibt `'lalala'`.
- ** (Potenz, power)
  - Gibt x hoch y zurück. (x{sup}y)
  - `3 ** 4` ergibt `81` (d. h. 3 * 3 * 3 * 3)
- / (Division)
  - Dividiert x durch y
  - `13 / 3` ergibt `4.333333333333333`
- // (Division und Abrunden nach unten, divide and floor)
  - Dividiert x durch y und rundet das Ergebnis nach unten auf den nächsten ganzzahligen Wert. Beachten Sie, dass Sie einen Float zurückbekommen, wenn einer der Werte ein Float ist.
  - `13 // 3` ergibt `4`
  - `-13 // 3` ergibt `-5`
  - `9 // 1.81` ergibt `4.0`
- % (Modulo)
  - Gibt den Rest der Division zurück
  - `13 % 3` ergibt `1`. (3 ist in 13 4 mal enthalten, 3 x 4 ergibt 12, bleibt auf 1 Rest auf 13)
  - `25.5 % 2.25` ergibt `1.5`.

7.1.1 Vergleichsoperatoren

- < (kleiner als)
  - Gibt zurück, ob x kleiner als y ist. Alle Vergleichsoperatoren geben `True` oder `False` zurück. Beachten Sie die Großschreibung dieser Namen.
  - `5 < 3` ergibt `False`
  - `3 < 5` ergibt `True.`
  - Vergleiche können beliebig verknüpft werden: `3 < 5 < 7` ergibt True.
- > (größer als)
  - Gibt zurück, ob x größer als y ist.
  - `5 > 3` ergibt `True`. Wenn beide Operanden Zahlen sind, werden sie zunächst in einen gemeinsamen Typ umgewandelt. Andernfalls ergibt es immer False.
- <= (kleiner oder gleich)
  - Gibt zurück, ob x kleiner oder gleich y ist
  - `x = 3; y = 6; x <= y` ergibt `True`
- >= (größer oder gleich)

  - Gibt zurück, ob x größer oder gleich y ist
  - `x = 4; y = 3; x >= 3` ergibt `True`
- == (gleich)
  - Vergleicht, ob die Objekte gleich sind
  - `x = 2; y = 2; x == y` ergibt `True`
  - `x = 'str'; y = 'stR'; x == y` ergibt `False`
  - `x = 'str'; y = 'str'; x == y` ergibt `True`
- != (ungleich)
  - Vergleicht, ob die Objekte ungleich sind
  - `x = 2; y = 3; x != y` ergibt `True`
  - `not` (boolesches NICHT)
  - Wenn `x` `True` ist, ergibt es `False`.
  - Wenn `x` `False` ist, ergibt es `True`.
  - `x = True; not x` ergibt `False`.

7.1.2 Boolsche Operatoren

Siehe auch

- https://de.wikipedia.org/wiki/Boolesche_Algebra

- and (boolesches UND)
  - `x and y` ergibt `False`, wenn `x` `False` ist, sonst ergibt es die Auswertung von `y`.
  - `x = False; y = True; x and y` ergibt `False`, da `x` `False` ist. In diesem Fall wertet Python y nicht aus, da es weiß, dass die linke Seite des 'and'-Ausdrucks `False` ist, was bedeutet, dass der gesamte Ausdruck `False` sein wird, egal wie der andere Wert lautet. Dies nennt man *short-Circuit*-Auswertung.
- `or` (boolesches ODER)
  - Wenn `x` `True` ist, ergibt es `True`, sonst ergibt es die Auswertung von `y`.
  - `x = True; y = False; x or y` ergibt `True`. -Short-Circuit-Auswertung gilt auch hier.

7.1.3 Bit-Operatoren

Siehe auch

- https://de.wikipedia.org/wiki/Bitweiser_Operator

- << (Linksverschiebung, left shift)
  - Verschiebt die Bits der Zahl nach links um die angegebene Anzahl von Bits. (Jede Zahl wird im Speicher durch Bits dargestellt, d. h. 0 und 1.) `-2 << 2` ergibt `8`. `2` wird in Bits als `10` dargestellt.
  - Eine Linksverschiebung um 2 Bits ergibt 1000, was die Dezimalzahl 8 repräsentiert.
- >> (Rechtsverschiebung, right shift)

  - Verschiebt die Bits der Zahl nach rechts um die angegebene Anzahl von Bits.
  - `11 >> 1` ergibt `5`.
  - `11` wird in Bits als `1011` dargestellt, was bei Rechtsverschiebung um 1 Bit `101` ergibt, also die Dezimalzahl `5`.
- `&` (bitweises UND)
  - Bitweises UND der Zahlen: Wenn beide Bits `1` sind, ist das Ergebnis `1`. Andernfalls ist es `0`.
  - `5 & 3` ergibt `1` (`0101 & 0011` ergibt `0001`)
- `|` (bitweises ODER)
  - Bitweises ODER der Zahlen: Wenn beide Bits `0` sind, ist das Ergebnis `0`. Andernfalls ist es `1`.
  - `5 | 3` ergibt `7` (`0101 | 0011` ergibt `0111`)
- `^` (bitweises XOR (exclusives ODER))
  - Bitweises XOR der Zahlen: Wenn beide Bits gleich sind, ist das Ergebnis 0. Andernfalls ist es 1.
  - `5 ^ 3` ergibt `6` (`0101 ^ 0011` ergibt `0110`)
- ~ (bitweises Invertieren)
  - Die bitweise Inversion von `x` ist `-(x+1)`
  - `~5` ergibt `-6`. Mehr Details unter http://stackoverflow.com/a/11810203

7.2 Abkürzung für mathematische Operation und Zuweisung

Es ist üblich, eine mathematische Operation auf einer Variablen auszuführen und dann das Ergebnis der Operation wieder der Variablen zuzuweisen; daher gibt es eine Abkürzung für solche Ausdrücke:

```
a = 2
a = a * 3
```

kann geschrieben werden als:

```
a = 2
a *= 3
```

Beachten Sie, dass `var = var operation expression` zu `var operation= expression` wird.

7.3 Auswertungsreihenfolge

Wenn Sie einen Ausdruck wie `2 + 3 * 4` haben, wird dann zuerst die Addition oder die Multiplikation durchgeführt? Unsere Mathematik aus der Schule sagt uns, dass die Multiplikation zuerst erfolgen sollte (Punktrechung vor Strichrechnung). Das bedeutet, dass der Multiplikationsoperator eine höhere Priorität besitzt als der Additionsoperator.

Die folgende Tabelle zeigt die Prioritätstabelle für Python, von der niedrigsten Priorität (geringste Bindung) zur höchsten Priorität (stärkste Bindung). Das bedeutet, dass Python in einem gegebenen Ausdruck zuerst die Operatoren und Ausdrücke weiter unten in der Tabelle auswertet, bevor es die weiter oben in der Tabelle auswertet.

Die folgende Tabelle, entnommen aus dem Python-Referenzhandbuch, wird der Vollständigkeit halber bereitgestellt. Es ist weit besser, Klammern zu verwenden, um Operatoren und Operanden angemessen zu gruppieren, um die Priorität ausdrücklich festzulegen. Dies macht das Programm lesbarer. Siehe unten Ändern der Auswertungsreihenfolge für Details.

- `lambda` : Lambda Expression
- `if - else` : Conditional expression
- `or` : Boolean OR
- `and` : Boolean AND
- `not x` : Boolean NOT
- `in, not in, is, is not, <, <=, >, >=, !=, ==` : Vergleiche, inklusive membership tests und identity tests
- `|` : Bitwise OR
- `^` : Bitwise XOR
- `&` : Bitwise AND
- `<<, >>` : Bit Shifts
- `+, -` : Addition und Subtraktion
- `*, /, //, %` : Multiplikation, Division, Ganzzahl-Division und Modulo (Rest)
- `+x, -x, ~x` : Positive, Negative, bitwise NOT
- `**` : Exponentiation
- `x[index], x[index:index], x(arguments...), x.attribute` : Subscription, slicing, call, attribute reference
- `(expressions...), [expressions...], {key: value...}, {expressions...}` : Binding or tuple display, list display, dictionary display, set display

Die Operatoren, die wir bisher noch nicht gesehen haben, werden in späteren Kapiteln erklärt.

Operatoren mit der gleichen Priorität werden in derselben Zeile der obigen Tabelle aufgeführt. Zum Beispiel haben + und - dieselbe Priorität.

7.4 Ändern der Auswertungsreihenfolge

Um Ausdrücke lesbarer zu machen, können wir Klammern verwenden. Zum Beispiel ist `2 + (3 * 4)` definitiv leichter zu verstehen als `2 + 3 * 4`, das Kenntnisse über die Operatorprioritäten erfordert. Wie bei allem sollten Klammern vernünftig verwendet werden (nicht übertreiben) und nicht redundant sein, wie in `(2 + (3 * 4))`.

Es gibt einen weiteren Vorteil bei der Verwendung von Klammern – sie helfen uns, die Auswertungsreihenfolge zu ändern. Wenn Sie zum Beispiel möchten, dass die Addition vor der Multiplikation in einem Ausdruck ausgewertet wird, können Sie etwas wie `(2 + 3) * 4` schreiben.

7.5 Assoziativität

Operatoren werden normalerweise von links nach rechts ausgewertet. Das bedeutet, dass Operatoren mit derselben Priorität in einer von links nach rechts verlaufenden Weise behandelt werden. Zum Beispiel wird `2 + 3 + 4` als `(2 + 3) + 4` ausgewertet.

7.6 Ausdrücke

Beispiel expression_de.py

Quellcode

```
1 länge = 5
2 breite = 2
3
4 fläche = länge * breite
5 print('Die Fläche ist', fläche)
6 print('Der Umfang ist', 2 * (länge + breite))
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
Die Fläche ist 10
Der Umfang ist 14
```

Wie es funktioniert:

Die Länge und Breite des Rechtecks werden in Variablen mit denselben Namen gespeichert. Wir verwenden diese, um die Fläche und den Umfang des Rechtecks mit Hilfe von Ausdrücken zu berechnen. Wir speichern das Ergebnis des Ausdrucks `länge * breite` in der Variablen namens `fläche` und drucken ihn dann mit der `print`-Funktion aus. Im zweiten Fall verwenden wir den Wert des Ausdrucks `2 * (länge + breite)` direkt in der Print-Funktion.

Beachten Sie auch, wie Python die Ausgabe „schön formatiert“. Obwohl wir keinen Abstand zwischen `'Die Fläche ist'` und der Variablen `fläche` angegeben haben, fügt Python ihn für uns ein, sodass wir eine saubere, ordentliche Ausgabe bekommen und das Programm auf diese Weise viel lesbarer ist (da wir uns keine Gedanken über Abstände in den für die Ausgabe verwendeten Strings machen müssen). Dies ist ein Beispiel dafür, wie Python dem Programmierer das Leben erleichtert.

7.7 Zusammenfassung

Wir haben gesehen, wie man Operatoren, Operanden und Ausdrücke verwendet – dies sind die grundlegenden Bausteine jedes Programms. Als Nächstes werden wir sehen, wie wir diese in unseren Programmen mittels Anweisungen verwenden.

CHAPTER 8

Kontrollfluss

( control-flow)

In den Programmen, die wir bisher gesehen haben, wurde immer eine Reihe von Anweisungen von Python exakt in der Reihenfolge von oben nach unten ausgeführt. Was wäre, wenn Sie den Ablauf ändern möchten? Wenn Sie z.B. möchten, daß das Programm Entscheidungen trifft und je nach Situation verschiedene Dinge tut, wie z.B. “Guten Morgen” oder “Guten Abend” abhängig von der Tageszeit ausgibt?

Wie Sie vielleicht bereits vermuten, wird dies durch **Kontrollfluss-Anweisungen** erreicht. In Python gibt es vier Kontrollfluss-Anweisungen: `if`, `for` und `while`. Seit Python Version 3.10 gibt es zusätzlich noch das `match` statement.

8.1 Die `if`-Anweisung

- ( if: *wenn*)
- ( else: *ansonsten*)
- ( elif (else if): *anderenfalls wenn*)

Die `if`-Anweisung wird verwendet, um eine *Bedingung* zu prüfen: **Wenn** die Bedingung wahr ist, wird ein Block von Anweisungen ausgeführt (der sogenannte *if-Block*). **Andernfalls** wird ein anderer Block von Anweisungen verarbeitet (der sogenannte *else-Block*). Die *else*-Klausel ist optional.

Pro `if`-Anweisung darf es maximal eine `else`-Anweisung geben, aber beliebig viele `elif`-Anweisungen. `elif` steht für *else if*.

Beispiel if_de.py

Quellcode

```
geheim_zahl = 23
antwort_zahl = int(input('Errate meine Zahl (Ganzzahl): '))

if antwort_zahl == geheim_zahl:
    # Anfang vom Code-Block
    print('Gratulation, richtig erraten!')
    print("Leider gibt's nichts zu gewinnen...")
    # Ende vom Code-Block
elif antwort_zahl < geheim_zahl:
    # noch ein Code-Block
    print('zu niedrig')
    # Im Code-Block können beliebig viele Python-Zeilen stehen...
else:
    print('zu hoch')
    # Dieser Block wird ausgeführt wenn die antwort_zahl
    # größer ist als die geheim_zahl

print('Fertig')
# Diese Zeile wird immer ausgeführt

```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
$ python3 if_de.py
Enter an integer : 50
No, it is a little lower than that
Done

$ python3 if_de.py
Enter an integer : 22
No, it is a little higher than that
Done

$ python3 if_de.py
Enter an integer : 23
Congratulations, you guessed it.
(but you do not win any prizes!)
Done
```

Wie es funktioniert

In diesem Programm nehmen wir Vermutungen vom Benutzer entgegen und überprüfen, ob es sich um die von uns festgelegte Zahl handelt. Wir setzen die Variable `number` auf eine beliebige Ganzzahl, z. B. `23`. Dann nehmen wir die Vermutung des Benutzers mit der Funktion `input()` entgegen. Funktionen sind wiederverwendbare Programmteile. Wir werden im *nächsten Kapitel* mehr darüber lesen.

Wir übergeben der eingebauten Funktion `input` eine Zeichenkette, die auf dem Bildschirm angezeigt wird und auf die Eingabe des Benutzers wartet. Sobald wir etwas eingeben und die **Eingabetaste** drücken, gibt die Funktion `input()` das Eingabefeld als Zeichenkette zurück. Anschließend wandeln wir diese Zeichenkette mit `int` in eine Ganzzahl um und speichern sie in der Variable `guess`. Tatsächlich ist `int` eine Klasse, aber alles, was Sie jetzt wissen müssen, ist,

dass Sie sie verwenden können, um eine Zeichenkette in eine Ganzzahl umzuwandeln (vorausgesetzt, die Zeichenkette enthält eine gültige Ganzzahl im Text).

Als Nächstes vergleichen wir die Vermutung des Benutzers mit unserer Zahl. Wenn die Vermutung mit unserer Zahl übereinstimmt, geben wir aus, dass die Vermutung richtig war. Andernfalls geben wir aus, dass die Vermutung falsch war.

8.2 Die `while`-Schleife

- ( while: *während, währenddessen, solange wie*)
- ( break: *ausbrechen, entwischen*)

Die `while`-Schleife ermöglicht es Ihnen, einen Block von Anweisungen **solange** auszuführen, wie eine Bedingung wahr ist.

Beispiel while_de.py

Quellcode

```
1  geheimzahl = 23
2  spielen = True
3
4  while spielen:
5      rate_zahl = int(input('Errate meine Geheimzahl : '))
6
7      if rate_zahl == geheimzahl:
8          print('Gratulation, das war richtig!')
9          spielen = False # die while Schleife wird nicht nochmal wiederholt
10     elif rate_zahl < geheimzahl:
11         print('zu klein')
12     else:
13         print('zu groß')
14 else:
15     print('Die while Schleife ist fertig')
16     # Hier kann beliebiger Code stehen
17
18 print('Fertig')
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
$ python3 while_de.py
Errate meine Geheimzahl : 55
zu groß
Errate meine Geheimzahl : 11
zu klein
Errate meine Geheimzahl : 23
Gratulation, das war richtig!
Die while Schleife ist fertig
Fertig
```

Wie es funktionert

In diesem Programm nehmen wir wiederholt die Eingabe des Benutzers entgegen und geben die Länge jeder Eingabe aus. Wir bieten eine spezielle Bedingung an, um das Programm zu beenden, indem wir überprüfen, ob die Benutzereingabe `'quit'` ist. Wir beenden das Programm, indem wir die Schleife mit `break` verlassen und das Ende des Programms erreichen.

Die Länge der Eingabezeichenkette kann mit der eingebauten Funktion `len` ermittelt werden.

Denken Sie daran, dass die Anweisung `break` auch mit der `for`-Schleife verwendet werden kann.

8.2.1 Swaroops poetisches Python

Die von mir hier verwendete Eingabe ist ein Mini-Gedicht, das ich geschrieben habe:

```
Programmieren macht Spaß
wenn die Arbeit getan ist.
Wenn Du Spaß an der Arbeit haben willst:
    verwende Python!
```

8.3 Die `continue`-Anweisung

- ( continue: *weitermachen, fortsetzten*)

Die `continue`-Anweisung wird verwendet, um Python mitzuteilen, dass es den Rest der Anweisungen im aktuellen Schleifenblock **überspringen** und mit der **nächsten Iteration** der Schleife fortfahren soll.

Beispiel [continue_de.py](programs/

continue_de.py)

class

note

```
while True:
    s = input('Schreib etwas : ')
    if s == 'aufhören':
        break
    if len(s) < 3:
        print('Zu wenig')
        continue
    print('Der Text ist lang genug')
    # Hier weitere Textverarbeitungen programmieren...
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
$ python3 continue_de.py
Schreib etwas : ja
Zu wenig
Schreib etwas : na
Zu wenig
Schreib etwas : jo
Zu wenig
Schreib etwas : echt?
Der Text ist lang genug
Schreib etwas : na gut
Der Text ist lang genug
Schreib etwas : aufhören
```

Wie es funktioniert

In diesem Programm nehmen wir Eingaben vom Benutzer entgegen, verarbeiten die Eingabezeichenkette jedoch nur, wenn sie mindestens 3 Zeichen lang ist. Daher verwenden wir die eingebaute Funktion `len`, um die Länge zu ermitteln. Wenn die Länge weniger als 3 beträgt, überspringen wir den Rest der Anweisungen im Block mit der Anweisung `continue`. Andernfalls werden die restlichen Anweisungen in der Schleife ausgeführt, um die gewünschte Verarbeitung durchzuführen.

Beachten Sie, dass die Anweisung `continue` auch mit der `for`-Schleife funktioniert.

8.4 Die `match ... case` Anweisung

- ( match: *zusammenpassen, übereinstimmen*)
- ( case: *Fall*)

Added in version 3.10: Die `match` Anweisung wurde in Python ab Version 3.10 eingeführt

Ähnlich zu einem `if ... elif ... else` Block unterstützt Python auch eine `match` Anweisung, die in manchen Fällen eleganteren Code erlaubt als die `if` Anweisungen.

Prinzipiell erwartet `match` einen Ausdruck und dieser Ausdruck wird dann mit mehreren `case` Blöcken verglichen. Wenn der Vergleich `True` ergibt, wird der Code im entsprechenden `case`-Block ausgeführt alle anderen `case`-Blöcke werden ignoriert (ähnlich wie beim `elif`). Die Besonderheit ist daß ein Pipe Symbol `|` als "oder" dient. Außerdem kann der Ausdruck in jedem Case Block in eine Variable (oder mehrere Variablen, bei einem Tuple / einer Liste) umgewandelt werden. Diese Variablen bleiben bestehen auch wenn der `match` Block beendet ist.

Man kann jeden `case` Befehl mit einen "Wächter" (*guard*) kombinieren: eine `if`-Anweisung welche prüft ob der `case` Block überhaupt ausgeführt werden darf. Und schlussendlich kann mit dem *-Präfix eine *beliebige* Anzahl von Variablen zusammengefasst werden (siehe auch **args* und ***kwargs* in einem späteren Kapitel.)

Ein spezieller Fall ist der `case _:` Befehl: Er kann als abschließender `case`-Block eingebaut werden und ist das Äquivalent zum `else` Befehl.

Schauen Sie sich folgendes Beispiel an:

Beispiel match1_de.py

Quellcode

```
import random
geheimzahl = random.randint(1,10)
wörter = ["eins","zwei","drei","vier","fünf",
          "sechs","sieben","acht","neun","zehn"]
print("Zahlenraten")
while True:
    print("errate meine Zahl (1-10)")
    eingabe = input("(tippe `q` oder `ende` zum aufhören) >>>")
    match eingabe:
        case "q" | "ende":
            print("Spiel beendet")
            break
        case x if x.isdigit():
            zahl = int(x)
            if not (0 < zahl < 11):
                print("Bitte nur Zahlen zwischen 1 und 10 eingeben")
                continue
        case wort if wort in wörter:
            zahl = wörter.index(wort) + 1 # index beginnt mit Null
        case _:
            print("ungültige Eingabe")
            continue
    # --- ende vom match .. case block
    if zahl == geheimzahl:
        print("Bravo richtig erraten! ich denke mir eine neue Zahl aus")
        geheimzahl = random.randint(1,10)
    elif zahl > geheimzahl:
        print("falsch geraten, zu hoch! probier es nochmal")
    else:
        print("falsch geraten, zu niedrig! probier es nochmal")


```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
$ python3 match1_de.py
Zahlenraten
errate meine Zahl (1-10)
(tippe `q` oder `ende` zum aufhören) >>>x
ungültige Eingabe
errate meine Zahl (1-10)
(tippe `q` oder `ende` zum aufhören) >>>5
falsch geraten, zu hoch! probier es nochmal
errate meine Zahl (1-10)
(tippe `q` oder `ende` zum aufhören) >>>drei
falsch geraten, zu hoch! probier es nochmal
errate meine Zahl (1-10)
(tippe `q` oder `ende` zum aufhören) >>>1
Bravo richtig erraten! ich denke mir eine neue Zahl aus
errate meine Zahl (1-10)
(tippe `q` oder `ende` zum aufhören) >>>q
Spiel beendet
```

Wie es funktioniert

Diese (leicht verbesserte) Funktion des Zahlenratenspiels akzeptiert sowohl Ziffern (`3`, `5` etc.) als Eingabe als auch ausgeschreibene Wörter wie z.B. `eins`, `zwei` usw.

Der Anfang bis inkl. Zeile 8 ist nicht wesentlich anders als das Beispiel zu `if` weiter oben. Anstatt die Variable `eingabe` mit `if...elif...else` auszuwerten wird in Zeile 9 hier ein `match ... case` Block begonnen. In Zeile 10 (dem ersten `case`-Block) wird geprüft ob der Wert von `eingabe` den Strings `"q"` *oder* `"ende"` entspricht.

Der senkrechte Strich (*pipe* genannt) fungiert als logisches oder (*or*). Die Zeilen 9 bis 12 könnte man auch schreiben als:

```
if (eingabe == "q") or (eingabe == "ende"):
    print("Spiel beendet")
    break
```

und man würde sich damit sogar eine Einrückungsebene sparen.

`match` kann aber noch ein wenig mehr, wie im nächsten `case`-Block (Zeile 13-17) demonstriert:

Hier wird der Textstring `eingabe` implizit in die *integer*-Variable `x` umgewandelt. . . aber nur wenn die Prüfung mittels `.isdigit()' den Wert True` ergibt. Dies ist Äquivalent zu:

```
if eingable.isdigit():
    x = eingabe
    zahl = int(x)
```

Die Variable `x` bleibt auch nach dem Ende des `match` Blocks bestehen, falls wir sie danach noch brauchen sollten (in diesem Beispiel wird sie aber nicht mehr gebraucht, sobald `zahl` berechnet wurde)

In Zeile 18 wird `eingabe` kurzerhand in die Variable `wort`umgewandelt, sofern der Wert von `eingabe` in der Liste `wörter` enthalten war.

In Zeile 20 fungiert der Unterstrich (_) als Auffangbecken für alle Fälle, die von keinem vorigen *case* Block behandelt wurden.

Warnung

Das folgende, komplexere Beispiel für eine `match ... case` Anweisung wird leichter verständlich wenn Sie die Kapitel *Datenstrukturen* und *Funktionen* breits gelesen haben. Andererseits: es ist ein funktionierendes Spiel und leicht erweiterbar! Starten Sie es und spielen Sie ein wenig damit herum. . .

Beispiel match2_de.py

Quellcode

```
# Textadventure, siehe https://peps.python.org/pep-0636/
dinge = {"Garten":["Rechen","Apfel","Birne"],
            "Haus":["Brot","Messer","Gabel","Teller"],
          "Straße":[],
         }
verbindungen = {"Straße": {"Süden":"Garten"},
                 "Garten": {"Norden":"Straße", "Süden":"Haus"},
```

```
                "Haus":   {"Norden":"Garten"}
                }
essbar = ("Brot","Apfel","Birne")
ort = "Haus"
was_ich_trage = [] # was ich trage
zähler = 1
print("Textadventure")
while True:
    print("---")
    print("Ich bin jetzt hier:", ort)
    print("Ich sehe hier folgende Gegenstände:", dinge[ort] )
    print("mögliche Richtungen:", list(verbindungen[ort].keys()))
    eingabe = input(f"({zähler}): kommandos (z.B. hilfe) >>>").strip()
    zähler += 1
    eingabe = eingabe.replace(",", " ") # Ersetze Komma mit Leerzeichen
    match eingabe.split():
        case ["hilfe"]:
            print("ich verstehe nur folgende Kommandos:")
            print("nimm, benutze, wirf, gehe, status, ende")
        case ["status"]:
            print("ich halte/trage momentan:", was_ich_trage)
        case ["ende"] | ["aufhören"] | ["hör", "auf"]:
            break
        case (["gehe", "nach", richtung] | ["gehe", richtung] |
              [richtung]) if richtung in verbindungen[ort]:
                ort = verbindungen[ort][richtung]
                print("ich gehe nach", richtung)
        case ["benutze", *gegenstände]:
            for g in gegenstände:
                if g not in was_ich_trage:
                    print("ich habe kein",g)
                elif g in essbar:
                    print("ich esse:", g)
                    was_ich_trage.remove(g) # zerstört gegenstand
                else:
                    print("ich benutze:",g)
        case (["nimm",  *gegenstände] | ["hebe", *gegenstände, "auf"] |
              ["hebe", "auf", *gegenstände]):
            for g in gegenstände:
                if g in dinge[ort]:
                    dinge[ort].remove(g)
                    was_ich_trage.append(g)
                    print("ich nehme", g)
                else:
                    print("da ist kein",g,"den ich nehmen könnte")
        case (["lass", "fallen", *gegenstände] |
              ["lass", *gegenstände, "fallen"] | ["wirf", *gegenstände]|
              ["wirf", *gegenstände, "weg"]):
            for g in gegenstände:
                if g not in was_ich_trage:
                    print("ich habe kein",g)
                else:
                    was_ich_trage.remove(g)
```

```
                    dinge[ort].append(g)
                    print("ich lasse fallen:", g)

        case _:
            print("ungültiges Kommando oder ungültige Aktion")
print("Spiel beendet")
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
Textadventure
---
Ich bin jetzt hier: Haus
Ich sehe hier folgende Gegenstände: ['Brot', 'Messer', 'Gabel', 'Teller']
mögliche Richtungen: ['Norden']
(1): kommandos (z.B. hilfe) >>>nimm Brot
ich nehme Brot
---
Ich bin jetzt hier: Haus
Ich sehe hier folgende Gegenstände: ['Messer', 'Gabel', 'Teller']
mögliche Richtungen: ['Norden']
(2): kommandos (z.B. hilfe) >>>hebe Messer auf
ich nehme Messer
---
Ich bin jetzt hier: Haus
Ich sehe hier folgende Gegenstände: ['Gabel', 'Teller']
mögliche Richtungen: ['Norden']
(3): kommandos (z.B. hilfe) >>>hebe auf Gabel Teller
ich nehme Gabel
ich nehme Teller
---
Ich bin jetzt hier: Haus
Ich sehe hier folgende Gegenstände: []
mögliche Richtungen: ['Norden']
(4): kommandos (z.B. hilfe) >>>gehe nach Norden
ich gehe nach Norden
---
Ich bin jetzt hier: Garten
Ich sehe hier folgende Gegenstände: ['Rechen', 'Apfel', 'Birne']
mögliche Richtungen: ['Norden', 'Süden']
(5): kommandos (z.B. hilfe) >>>nimm Apfel Birne
ich nehme Apfel
ich nehme Birne
---
Ich bin jetzt hier: Garten
Ich sehe hier folgende Gegenstände: ['Rechen']
mögliche Richtungen: ['Norden', 'Süden']
(6): kommandos (z.B. hilfe) >>>Norden
ich gehe nach Norden
---
Ich bin jetzt hier: Straße
Ich sehe hier folgende Gegenstände: []
mögliche Richtungen: ['Süden']
```

```
(7): kommandos (z.B. hilfe) >>>status
ich halte/trage momentan: ['Brot', 'Messer', 'Gabel', 'Teller', 'Apfel', 'Birne']
---
Ich bin jetzt hier: Straße
Ich sehe hier folgende Gegenstände: []
mögliche Richtungen: ['Süden']
(8): kommandos (z.B. hilfe) >>>lass fallen Messer Gabel
ich lasse fallen: Messer
ich lasse fallen: Gabel
---
Ich bin jetzt hier: Straße
Ich sehe hier folgende Gegenstände: ['Messer', 'Gabel']
mögliche Richtungen: ['Süden']
(9): kommandos (z.B. hilfe) >>>iss Brot Apfel
ungültiges Kommando oder ungültige Aktion
---
Ich bin jetzt hier: Straße
Ich sehe hier folgende Gegenstände: ['Messer', 'Gabel']
mögliche Richtungen: ['Süden']
(10): kommandos (z.B. hilfe) >>>benutze Brot Messer Apfel
ich esse: Brot
ich habe kein Messer
ich esse: Apfel
---
Ich bin jetzt hier: Straße
Ich sehe hier folgende Gegenstände: ['Messer', 'Gabel']
mögliche Richtungen: ['Süden']
(11): kommandos (z.B. hilfe) >>>ende
Spiel beendet
```

Wie es funktionert

Zeile 2 bis 10 erstellt *Datenstrukturen* (mehr dazu im entsprechenden Kapitel dieses Buches) und zwar drei *dictionaries*. `dinge` hat Textstrings als *keys* und *Listen* als *values* (erkennbar an den eckigken Klammern). Der *Key* `Straße` hat als value eine *leere Liste* (die zwei eckigen Klammern). Auch wenn Sie jetzt noch nichts von Datenstrukturen wissen könnten Sie das Programm schon erweitern, indem Sie Zeilen einfügen mit neuen Orten (der Text links vom Doppelpunkt und Listen mit Gegenständen an diesen Orten (Die Textstrings innerhalb der eckigen Klammern rechts vom Doppelpunkt).

Die `verbindungen` sind wiederum ein *dictionary*, diesmals allerdings mit *dictionaries* als *values*. Hier wird abgespeichert welcher Ort mit welchem anderen Ort verbunden ist. Ganz im Norden ist die Straße, südlich davon der Garten und ganz im Süden das Haus.

Das *Tuple* `essbar` enthält Gegenstände die der Spieler essen kann (mit dem "benutzen" Kommando)

Die Variable `ort` enthält einen Textstring um anzuzeigen wo sich dier Spieler gerade befindet und `was_ich_trage` ist derzeit eine leere Liste.

Ab Zeile 15 geht es los: Des Spieler bekommt angezeigt wo er sich befindent, welche Gegenstände es dort gibt und in welche Richtungen er gehen kann.

In Zeile 20 wird die `eingabe` des Spielers wird mittels der *Textfunktion* `.strip()` von unnötigen Leerzeichen gereinigt (Leerzeichen am Anfang und am Ende des Strings werden entfernt).

In Zeile 22 wird die `eingabe` nocheinmal mit Hilfe der *Textfunktion* `replace` verändert: Alle Beistriche werden durch

Leerzeichen ersetzt.

In Zeile 23 beginnt der `match` Block. Der auszuwertende Ausdruck ist eine Liste von Wörtern, erzeugt mittels der *Textfunktion* `split()`. Hat der Spieler z.B. den Textstring `"Hebe Apfel auf" eingegeben so erzeugt` split()`daraus die _Liste_`["Hebe", "Apfel", "auf"]`.

Zeile 24 und Zeile 27 behandeln relativ einfache Fälle: Der Spieler hat nur ein einziges Wort eingegeben. Da `split()` immer eine *Liste* erzeugt, auch wenn nur ein einzelenes Element in der Liste drin ist, muss in den `case`-Ausdrücken ein eckiges Klammernpaar stehen.

Zeile 29 verbindet mehrere Ausdrücke mittels der Pipe (`|`) zu einer logischen *oder* Verknüpfung.

Zeile 31 wird richtig kompliziert: Wiederum erzeugt die Pipe eine *oder* Verknüpfung. Da die Zeile sehr lang wird habe ich sie in mehrere physikalische Zeilen unterteilt und runde Klammern verwendet. Hier zeigt sich eine der Stärken der `match ... case` Anweisung: Der Spieler z.B. kann *Norden*, *gehe Norden* oder *gehe nach Norden* eingeben, die case Anweisung wird in allen Fällen `True` und setzt den Wert der Variabel `richtung` auf `"Norden"`. Dies funktioniert allerdings nur *wenn* der Wert von `richtung` im *directory* `verbindungen` des aktuellen `ort`'s enthalten ist.

Zeile 35 demonstriert den Stern-Präfix (*) in einem Case-Ausdruck: Sobald das erste Wort innerhalb der von `eingabe.split()` erzeugten Liste `benutze` ist, werden alle folgenden Worte in die Liste `*gegenstände` zusammengefaßt. In Zeile 36 wird daraufhin mit einer `for`-Schleife über alle Gegenstände iteriert:

```
for g in gegenstände:
    # ...
```

Hinweis

Das *-Präfix wird nur im Ausdruck des `case`-Blocks verwendet (`*gegenstände`) . Danach heißt die damit erzeugte Variable, über die iteriert werden kann, einfach `gegenstände` (ohne Stern davor).

Mehr zum Stern-Präfix gibt es im Kapitel *Funktionen* zu lesen, beim `*args` *Parameter*.

Die Zeilen 44, 53 demonstrieren das Stern-Präfix in Kombination mit einer Pipe (`|`) um logische *oder* Verknüpfungen zu erzeugen. Mittels `if` Befehl würde Zeile 44 lauten:

```
wortliste = eingabe.split()
if len(wortliste) >= 1:
    if wortliste[0] == "nimm":
        gegenstände = wortliste[1:]
elif len(wortliste) >= 2:
    if (wortliste[0] == "hebe") and (wortliste[-1] == "auf"):
        gegenstände = wortliste[1:-1]
    elif (worstliste[0] == "hebe") and (wortliste[1] == "auf"):
        gegenstände = wortliste[2:]
for g in gegenstände:
    # ...
```

Man sieht daß die Verwendung von `case` hier kürzeren, eleganteren Code ermöglicht.

Zeile 63 fungiert als `else` Block und fängt alle Fälle auf, die nicht von einem vorherigen `case`-Block behandelt wurden.

8.5 Zusammenfassung

Wir haben gesehen, wie man die drei Kontrollfluss-Anweisungen `if`, `while` und `for` zusammen mit den zugehörigen Anweisungen `break` und `continue` verwendet. Dies sind einige der am häufigsten verwendeten Teile von Python, und daher ist es unerlässlich, sich mit ihnen vertraut zu machen.

Als Nächstes werden wir sehen, wie man *Funktionen* erstellt und verwendet.

CHAPTER 9

Funktionen

- ( functions)

Funktionen sind wiederverwendbare Bestandteile von Programmen. Sie ermöglichen es Ihnen, einem Block von Anweisungen einen Namen zu geben, sodass Sie diesen Block mithilfe des angegebenen Namens überall in Ihrem Programm und beliebig oft ausführen können. Dies wird als Aufruf der Funktion bezeichnet. Wir haben bereits viele eingebaute Funktionen wie `len` und `range` verwendet.

Das Funktionskonzept ist wahrscheinlich der wichtigste Baustein jeder nicht-trivialen Software (in jeder Programmiersprache). Daher werden wir in diesem Kapitel verschiedene Aspekte von Funktionen untersuchen.

Funktionen werden mit dem Schlüsselwort `def` definiert. Nach diesem Schlüsselwort folgt ein Bezeichner als Funktionsname, gefolgt von einem Paar Klammern, das einige Variablennamen enthalten kann, und schließlich einem Doppelpunkt, der die Zeile beendet. Anschließend folgt der Block von Anweisungen, die zu dieser Funktion gehören. Ein Beispiel zeigt, dass dies tatsächlich sehr einfach ist:

Beispiel function1_de.py

Quellcode

```
1 def sag_hallo():
2     # Dieser eingerückte Code-Block
3     # gehört zur Funktion
4     print('Hallo Welt')
5 # Ende der Einrückung (und damit der Funktion)
6
7 sag_hallo()  # Funktionsaufruf (function call)
8 sag_hallo()  # noch ein function call
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
$ python function1_de.py
Hallo Welt
Hallo Welt
```

Wie es funktioniert

Wir definieren eine Funktion namens `sag_hallo` mit der oben erklärten Syntax. Diese Funktion nimmt keine Parameter entgegen und daher werden keine Variablen in den Klammern deklariert. Parameter von Funktionen stellen lediglich Eingaben dar, sodass wir unterschiedliche Werte übergeben und entsprechende Ergebnisse zurückerhalten können.

Beachten Sie, dass wir dieselbe Funktion zweimal aufrufen können, was bedeutet, dass wir denselben Code nicht erneut schreiben müssen.

9.1 Funktionsparameter

- ( function parameters)

Eine Funktion kann *Parameter* annehmen, also Werte, die Sie der Funktion übergeben, sodass die Funktion etwas tun kann, indem sie diese Werte verwendet. Diese Parameter sind ähnlich wie Variablen, mit dem Unterschied, dass ihre Werte beim Aufruf der Funktion definiert und beim Ausführen der Funktion bereits zugewiesen sind.

Parameter werden innerhalb der Klammern in der Funktionsdefinition angegeben und durch Kommata getrennt. Beim Funktionsaufruf übergeben wir die Werte in derselben Weise. Beachten Sie die Terminologie: Die in der Funktionsdefinition angegebenen Namen heißen Parameter, während die in einem Funktionsaufruf übergebenen Werte Argumente heißen.

Beispiel function_param_de.py

Quellcode

```
1  def drucke_das_maximum(a, b):
2      if a > b:
3          print(a, 'ist das Maximum')
4      elif a == b:
5          print(a, 'ist gleich groß wie', b)
6      else:
7          print(b, 'ist das Maximum')
8
9  # direkt Werte übergeben (directly pass literal values)
10 drucke_das_maximum(3, 4)
11
12 x = 5
13 y = 7
14
15 # Variablen als *argumente* übergeben
16 drucke_das_maximum(x, y)
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
$ python3 function_param_de.py
4 ist das Maximum
7 ist das Maximum
```

Wie es funktioniert

Hier definieren wir eine Funktion namens `drucke_das_maximum`, die zwei Parameter `a` und `b` verwendet. Wir bestimmen die größere Zahl mithilfe einer einfachen `if..else`-Anweisung und geben anschließend die größere der beiden Zahlen aus.

Beim ersten Funktionsaufruf von `drucke_das_maximum` übergeben wir die Zahlen direkt als *Argumente*. Im zweiten Fall rufen wir die Funktion mit Variablen als Argumente auf. `drucke_das_maximum(x, y)` bewirkt, dass der Wert des Arguments `x` dem Parameter `a` und der Wert des Arguments `y` dem Parameter `b` zugewiesen wird. Die Funktion `drucke_das_maximum` funktioniert in beiden Fällen gleich.

9.2 Lokale Variablen

- ( local variales)

Wenn Sie Variablen innerhalb einer Funktionsdefinition deklarieren, haben sie keinerlei Beziehung zu anderen Variablen mit demselben Namen außerhalb der Funktion – d. h. Variablennamen sind lokal zur Funktion. Dies wird als Gültigkeitsbereich (Scope) der Variablen bezeichnet. Alle Variablen haben den Gültigkeitsbereich des Blocks, in dem sie deklariert werden, beginnend ab dem Punkt ihrer Definition.

Beispiel function_local_de.py

Quellcode

```
1  x = 50
2
3  def func(x):
4      print('x hat den Wert', x)
5      x = 2
6      print('Lokales x geänder zu:', x)
7
8
9  func(x)
10 print('x hat noch immer den Wert', x)
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
$ python3 function_local_de.py
x ist 50
Lokales x geändert zu: 2
x ist immer noch 50
```

Wie es funktioniert

Wenn wir das erste Mal den Wert des Namens x innerhalb des Funktionskörpers ausgeben, verwendet Python den Wert des Parameters, der im Hauptblock über der Funktionsdefinition deklariert wurde.

Als Nächstes weisen wir den Wert `2` an `x` zu. Der Name `x` ist *lokal* zu unserer Funktion. Wenn wir den Wert von `x` in der Funktion ändern, bleibt das ``x, das im Hauptblock definiert wurde, unbeeinflusst.

Mit der letzten `print`-Anweisung geben wir den Wert von `x` aus, wie er im Hauptblock definiert wurde, und bestätigen damit, dass er von der lokalen Zuweisung innerhalb der zuvor aufgerufenen Funktion tatsächlich unberührt geblieben ist.

9.3 Die `global`-Anweisung

Wenn Sie einem Namen, der auf oberster Ebene eines Programms definiert wurde (d. h. nicht innerhalb eines Gültigkeitsbereichs wie Funktionen oder Klassen), einen Wert zuweisen möchten, müssen Sie Python mitteilen, dass dieser Name nicht lokal, sondern *global* ist. Dies geschieht mit der `global`-Anweisung. Es ist unmöglich, einer außerhalb einer Funktion definierten Variablen einen Wert zuzuweisen, ohne die `global`-Anweisung zu verwenden.

Sie können die Werte solcher außerhalb definierten Variablen in einer Funktion verwenden (vorausgesetzt, es gibt innerhalb der Funktion keine Variable mit demselben Namen). Dies wird jedoch nicht empfohlen, da es für den Leser des Programms unklar wird, wo die Variable definiert wurde. Die Verwendung der `global`-Anweisung macht deutlich, dass die Variable in einem äußersten Block definiert wurde.

Beispiel (als function_global_de.py speichern):

Beispiel function_global_de.py

Quellcode

```
1  x = 50
2
3
4  def funktion():
5      global x
6
7      print('x ist', x)
8      x = 2
9      print('Habe das globale x verändert zu:', x)
10
11
12 funktion()
13 print('Der Wert von x ist:', x)
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
$ python3 function_global_de.py
x ist 50
Habe das globale x verändert zu: 2
Der Wert von x ist: 2
```

Wie es funktioniert

Die `global`-Anweisung wird verwendet, um zu deklarieren, dass `x` eine globale Variable ist. Wenn wir innerhalb

der Funktion einen Wert an `x` zuweisen, wird diese Änderung im Hauptblock sichtbar, wenn wir dort den Wert von `x` verwenden.

Sie können mehrere globale Variablen mit derselben global-Anweisung angeben, z. B. `global x, y, z`.

9.4 Standardargumentwerte

- ( default-arguments)

Für einige Funktionen möchten Sie möglicherweise bestimmte Parameter optional machen und Standardwerte (*default values*) verwenden, falls der Benutzer keine Werte bereitstellt. Dies geschieht mithilfe von Standardargumentwerten. Sie können einem Parameter einen Standardwert zuweisen, indem Sie in der Funktionsdefinition nach dem Parameternamen den Zuweisungsoperator (=) gefolgt vom Standardwert angeben.

Beachten Sie, dass Standardargumentwerte Konstanten sein sollten. Genauer gesagt sollten Standardargumentwerte unveränderlich (immutable) sein – dies wird in späteren Kapiteln näher erläutert. Für jetzt genügt es, sich dies zu merken.

Beispiel function_default_de.py

Quellcode

```
1 def sag(meldung, wie_oft=1):
2     print(meldung * wie_oft)
3
4 sag('Hallo')
5 sag('Welt', 5)
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
$ python3 function_default_de.py
Hallo
WeltWeltWeltWeltWelt
```

Wie es funktioniert

Die Funktion `sag` wird verwendet, um eine Zeichenkette so oft auszugeben, wie angegeben. Wenn wir keinen Wert übergeben, wird standardmäßig nur einmal ausgegeben. Dies erreichen wir, indem wir dem Parameter `wie_oft` den Standardwert 1 zuweisen.

Im ersten Aufruf von `sag` übergeben wir nur die Zeichenkette, und sie wird einmal ausgegeben. Im zweiten Aufruf übergeben wir die Zeichenkette sowie das Argument 5, womit wir angeben, dass die Zeichenkette fünfmal ausgegeben werden soll.

Achtung

Nur jene Parameter, die am Ende der Parameterliste stehen, dürfen Standardargumentwerte besitzen. D. h. Sie können keinen Parameter mit Standardwert vor einem Parameter ohne Standardwert in der Parameterliste einer Funktion definieren.

Der Grund dafür ist, dass Werte positionsabhängig zugewiesen werden. Beispielsweise ist def `func(a, b=5)` gültig, aber `def func(a=5, b)` ist nicht gültig.

9.5 Schlüsselwortargumente

- ( keyword arguments)

Wenn Sie Funktionen mit vielen Parametern haben und nur einige davon angeben möchten, können Sie für solche Parameter Werte mithilfe ihrer Namen festlegen – dies nennt man Schlüsselwortargumente (keyword arguments). Dabei verwenden wir den Namen (Schlüsselwort) anstelle der Position (die wir bisher verwendet haben), um Argumente für die Funktion anzugeben.

Dies bietet zwei Vorteile: Erstens wird die Verwendung der Funktion einfacher, da wir uns nicht um die Reihenfolge der Argumente kümmern müssen. Zweitens können wir nur bestimmte Parameter angeben, sofern die anderen Parameter Standardwerte besitzen.

Beispiel function_keyword_de.py

Quellcode

```
1 def funktion(a, b=5, c=10):
2     print('a ist', a, 'und b ist', b, 'und c ist', c)
3
4 funktion(3, 7)
5 funktion(25, c=24)
6 funktion(c=50, a=100)
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
$ python3 function_keyword_de.py
a ist 3 und b ist 7 und c ist 10
a ist 25 und b ist 5 und c ist 24
a ist 100 und b ist 5 und c ist 50
```

Wie es funktioniert

Die Funktion mit dem Namen `funktion` hat einen Parameter ohne Standardwert, gefolgt von zwei Parametern mit Standardwerten.

Im ersten Aufruf `funktion(3, 7)` erhält der Parameter `a` den Wert `3`, der Parameter `b` den Wert `7` und `c` den Standardwert `10`.

Im zweiten Aufruf `funktion(25, c=24)` erhält `a` aufgrund der Position den Wert `25`. Danach erhält `c` aufgrund des Namens (Schlüsselwortargument) den Wert `24`. Der Parameter `b` erhält den Standardwert `5`.

Im dritten Aufruf `funktion(c=50, a=100)` verwenden wir ausschließlich Schlüsselwortargumente. Beachten Sie, dass wir den Wert für `c` vor dem Wert für `a` angeben, obwohl `a` in der Funktionsdefinition vor `c` steht.

9.6 VarArgs-Parameter

Manchmal möchten Sie eine Funktion definieren, die beliebig viele Parameter annehmen kann, d. h. eine *var*iable Anzahl von *Arg*umenten. Dies lässt sich mithilfe von Sternen erreichen:

Beispiel function_varargs_de.py

Quellcode

```
def total(a=5, *nummern, **telefonbuch):
    print('a', a)

    #iteriere durch alle Items in einem tuple
    for eine_nummer in nummern:
        print('einzelnes Item', eine_nummer)

    #iteriere durch alle Items im dictionary
    for name, nummer in telefonbuch.items():
        print(name,nummer)

total(10,1,2,3,Jack=1123,John=2231,Inge=1560)
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
$ python3 function_varargs_de.py
a 10
einzelnes Item 1
einzelnes Item 2
einzelnes Item 3
Jack 1123
John 2231
Inge 1560
```

Wie es funktioniert

Wenn wir einen Stern-Parameter wie `*nummern` deklarieren, werden alle ab dieser Position angegebenen positionsbasierten Argumente bis zum Ende gesammelt und als *tuple* namens `nummern` bereitgestellt.

Wenn wir einen Doppelstern-Parameter wie `**telefonbuch` deklarieren, werden alle Schlüsselwortargumente ab dieser Position bis zum Ende gesammelt und als *dictionary* namens `telefonbuch` bereitgestellt.

Wir werden Tupel (tuple) und Wörterbücher (dictionaries) in einem *späteren Kapitel* genauer untersuchen.

9.7 Die `return`-Anweisung

- ( return-statement)
- ( return: *zurückgeben*)

Die `return`-Anweisung wird verwendet, um aus einer Funktion zurückzukehren, d. h. die Funktion zu verlassen. Wir können optional auch einen Wert zurückgeben.

Beispiel function_return_de.py

Quellcode

```
1 def maximum(x, y):
2     if x > y:
3         return x
4     elif x == y:
5         return 'Beide Zahlen sind gleich groß'
6     else:
7         return y
8
9 print(maximum(2, 3))
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
$ python3 function_return_de.py
3
```

Wie es funktioniert

Die Funktion namens `maximum` gibt das Maximum der übergebenen Parameter zurück, in diesem Fall die übergebenen Zahlen. Sie verwendet eine einfache `if..else`-Anweisung, um den größeren Wert zu bestimmen, und gibt diesen Wert zurück.

Beachten Sie, dass `return` ohne Wert gleichbedeutend ist mit `return None`. `None` ist ein spezieller Python-Typ, der „Nichts“ repräsentiert. Beispielsweise zeigt er an, dass eine Variable keinen Wert hat, wenn ihr Wert `None` ist.

Jede Funktion enthält implizit ein `return None` am Ende, sofern Sie nicht selbst eine return-Anweisung angegeben haben. Sie können dies überprüfen, indem Sie `print(some_function())` ausführen, wobei die Funktion `some_function` keine return-Anweisung enthält, etwa:

```
def some_function():
    pass
```

Die `pass`-Anweisung wird in Python verwendet, um einen leeren Anweisungsblock zu markieren.

Tipp

Es gibt eine eingebaute Funktion namens `max`, die die „Maximum finden“-Funktionalität bereits implementiert. Verwenden Sie diese eingebaute Funktion wann immer möglich.

9.8 Dokumentations-Strings

- ( Docstrings)

Python verfügt über eine praktische Funktion namens *Documentation Strings*, üblicherweise als *DocStrings* bezeichnet. DocStrings sind ein wichtiges Werkzeug, das Sie nutzen sollten, da es hilft, Programme besser zu dokumentieren und verständlicher zu machen. Erstaunlicherweise können wir den DocString einer Funktion sogar während der Programmausführung abrufen!

Beispiel function_docstring_de.py

Quellcode

```
1  def print_max(x, y):
2      '''
3      Druckt die größere von 2 Zahlen.
4
5      Beide Zahlen müssen Ganzzahlen (integers) sein.
6      '''
7      # konvertiere die Zahlen zu Integern (sofern möglich)
8      x = int(x)
9      y = int(y)
10
11     if x > y:
12         print(x, 'ist die größere Zahl')
13     else:
14         print(y, 'ist die größere Zahl')
15
16 print_max(3, 5)
17 print(print_max.__doc__)
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
$ python3 function_docstring_de.py
5 ist die größere Zahl

    Druckt die größere von 2 Zahlen.

    Beide Zahlen müssen Ganzzahlen (integers) sein.
```

Wie es funktioniert

Eine Zeichenkette in der ersten logischen Zeile einer Funktion ist der DocString dieser Funktion. Beachten Sie, dass DocStrings auch für *Module* und *Klassen* gelten, die wir in den entsprechenden Kapiteln behandeln werden.

Die Konvention für einen DocString ist eine mehrzeilige Zeichenkette, bei der die erste Zeile mit einem Großbuchstaben beginnt und mit einem Punkt endet. Die zweite Zeile ist leer, gefolgt von einer detaillierten Erklärung ab der dritten Zeile. Sie sind dringend angehalten, diese Konvention für alle Ihre DocStrings zu befolgen, insbesondere für nicht-triviale Funktionen.

Wir können auf den DocString der Funktion `print_max` mithilfe des Attributs `__doc__` zugreifen (beachten Sie die doppelten Unterstriche). Denken Sie daran, dass Python alles als Objekt behandelt, einschließlich Funktionen. Mehr über Objekte erfahren wir im Kapitel über *Klassen*.

Wenn Sie `help()` in Python verwendet haben, haben Sie bereits DocStrings gesehen! Diese Funktion ruft einfach das Attribut `__doc__` der Funktion auf und zeigt es übersichtlich an. Sie können dies selbst ausprobieren, indem Sie `help(print_max)` in Ihr Programm einfügen. Denken Sie daran, die Taste `q` zu drücken, um help zu verlassen.

Automatisierte Werkzeuge können die Dokumentation auf diese Weise aus Ihrem Programm extrahieren. Daher empfehle ich dringend, DocStrings für jede nicht-triviale Funktion zu verwenden, die Sie schreiben. Der Befehl `pydoc`, der mit Ihrer Python-Installation geliefert wird, funktioniert ähnlich wie `help()` und nutzt ebenfalls DocStrings.

9.9 Zusammenfassung

Wir haben viele Aspekte von Funktionen betrachtet, aber dennoch nicht alle Aspekte abgedeckt. Allerdings haben wir bereits den Großteil dessen behandelt, was Sie im Alltag über Python-Funktionen wissen müssen.

Als Nächstes werden wir sehen, wie man Module verwendet und erstellt.

CHAPTER 10

Module

( modules)

Sie haben gesehen, wie Sie Code in Ihrem Programm wiederverwenden können, indem Sie Funktionen einmal definieren. Was ist, wenn Sie eine Reihe von Funktionen in anderen Programmen wiederverwenden möchten, die Sie schreiben? Wie Sie vielleicht erraten haben, lautet die Antwort: Module.

Es gibt verschiedene Methoden, Module zu schreiben, aber die einfachste Möglichkeit ist es, eine Datei mit der Erweiterung `.py` zu erstellen, die Funktionen und Variablen enthält.

Eine andere Methode besteht darin, die Module in der nativen Sprache zu schreiben, in der der Python-Interpreter selbst geschrieben wurde. Zum Beispiel können Sie Module in der Programmiersprache C schreiben, und wenn sie kompiliert wurden, können sie von Ihrem Python-Code verwendet werden, wenn Sie den Standard-Python-Interpreter benutzen.

Ein Modul kann von einem anderen Programm importiert werden, um dessen Funktionalität zu nutzen. So können wir auch die Standardbibliothek von Python verwenden. Zuerst werden wir sehen, wie man die Module der Standardbibliothek benutzt.

Beispiel module_using_sys_de.py

Quellcode

```
1 import sys
2
3 print('Die Kommandozeilenargumente sind:')
4 for i in sys.argv:
5     print(i)
6
7 print('\n\nDer PYTHONPATH lautet', sys.path, '\n')
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
# jedem Python-Programm können Kommandozeilenargumente
# mitgegeben werden (durch Leerzeichen trennen)

$ python3 module_using_sys_de.py Alice Bob Carl
Die Kommandozeilenargumente sind:
module_using_sys_de.py
Alice
Bob
Carl


Der PYTHONPATH lautet
['/home/horst/code/byte-of-python_deutsch_horst/programs',
 '/usr/lib/python312.zip', '/usr/lib/python3.12',
 '/usr/lib/python3.12/lib-dynload',
 '/usr/local/lib/python3.12/dist-packages',
 '/usr/lib/python3/dist-packages']
```

Wie es funktioniert:

Zuerst _import_ieren wir das `sys`-Modul mit der `import`-Anweisung. Im Grunde bedeutet das, dass wir Python mitteilen, dass wir dieses Modul benutzen möchten. Das `sys`-Modul enthält Funktionalität, die sich auf den Python-Interpreter und seine Umgebung bezieht, also das System.

Wenn Python die Anweisung `import sys` ausführt, sucht es nach dem Modul `sys`. In diesem Fall ist es eines der eingebauten Module, und daher weiß Python, wo es zu finden ist.

Wenn es kein kompiliertes Modul wäre, d. h. ein in Python geschriebenes Modul, würde der Interpreter in den Verzeichnissen suchen, die in seiner Variablen `sys.path` aufgeführt sind. Wenn das Modul gefunden wird, werden die Anweisungen im Körper dieses Moduls ausgeführt und das Modul wird für die Verwendung verfügbar gemacht. Beachten Sie, dass die Initialisierung nur beim ersten Import eines Moduls durchgeführt wird.

Die Variable `argv` im `sys`-Modul wird mithilfe der Punktnotation angesprochen, also `sys.argv`. Dies zeigt deutlich, dass dieser Name Teil des sys-Moduls ist. Ein weiterer Vorteil dieser Vorgehensweise besteht darin, dass der Name nicht mit einer anderen Variable namens `argv` kollidiert, falls es eine solche irgendwo in Ihrem Programm geben sollte.

Die Variable `sys.argv` ist eine Liste von Strings (Listen werden in einem *späteren Kapitel* ausführlich erklärt). Konkret enthält `sys.argv` die Liste der Kommandozeilenargumente (command line arguments), also die Argumente, die Ihrem Programm über die Kommandozeile übergeben wurden.

Wenn Sie eine IDE (*i*ntegrated *d*evelopment *e*nvironment, integrierte Entwicklungsumgebung) verwenden, um diese Programme zu schreiben und auszuführen, suchen Sie in den Menüs nach einer Möglichkeit, Kommandozeilenargumente für das Programm festzulegen.

Wenn wir hier `python3 module_using_sys_de.py Alice Bob Carl` ausführen, starten wir das Modul `module_using_sys_de.py` mit dem python-Befehl (`python3`), und `Alice`, `Bob` und `Carl` sind Argumente, die an das Programm übergeben werden. Python speichert die Kommandozeilenargumente in der Variable sys.argv, damit wir sie verwenden können.

Denken Sie daran, dass der Name des Skripts, das ausgeführt wird, immer das erste Element der Liste `sys.argv` ist. In diesem Fall haben wir also `'module_using_sys_de.py'` als `sys.argv[0]`, `'Alice'` als sys.argv[1], `'Bob'` als `sys.argv[2]` und `'Carl'` als `sys.argv[3]`. Beachten Sie, dass Python nicht bei 1, sondern bei 0 zu zählen beginnt.

`sys.path` enthält die Liste der Verzeichnisse, aus denen Module importiert werden. Beachten Sie, dass der erste String in `sys.path` leer ist – dieser leere String zeigt an, dass das aktuelle Verzeichnis ebenfalls Teil von `sys.path` ist, was dasselbe ist wie die Umgebungsvariable `PYTHONPATH`. Das bedeutet, dass Sie Module, die sich im aktuellen

Verzeichnis befinden, direkt importieren können. Andernfalls müssten Sie Ihr Modul in eines der Verzeichnisse legen, die in `sys.path` aufgeführt sind.

Beachten Sie, dass das aktuelle Verzeichnis das Verzeichnis ist, aus dem das Programm gestartet wurde. Führen Sie:

```
import os
print(os.getcwd())
```

aus, um das aktuelle Verzeichnis Ihres Programms herauszufinden. (`getcwd` steht für get c*urrent* w*orking* d*irectory*)

10.1 Byte-kompilierte `.pyc`-Dateien

Das Importieren eines Moduls ist eine relativ kostspielige Angelegenheit, daher verwendet Python einige Tricks, um es schneller zu machen. Eine Möglichkeit besteht darin, byte-kompilierte Dateien mit der Erweiterung `.pyc` zu erzeugen, was eine Zwischenform ist, in die Python das Programm übersetzt (erinnern Sie sich an den *Einführungsteil* darüber, wie Python funktioniert?). Diese `.pyc`-Datei ist nützlich, wenn Sie das Modul das nächste Mal aus einem anderen Programm importieren – es wird wesentlich schneller gehen, da ein Teil der Arbeit beim Import bereits erledigt wurde. Außerdem sind diese byte-kompilierten Dateien plattformunabhängig.

Hinweis: Diese `.pyc`-Dateien werden normalerweise im selben Verzeichnis wie die entsprechende `.py`-Datei erzeugt. Wenn Python keine Berechtigung hat, in diesem Verzeichnis zu schreiben, werden die `.pyc`-Dateien nicht erstellt.

10.2 Die `from..import` - Anweisung

Wenn Sie die Variable `argv` direkt in Ihr Programm importieren möchten (um zu vermeiden, jedes Mal `sys.` davor zu schreiben), können Sie die Anweisung `from sys import argv` verwenden.

Warnung

Im Allgemeinen sollten Sie die `from..import` - Anweisung vermeiden und stattdessen die `import` - Anweisung verwenden. Das liegt daran, dass Ihr Programm dadurch Namenskollisionen vermeidet und besser lesbar bleibt.

Beispiel (schlecht):

```
from math import sqrt
print("Die Quadratwurzel von 16 ist:", sqrt(16))
```

Beispiel (gut):

```
import math
print("Die Quadratwurzel von 16 ist:", math.sqrt(16))
```

10.3 Der `__name__` eines Moduls

Jedes Modul hat einen Namen, und Anweisungen innerhalb eines Moduls können den Namen ihres Moduls herausfinden. Das ist praktisch, um festzustellen, ob das Modul eigenständig ausgeführt wird oder importiert wird. Wie bereits erwähnt, wird der Code eines Moduls beim ersten Import ausgeführt. Wir können dies nutzen, um das Modul sich unterschiedlich verhalten zu lassen, je nachdem, ob es direkt ausgeführt oder aus einem anderen Modul importiert wird. Dies lässt sich mit dem Attribut `__name__` des Moduls erreichen.

Beispiel module_using_name_de.py

Quellcode

```
if __name__ == '__main__':
    print('Dieses Programm wurde direkt gestartet')
else:
    print('Ich wurde von einem anderen Programm importiert')
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
$ python3 module_using_name_de.py
Dieses Programm wurde direkt gestartet


$ python3
Python 3.12.3 (main, Mar 23 2026, 19:04:32) [GCC 13.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import module_using_name_de
Ich wurde von einem anderen Programm importiert
>>>
```

Wie es funktioniert:

Jedes Python-Modul hat einen definierten Namen, auf den man mittels `__name__` zugreifen kann. Wenn dieser Name `'__main__'` lautet, bedeutet das, dass das Modul vom Benutzer direkt ausgeführt wird, und wir können entsprechende Aktionen ausführen.

10.4 Eigene Module erstellen

Eigene Module zu erstellen ist einfach – Sie haben es die ganze Zeit schon getan! Das liegt daran, dass jedes Python-Programm auch ein Modul ist. Sie müssen nur sicherstellen, dass es eine `.py`-Erweiterung hat. Das folgende Beispiel sollte das deutlich machen.

Beispiel mymodule_de.py

Quellcode

```
def sag_hallo():
    print('Hallo, hier spricht mymodule_de.')

__version__ = '0.1'
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Das obige war ein Beispielmodul. Wie Sie sehen können, ist nichts Besonderes daran im Vergleich zu unseren üblichen Python-Programmen. Als Nächstes sehen wir, wie wir dieses Modul in anderen Programmen verwenden können.

Denken Sie daran, dass sich das Modul entweder im selben Verzeichnis befinden muss wie das Programm, das es importiert, oder in einem der Verzeichnisse, die in `sys.path` aufgeführt sind.

Beispiel mymodule_demo_de.py

Quellcode

```
1 import mymodule_de
2
3 mymodule_de.sag_hallo()
4 print('Version', mymodule_de.__version__)
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
$ python3 mymodule_demo_de.py
Hallo, hier spricht mymodule_de.
Version 0.1
```

Wie es funktioniert:

Beachten Sie, dass wir dieselbe Punktnotation verwenden, um die Elemente des Moduls anzusprechen. Python nutzt diese Notation konsequent, um das charakteristische „pythonic“ Gefühl zu vermitteln, sodass wir nicht ständig neue Arten lernen müssen, Dinge zu tun.

Hier ist eine Version, die die `from..import`-Syntax verwendet (speichern als mymodule_demo2_de.py):

Beispiel mymodule_demo2_de.py

Quellcode

```
1 from mymodule_de import sag_hallo, __version__
2
3 sag_hallo()
4 print('Version', __version__)
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Die Ausgabe von `mymodule_demo2_de.py` ist dieselbe wie die Ausgabe von `mymodule_demo_de.py`.

Beachten Sie, dass es zu einem Konflikt käme, wenn im importierenden Modul bereits ein Name `__version__` existiert. Das ist wahrscheinlich, da es übliche Praxis ist, dass jedes Modul seine Versionsnummer unter diesem Namen angibt. Daher wird immer empfohlen, die `import`-Anweisung zu verwenden, auch wenn das Programm dadurch etwas länger wird.

Sie könnten auch Folgendes verwenden:

```
from mymodule import *
```

Dies wird alle öffentlichen Namen wie `sag_hallo` importieren, aber nicht `__version__`, weil es mit doppelten Unterstrichen beginnt.

Warnung

Denken Sie daran, dass Sie Import-Stern vermeiden sollten, also `from mymodule import *`.

{attribution=Zen of Python (Pep20)}

Explitzit ist besser als implizit.

Tipp

Führen Sie `import this` in einem Python-prompt (>>>) aus, um mehr zu erfahren.

10.5 Die `dir`-Funktion

Die eingebaute Funktion `dir()` gibt die Liste der Namen zurück, die durch ein Objekt definiert sind. Wenn das Objekt ein Modul ist, enthält diese Liste die Funktionen, Klassen und Variablen, die innerhalb dieses Moduls definiert sind.

Diese Funktion kann Argumente annehmen. Wenn das Argument der Name eines Moduls ist, gibt die Funktion die Liste der Namen dieses Moduls zurück. Wenn es kein Argument gibt, gibt die Funktion die Liste der Namen des aktuellen Moduls zurück.

Beispiel:

```
$ python3
>>> import sys

# Namen und Attribute des sys Moduls:
>>> dir(sys)
['__displayhook__', '__doc__',
'argv', 'builtin_module_names',
'version', 'version_info']
# hier im Buch verkürzt dargestellt

# Namen und Attribute des aktuellen Moduls:
>>> dir()
['__builtins__', '__doc__',
'__name__', '__package__', 'sys']

# Eine neue Variable namens 'a' erzeugen
>>> a = 5

>>>dir()
['__builtins__', '__doc__', '__name__', '__package__', 'sys', 'a']

# Namen/Variable löschen:
>>> del a

>>> dir()
['__builtins__', '__doc__', '__name__', '__package__', 'sys']
```

Wie es funktioniert:

Zuerst sehen wir die Verwendung von `dir` auf dem importierten `sys`-Modul. Wir können die riesige Liste der Attribute

sehen, die es enthält.

Als Nächstes verwenden wir die `dir`-Funktion, ohne ihr Parameter zu übergeben. Standardmäßig gibt sie die Liste der Attribute für das aktuelle Modul zurück. Beachten Sie, dass die Liste der importierten Module ebenfalls Teil dieser Liste ist.

Um `dir` in Aktion zu sehen, definieren wir eine neue Variable `a` und weisen ihr einen Wert zu und überprüfen dann `dir`. Wir sehen, dass ein zusätzlicher Name in der Liste erscheint. Wir löschen dann diese Variable/Eigenschaft des aktuellen Moduls mit der Anweisung `del`, und die Änderung spiegelt sich erneut in der Ausgabe von `dir` wider.

Ein Hinweis zu `del`: Diese Anweisung löscht einen Variablennamen. Nachdem die Anweisung ausgeführt wurde, in diesem Fall `del a`, kann die Variable `a` nicht mehr verwendet werden – es ist, als hätte sie nie existiert.

Beachten Sie, dass die Funktion `dir()` mit jedem Objekt funktioniert. Zum Beispiel: Führen Sie `dir(str)` aus, um die Attribute der Klasse `str` zu sehen.

Es gibt auch eine Funktion `vars()`, die Ihnen möglicherweise die Attribute und deren Werte liefert, aber sie funktioniert nicht in allen Fällen.

10.6 Pakete

 packages)

Inzwischen sollten Sie begonnen haben, die Hierarchie in der Organisation Ihrer Programme zu verstehen. Variablen gehören normalerweise in Funktionen. Funktionen und globale Variablen gehören in Module. Was ist, wenn Sie Module organisieren möchten? Dafür gibt es Pakete.

Pakete sind einfach Ordner mit Modulen und einer speziellen Datei `__init__.py`, die Python anzeigt, dass dieser Ordner etwas Besonderes ist, weil er Python-Module enthält.

Angenommen, Sie möchten ein Paket namens “welt” erstellen, mit Unterpaketen namens “asien” , “afrika” usw., und diese Unterpakete enthalten wiederum Module wie “indien”, “madasgaskar” usw.

So würde die Ordnerstruktur aussehen:

```
- <Irgendein Ordner der in sys.path enthalten ist>/
    - welt/
        - __init__.py
        - asia/
            - __init__.py
            - india/
                - __init__.py
                - irgendetwas.py
        - afrika/
            - __init__.py
            - madagaskar/
                - __init__.py
                - nochetwas.py
```

Pakete sind lediglich eine Komfortfunktion, um Module hierarchisch zu organisieren. Sie werden viele Beispiele dafür in der *Python-Standardbibliothek (standard library)* sehen.

10.7 Zusammenfassung

Genauso wie Funktionen wiederverwendbare Teile von Programmen sind, sind Module wiederverwendbare Programme. Pakete sind eine weitere Hierarchie, um Module zu organisieren. Die mit Python ausgelieferte Standardbibliothek ist ein Beispiel für ein solches Set aus Paketen und Modulen.

Wir haben gesehen, wie man diese Module verwendet und wie man eigene Module erstellt.

Als Nächstes lernen wir über einige interessante Konzepte namens Datenstrukturen.

CHAPTER 11

Datenstrukturen

( data-structures)

Datenstrukturen sind im Grunde genau das – sie sind *Strukturen*, die einige *Daten* zusammenhalten können. Mit anderen Worten: Sie werden verwendet, um eine Sammlung verwandter Daten zu speichern.

In Python gibt es vier eingebaute Datenstrukturen:

Liste (*list*), Tupel (*tuple*), Wörterbuch (*dictionary*) und Menge (*set*). Wir werden uns ansehen, wie man jede davon verwendet und wie sie uns das Leben erleichtern.

Datens	Übers	Beispiel	Merkmale
list	Liste	`["abc",1,-3.14,False]`	eckige Klammern, Elemente getrennt durch Kommas. Listen sind *mutable*
tuple	Tupel	`(1,33,44,"abc")`	runde Klammern, Elemente getrennt durch Kommas. Tuples sind *immutable*
dictiona	Wörterl	`{"Vorname":"Alice", "Nachname":"Apfelbaum", "Geburtsjahr":1988}`	geschweifte Klammern, keys-value Pärchen getrennt durch Kommas, zwischen key und value steht ein Doppelpunkt. Die keys müssen *unique* sein
set	Menge	`{"Alice","Bob","Carl"}`	geschweifte Klammern, Elemente durch Kommas getrennt. Die Elemente müssen *unique* sein

11.1 Liste

Eine `list` ist eine Datenstruktur, die eine geordnete Sammlung von Elementen enthält, d. h. Sie können eine *Sequenz* von Elementen in einer Liste speichern. Dies lässt sich leicht vorstellen, wenn Sie an eine Einkaufsliste denken, auf der Sie eine Liste von Dingen haben, die Sie kaufen möchten – mit dem Unterschied, dass Sie wahrscheinlich jeden Artikel in Ihrer Einkaufsliste in einer eigenen Zeile notieren, während Sie in Python Kommas zwischen die Elemente setzen.

Die Liste von Elementen sollte in eckige Klammern eingeschlossen werden, damit Python versteht, dass Sie eine Liste angeben. Sobald Sie eine Liste erstellt haben, können Sie Elemente in der Liste hinzufügen, entfernen oder suchen. Da

wir Elemente hinzufügen und entfernen können, sagen wir, dass eine Liste ein *veränderbarer* (*mutable*) Datentyp ist, d. h. dieser Typ kann verändert werden.

11.2 Kurze Einführung in Objekte und Klassen

Obwohl ich die Diskussion über Objekte und Klassen bisher im Allgemeinen hinausgezögert habe, ist an dieser Stelle eine kleine Erklärung notwendig, damit Sie Listen besser verstehen können. Wir werden dieses Thema in einem *späteren Kapitel* ausführlich behandeln.

Eine Liste ist ein Beispiel für die Verwendung von Objekten und Klassen. Wenn wir eine Variable `i` verwenden und ihr einen Wert zuweisen, beispielsweise die Ganzzahl `5`, dann können Sie sich das so vorstellen, dass ein *Objekt* (d. h. eine Instanz) `i` der *Klasse* (d. h. des Typs) `int` erstellt wird. Tatsächlich können Sie `help(int)` lesen, um dies besser zu verstehen.

Eine Klasse kann auch *Methoden* besitzen, d. h. Funktionen, die ausschließlich für die Verwendung mit dieser Klasse definiert sind. Sie können diese Funktionalitäten nur verwenden, wenn Sie ein Objekt dieser Klasse besitzen. Beispielsweise stellt Python eine `append`-Methode für die Klasse `list` bereit, mit der Sie ein Element an das Ende der Liste anhängen können. Zum Beispiel fügt `meine_liste.append('Mein Element')` die Zeichenkette `"Mein Element"` hinten and die Liste `meine_liste` dazu. Beachten Sie die Verwendung der *Punktnotation* für den Zugriff auf Methoden von Objekten.

Eine Klasse kann außerdem *Attribute* (Felder) besitzen, die nichts anderes als Variablen sind, die ausschließlich zur Verwendung mit dieser Klasse definiert wurden. Sie können diese Variablen/Attribute ebenfalls nur verwenden, wenn Sie ein Objekt dieser Klasse besitzen. Auch auf Felder wird mit der Punktnotation zugegriffen, beispielsweise `meine_liste.mein_feld`.

Beispiel ds_using_list_de.py

Quellcode

```
# Meine Einkaufsliste:
einkaufsliste = ['Apfel', 'Mango', 'Karotte', 'Banane']

print('Ich habe', len(einkaufsliste), 'Sachen zu kaufen.')

print('Und zwar:', end=' ')
for item in einkaufsliste:
    print(item, end=' ')

print('\nI muss auch Reis kaufen.')
einkaufsliste.append('Reis')
print('Meine Einkaufsliste schaut jetzt so aus:',einkaufsliste)

print('Ich sortiere die Einkaufsliste alphabetisch')
einkaufsliste.sort()  # sortiert "in place"
print('Sortierte Einkufsliste:', einkaufsliste)

print('Zuerst kaufe ich:', einkaufsliste[0])
altes_item = einkaufsliste[0]
del einkaufsliste[0]
print('I habe schon gekauft: ', altes_item)
print('Meine Einkaufsliste lautet jetzt: ', einkaufsliste)
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
$ python3 ds_using_list_de.py
Ich habe 4 Sachen zu kaufen.
Und zwar: Apfel Mango Karotte Banane
I muss auch Reis kaufen.
Meine Einkaufsliste schaut jetzt so aus: ['Apfel', 'Mango', 'Karotte', 'Banane', 'Reis
↪']
Ich sortiere die Einkaufsliste alphabetisch
Sortierte Einkufsliste: ['Apfel', 'Banane', 'Karotte', 'Mango', 'Reis']
Zuerst kaufe ich: Apfel
I habe schon gekauft:  Apfel
Meine Einkaufsliste lautet jetzt:  ['Banane', 'Karotte', 'Mango', 'Reis']
```

Wie es funktioniert

Die Variable `einkaufsliste` ist eine Einkaufsliste für jemanden, der zum Markt geht. In `einkaufsliste` speichern wir nur Zeichenketten mit den Namen der zu kaufenden Artikel, aber Sie können *jede Art von Objekt* zu einer Liste hinzufügen, einschließlich Zahlen und sogar anderer Listen.

Wir haben außerdem die Schleife `for..in` verwendet, um die Elemente der Liste zu durchlaufen (wir *iterieren* über die Liste). Inzwischen sollten Sie erkannt haben, dass eine Liste ebenfalls eine *Sequenz* ist. Die Besonderheiten von Sequenzen werden in einem [späteren Abschnitt] behandelt.

Beachten Sie die Verwendung des Parameters `end` im Aufruf der Funktion `print`, um anzugeben, dass wir die Ausgabe mit einem Leerzeichen statt mit dem üblichen Zeilenumbruch beenden möchten.

Als Nächstes fügen wir der Liste mithilfe der Methode `append` des Listenobjekts ein Element hinzu, wie bereits zuvor besprochen. Danach überprüfen wir, dass das Element tatsächlich der Liste hinzugefügt wurde, indem wir den Inhalt der Liste ausgeben. Dazu übergeben wir die Liste einfach an die Funktion `print`, die sie übersichtlich darstellt.

Danach sortieren wir die Liste mithilfe der Methode `sort` der Liste. Es ist wichtig zu verstehen, dass diese Methode die Liste selbst verändert (*in_place*) und keine veränderte Liste zurückgibt – dies unterscheidet sich von der Funktionsweise von Zeichenketten. Genau das meinen wir, wenn wir sagen, dass Listen *veränderbar* (*mutable*) und Zeichenketten *unveränderlich* (*immutable*) sind.

Wenn wir anschließend auf dem Markt einen Artikel gekauft haben, möchten wir ihn aus der Liste entfernen. Dies erreichen wir mithilfe der Anweisung `del`. Hier geben wir an, welches Element der Liste entfernt werden soll, und die Anweisung `del` entfernt es für uns aus der Liste. Wir geben an, dass wir das erste Element aus der Liste entfernen möchten, und verwenden daher `del einkaufsliste[0]` (denken Sie daran, dass Python immer mit 0 zu zählen beginnt).

Wenn Sie alle Methoden kennenlernen möchten, die für das Listenobjekt definiert sind, lesen Sie `help(list)` für weitere Details.

11.3 Tupel

Tupel werden verwendet, um mehrere Objekte zusammenzuhalten. Sie ähneln Listen, jedoch ohne die umfangreiche Funktionalität, die Ihnen die Klasse `list` bietet. Eine wesentliche Eigenschaft von Tupeln ist, dass sie – genau wie Zeichenketten – *unveränderlich* (*immutable*) sind, d. h. Sie können Tupel nicht verändern.

Tupel werden definiert, indem Elemente angegeben werden, die durch Kommas getrennt und optional von einem Paar runder Klammern umschlossen sind.

Tupel werden üblicherweise in Fällen verwendet, in denen eine Anweisung oder eine benutzerdefinierte Funktion sicher davon ausgehen kann, dass sich die Sammlung von Werten (d. h. das verwendete Tupel von Werten) nicht ändern wird.

Beispiel ds_using_tuple_de.py

Quellcode

```
# Ich empfehle, immer runde Klammern zu verwenden
# um Start und Ende eines Tuples zu kennzeichnen
# (Obwohl die runden Klammern optional sind)
# Explizit ist besser als implizit
zoo = ('Python', 'Elefant', 'Pinguin')
print('Anzahl der Tiere im Zoo:', len(zoo))

neuer_zoo = 'Affe', 'Kamel', zoo    # Klammern sind nicht notwendig, aber eine gute␣
↪Idee
print('Anzahl der Käfige im Zoo:', len(neuer_zoo))
print('Alle Tiere im neuen Zoo:', neuer_zoo)
print('Tiere vom alten Zoo:', neuer_zoo[2])
print('Das letzte Tier vom alten Zoo:', neuer_zoo[2][2])
print('Anzahl der Tiere im neuen Zoo: ',
      len(neuer_zoo)-1+len(neuer_zoo[2]))
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
$ python3 ds_using_tuple_de.py
Anzahl der Tiere im Zoo: 3
Anzahl der Käfige im Zoo: 3
Alle Tiere im neuen Zoo: ('Affe', 'Kamel', ('Python', 'Elefant', 'Pinguin'))
Tiere vom alten Zoo: ('Python', 'Elefant', 'Pinguin')
Das letzte Tier vom alten Zoo: Pinguin
Anzahl der Tiere im neuen Zoo:  5
```

Wie es funktioniert

Die Variable `zoo` verweist auf ein Tupel von Elementen. Wir sehen, dass die Funktion `len` verwendet werden kann, um die Länge des Tupels zu bestimmen. Dies zeigt ebenfalls, dass ein Tupel eine Sequenz ist.

Wir verlegen diese Tiere nun in einen neuen Zoo, da der alte Zoo geschlossen wird. Daher enthält das Tupel `neuer_zoo` einige Tiere, die bereits dort sind, zusammen mit den Tieren, die aus dem alten Zoo übernommen wurden. Zurück zur Realität: Beachten Sie, dass ein Tupel innerhalb eines Tupels seine Identität nicht verliert.

Wir können auf die Elemente im Tupel zugreifen, indem wir die Position des Elements innerhalb eines Paares eckiger Klammern angeben, genau wie wir es bei Listen getan haben. Dies wird als *Indexierungsoperator* bezeichnet. Wir greifen auf das dritte Element in `neuer_zoo` zu, indem wir `neuer_zoo[2]` angeben, und wir greifen auf das dritte Element innerhalb des dritten Elements des Tupels `neuer_zoo` zu, indem wir `new_zoo[2][2]` angeben. Dies ist ziemlich einfach, sobald Sie das Prinzip verstanden haben.

Tupel mit 0 oder 1 Elementen

Ein leeres Tupel wird durch ein leeres Paar runder Klammern erzeugt, beispielsweise `nichts_drin = ()`. Ein Tupel mit nur einem einzelnen Element ist jedoch nicht so einfach. Sie müssen es mit einem Komma nach dem ersten (und einzigen) Element angeben, damit Python zwischen einem Tupel und einem Paar runder Klammern um ein Objekt in einem Ausdruck unterscheiden kann, d. h. Sie müssen `einsertupel = (2 , )` angeben, wenn Sie

ein Tupel mit dem Element 2 meinen.

Hinweis für Perl-Programmierer

Eine Liste innerhalb einer Liste verliert ihre Identität nicht, d. h. Listen werden nicht wie in Perl „abgeflacht“. Dasselbe gilt für ein Tupel innerhalb eines Tupels, ein Tupel innerhalb einer Liste oder eine Liste innerhalb eines Tupels usw. Aus Sicht von Python handelt es sich lediglich um Objekte, die mithilfe eines anderen Objekts gespeichert werden – mehr nicht.

11.4 Dictionary

- ( dictonary: *Wörterbuch, Verzeichnis*)
- ( key: *Schlüssel, Einträge*)
- ( values: *Werte*)
- ( unique: *eindeutig, einzigartig*)

Ein *Dictionary* ist wie ein Adressbuch, in dem Sie die Adresse oder Kontaktdaten einer Person finden können, von welcher Sie nur den Namen kennen, d. h. wir verknüpfen *Schlüssel* (*keys*) (Name) mit *Werten* (*values*) (Details). Beachten Sie, dass der Schlüssel eindeutig (*unique*) sein muss – genauso wie Sie die korrekten Informationen nicht finden können, wenn zwei Personen exakt denselben Namen haben.

Beachten Sie, dass Sie nur unveränderliche (*immutable*) Objekte (wie *Strings*) als Schlüssel eines Dictionarys verwenden können, während Sie sowohl unveränderliche als auch veränderliche Objekte als Werte eines Dictionarys verwenden können. Dies bedeutet im Wesentlichen, dass Sie nur einfache Objekte als Schlüssel verwenden sollten.

Paare aus Schlüsseln und Werten werden in einem Dictionary mithilfe der Notation `d = {key1 : value1, key2 : value2 }` angegeben. Beachten Sie, dass Schlüssel-Wert-Paare durch einen Doppelpunkt getrennt sind, die einzelnen Paare wiederum durch Kommas getrennt werden und das Ganze in geschweifte Klammern eingeschlossen ist.

Denken Sie daran, dass Schlüssel-Wert-Paare in einem Dictionary in keiner Weise geordnet sind. Wenn Sie eine bestimmte Reihenfolge wünschen, müssen Sie diese selbst sortieren, bevor Sie sie verwenden.

Die Dictionarys, die Sie verwenden werden, sind Instanzen/Objekte der Klasse `dict`.

Beispiel ds_using_dict_de.py

Quellcode

```
# 'ab' steht für AdressBuch

ab = {
    'Swaroop': 'swaroop@swaroopch.com',
    'Larry': 'larry@wall.org',
    'Matsumoto': 'matz@ruby-lang.org',
    'Spammer': 'spammer@hotmail.com'
}

print("Swaroop's Email ist", ab['Swaroop'])

```

```
# Löschen eines key-value Paares
del ab['Spammer']

# key-value Paare zählen
print('\nEs sind {} Kontakte im Adressbuch\n'.format(len(ab)))

# iterieren über das dictionary
for name, addresse in ab.items():
    print('Die Email von  {} ist {}'.format(name, addresse))

# key-value Paar hinzufügen
ab['Guido'] = 'guido@python.org'

# testen ob ein Key im Dictionary drin ist
if 'Guido' in ab:
    print("\nGuido's email lautet", ab['Guido'])
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
$ python3 ds_using_dict_de.py
Swaroop's Email ist swaroop@swaroopch.com

Es sind 3 Kontakte im Adressbuch

Die Email von  Swaroop ist swaroop@swaroopch.com
Die Email von  Larry ist larry@wall.org
Die Email von  Matsumoto ist matz@ruby-lang.org

Guido's email lautet guido@python.org
```

Wie es funktioniert

Wir erstellen das Dictionary `ab` mithilfe der bereits besprochenen Notation. Anschließend greifen wir auf Schlüssel-Wert-Paare zu, indem wir den Schlüssel mithilfe des Indexierungsoperators angeben, wie wir es bereits im Zusammenhang mit Listen und Tupeln besprochen haben. Beachten Sie die einfache Syntax.

Wir können Schlüssel-Wert-Paare mithilfe unseres alten Bekannten – der Anweisung `del` – löschen. Wir geben einfach das Dictionary und den Indexierungsoperator für den zu entfernenden Schlüssel an und übergeben dies an die Anweisung `del`. Für diese Operation ist es nicht notwendig, den Wert zu kennen, der dem Schlüssel entspricht.

Anschließend greifen wir mithilfe der Methode `items` des Dictionarys auf jedes Schlüssel-Wert-Paar zu. Diese Methode gibt eine Liste von Tupeln zurück, wobei jedes Tupel ein Paar von Elementen enthält – zuerst den Schlüssel, gefolgt vom Wert. Mithilfe der Schleife `for..in` holen wir dieses Paar ab und weisen es den Variablen `name` und `addresse` entsprechend zu. Danach geben wir diese Werte innerhalb des `for`-Blocks aus.

Wir können neue *key-value*-Paare hinzufügen, indem wir einfach den Indexierungsoperator verwenden, um auf einen Schlüssel zuzugreifen und diesem einen Wert zuzuweisen, wie wir es im obigen Beispiel für Guido getan haben.

Wir können mithilfe des Operators `in` überprüfen, ob ein Schlüssel-Wert-Paar existiert.

Eine Liste der Methoden der Klasse `dict` finden Sie unter `help(dict)`.

Schlüsselwortargumente und Dictionarys

Wenn Sie in Ihren Funktionen Schlüsselwortargumente verwendet haben, dann haben Sie bereits Dictionarys benutzt! Denken Sie einmal darüber nach – das Schlüssel-Wert-Paar wird von Ihnen in der Parameterliste der Funktionsdefinition angegeben, und wenn Sie innerhalb Ihrer Funktion auf Variablen zugreifen, ist dies lediglich ein Schlüsselzugriff auf ein Dictionary (das in der Terminologie des Compilerbaus als *Symboltabelle* (*symbol table*) bezeichnet wird).

11.5 Sequenz

Listen, Tupel und Zeichenketten sind Beispiele für Sequenzen – aber was sind Sequenzen und was ist so besonders an ihnen?

Die wichtigsten Eigenschaften sind *Mitgliedschaftstests* (d. h. die Ausdrücke `in` und `not in`) sowie *Indexierungsoperationen*, die es uns ermöglichen, direkt ein bestimmtes Element der Sequenz abzurufen.

Die drei oben genannten Sequenztypen – Listen, Tupel und Zeichenketten – besitzen außerdem eine *Slicing*-Operation, die es uns erlaubt, einen Ausschnitt der Sequenz, d. h. einen Teil der Sequenz, abzurufen.

Beispiel ds_seq_de.py

Quellcode

```
einkaufsliste = ['Apfel', 'Mango', 'Karotte', 'Banane']
name = 'Swaroop'

# Indexing oder 'Subscription' operation #
print('Item 0 ist', einkaufsliste[0])
print('Item 1 ist', einkaufsliste[1])
print('Item 2 ist', einkaufsliste[2])
print('Item 3 ist', einkaufsliste[3])
print('Item -1 ist', einkaufsliste[-1])
print('Item -2 ist', einkaufsliste[-2])
print('Buchstabe 0 ist', name[0])

# Slicing einer Liste #
print('Items mit Index 1 bis 3 sind', einkaufsliste[1:3])
print('Items mit Index 2 bis (inkl.) zum Ende sind', einkaufsliste[2:])
print('Items mit Index 1 bis Index -1 sind', einkaufsliste[1:-1])
print('Items vom Start bis zum Ende sind', einkaufsliste[:])

# Slicing eines Strings #
print('Buchstaben mit Index 1 bis 3 sind', name[1:3])
print('Buchstaben mit Index 2 bis (inkl.) zum Ende sind', name[2:])
print('Buchstaben mit Index 1 bis -1 sind', name[1:-1])
print('Buchstaben vom Start bis zum Ende sind', name[:])
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
$ python3 ds_seq_de.py
Item 0 ist Apfel
Item 1 ist Mango
Item 2 ist Karotte
Item 3 ist Banane
Item -1 ist Banane
Item -2 ist Karotte
Buchstabe 0 ist S
Items mit Index 1 bis 3 sind ['Mango', 'Karotte']
Items mit Index 2 bis (inkl.) zum Ende sind ['Karotte', 'Banane']
Items mit Index 1 bis Index -1 sind ['Mango', 'Karotte']
Items vom Start bis zum Ende sind ['Apfel', 'Mango', 'Karotte', 'Banane']
Buchstaben mit Index 1 bis 3 sind wa
Buchstaben mit Index 2 bis (inkl.) zum Ende sind aroop
Buchstaben mit Index 1 bis -1 sind waroo
Buchstaben vom Start bis zum Ende sind Swaroop
```

Wie es funktioniert

Zunächst sehen wir uns an, wie Indizes verwendet werden, um einzelne Elemente einer Sequenz abzurufen. Dies wird auch als *Subskriptionsoperation* bezeichnet. Immer wenn Sie einer Sequenz innerhalb eckiger Klammern eine Zahl angeben, wie oben gezeigt, liefert Ihnen Python das Element, das dieser Position in der Sequenz entspricht. Denken Sie daran, dass Python bei 0 mit dem Zählen beginnt. Daher liefert `einkaufsliste[0]` das erste Element und `einkaufsliste[3]` das vierte Element der Sequenz `einkaufsliste`.

Der Index kann auch eine negative Zahl sein. In diesem Fall wird die Position vom Ende der Sequenz aus berechnet. Daher bezieht sich `einkaufsliste[-1]` auf das letzte Element der Sequenz und `einkaufsliste[-2]` liefert das vorletzte Element der Sequenz.

Die *Slicing*-Operation wird verwendet, indem der Name der Sequenz gefolgt von einem optionalen Zahlenpaar angegeben wird, das durch einen Doppelpunkt innerhalb eckiger Klammern getrennt ist. Beachten Sie, dass dies der Indexierungsoperation sehr ähnlich ist, die Sie bisher verwendet haben. Denken Sie daran, dass die Zahlen optional sind – der Doppelpunkt jedoch nicht.

Die erste Zahl (vor dem Doppelpunkt) in der Slicing-Operation verweist auf die Position, an der der Ausschnitt beginnt, und die zweite Zahl (nach dem Doppelpunkt) gibt an, an welcher Stelle der Ausschnitt endet. Wenn die erste Zahl nicht angegeben wird, beginnt Python am Anfang der Sequenz. Wenn die zweite Zahl ausgelassen wird, endet Python am Ende der Sequenz. Beachten Sie, dass der zurückgegebene Ausschnitt an der Startposition *beginnt* und unmittelbar vor der *End*-Position endet, d. h. die Startposition ist enthalten, die Endposition jedoch nicht.

Somit liefert `einkaufsliste[1:3]` einen Ausschnitt der Sequenz beginnend bei Position 1, enthält Position 2, endet jedoch vor Position 3. Daher wird ein *Ausschnitt* aus zwei Elementen zurückgegeben. Ebenso liefert `einkaufsliste[:]` eine Kopie der gesamten Sequenz.

Sie können Slicing auch mit negativen Positionen durchführen. Negative Zahlen werden für Positionen vom Ende der Sequenz aus verwendet. Beispielsweise liefert `einkaufsliste[:-1]` einen Ausschnitt der Sequenz, der das letzte Element der Sequenz ausschließt, jedoch alles andere enthält.

Sie können außerdem ein drittes Argument für den Ausschnitt angeben, nämlich die *Schrittweite* (*step*) für das Slicing (standardmäßig beträgt die Schrittweite 1):

```
>>> einkaufsliste = ['Apfel', 'Mango', 'Karotte', 'Banane']
>>> shoplist[::1]
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
['Apfel', 'Mango', 'Karotte', 'Banane']
>>> shoplist[::2]
['Apfel', 'Karotte']
>>> shoplist[::3]
['Apfel', 'Banane']
>>> shoplist[::-1]
['Banane', 'Karotte', 'Mango', 'Apfel']
```

Beachten Sie, dass wir bei einer Schrittweite von 2 die Elemente mit den Positionen 0, 2, ... erhalten. Wenn die Schrittweite 3 beträgt, erhalten wir die Elemente mit den Positionen 0, 3 usw.

Probieren Sie verschiedene Kombinationen solcher Slice-Spezifikationen interaktiv im Python-Interpreter aus, d. h. an der Eingabeaufforderung, damit Sie die Ergebnisse sofort sehen können. Das Großartige an Sequenzen ist, dass Sie auf Tupel, Listen und Zeichenketten auf dieselbe Weise zugreifen können!

11.6 Set

Sets sind *ungeordnete* Sammlungen einfacher Objekte. Sie werden verwendet, wenn die Existenz eines Objekts in einer Sammlung wichtiger ist als die Reihenfolge oder wie oft es vorkommt.

Mithilfe von Sets können Sie auf Mitgliedschaft testen, überprüfen, ob ein Set eine Teilmenge eines anderen Sets ist, die Schnittmenge zweier Sets bestimmen und vieles mehr.

```
>>> bri= set(['Brasilien', 'Russland', 'Indien'])
>>> 'Indien' in bri
True
>>> 'USA' in bri
False
>>> bric = bri.copy()
>>> bric.add('China')
>>> bric.issuperset(bri)
True
>>> bri.remove('russia')
>>> bri & bric # OR bri.intersection(bric)
{'brazil', 'india'}
```

Wie es funktioniert

Wenn Sie sich an die grundlegende Mengenlehre aus der Schule erinnern, dann erklärt sich dieses Beispiel weitgehend von selbst. Falls nicht, können Sie nach „set theory“ und „Venn diagram“ suchen, um unsere Verwendung von Sets in Python besser zu verstehen.

11.7 Referenzen

Wenn Sie ein Objekt erstellen und es einer Variablen zuweisen, dann *verweist* die Variable lediglich auf das Objekt und repräsentiert nicht das Objekt selbst! Das bedeutet, dass der Variablenname auf den Bereich im Speicher Ihres Computers zeigt, in dem das Objekt gespeichert ist. Dies wird als *Bindung* des Namens an das Objekt bezeichnet.

Im Allgemeinen müssen Sie sich darüber keine Sorgen machen, aber es gibt einen subtilen Effekt aufgrund von Referenzen, den Sie kennen sollten:

Beispiel ds_reference_de.py

Quellcode

```
print('Einfache Zuweisung')
obst = ['Apfel', 'Mango', 'Karotte', 'Banane']
# einkaufsliste ist nur ein anderer name für das obst objekt
früchte = obst

# Erstes Element löschen (Das erste Element in Python hat immer den Index 0)
del früchte[0]

print('Obst:', obst)
print('Früchte:', früchte)
# Beachte daß der Apfel in beiden listen nicht mehr existiert.
# Dies beweist daß beide Listen auf ein und dasselbe Objekt zeigen

print('Kopieren (full slice)')
# eine Kopie der Liste Obst erzeugen (full slice)
früchte = obst[:]
# (Nur von) früchte das erste Element entfernen)
del früchte[0]

print('Obst:', obst)
print('Früchte', früchte)
# Beachte den Unterschied zwischen beiden Listen
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
$ python3 ds_reference_de.py
Einfache Zuweisung
Obst: ['Mango', 'Karotte', 'Banane']
Früchte: ['Mango', 'Karotte', 'Banane']
Kopieren (full slice)
Obst: ['Mango', 'Karotte', 'Banane']
Früchte ['Karotte', 'Banane']
```

Wie es funktioniert

Der Großteil der Erklärung befindet sich in den Kommentaren.

Denken Sie daran: Wenn Sie eine Kopie einer Liste oder ähnlicher Sequenzen bzw. komplexer Objekte (nicht einfacher *Objekte* wie Ganzzahlen) erstellen möchten, dann müssen Sie die Slicing-Operation verwenden, um eine Kopie anzulegen. Wenn Sie lediglich den Variablennamen einem anderen Namen zuweisen, dann werden beide auf dasselbe Objekt "verweisen", was problematisch sein kann, wenn Sie nicht vorsichtig sind.

Hinweis für Perl-Programmierer

Denken Sie daran, dass eine Zuweisungsanweisung für Listen **keine** Kopie erstellt. Sie müssen die Slicing-Operation verwenden, um eine Kopie der Sequenz anzulegen.

11.8 Mehr über Zeichenketten

- (strings:)

Wir haben Zeichenketten bereits früher ausführlich behandelt. Was könnte es darüber hinaus noch zu wissen geben? Nun, wussten Sie, dass Zeichenketten ebenfalls Objekte sind und Methoden besitzen, die alles Mögliche tun – vom Überprüfen eines Teils einer Zeichenkette bis hin zum Entfernen von Leerzeichen? Tatsächlich haben Sie bereits eine String-Methode verwendet … die Methode `format`!

Die Zeichenketten, die Sie in Programmen verwenden, sind allesamt Objekte der Klasse `str`. Einige nützliche Methoden dieser Klasse werden im nächsten Beispiel demonstriert. Eine vollständige Liste solcher Methoden finden Sie unter `help(str)`.

Beispiel ds_str_methods_de.py

Quellcode

```
# Das ist ein  String-Objekt
name = 'Swaroop'

if name.startswith('Swa'):
    print('Ja, der String beginnt mit "Swa"')

if 'a' in name:
    print('Ja, der String enthält ein "a"')

if name.find('war') != -1:
    print('Ja, der String enthält den (sub)string "war"')

delimiter = '_*_'
länderliste = ['Brazil', 'Russia', 'India', 'China']
print(delimiter.join(länderliste))
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

```
$ python3 ds_str_methods_de.py
Ja, der String beginnt mit "Swa"
Ja, der String enthält ein "a"
Ja, der String enthält den (sub)string "war"
Brazil_*_Russia_*_India_*_China
```

Wie es funktioniert

Hier sehen wir viele String-Methoden in Aktion. Die Methode `startswith` wird verwendet, um herauszufinden, ob die Zeichenkette mit einer bestimmten Zeichenkette beginnt. Der Operator `in` wird verwendet, um zu überprüfen, ob eine bestimmte Zeichenkette Teil einer anderen Zeichenkette ist.

Die Methode `find` wird verwendet, um die Position einer gegebenen Teilzeichenkette innerhalb der Zeichenkette zu finden; `find` gibt `-1` zurück, wenn die Teilzeichenkette nicht gefunden wird. Die Klasse `str` besitzt außerdem eine praktische Methode `join`, die die Elemente einer Sequenz mithilfe der Zeichenkette als Trennzeichen zwischen den einzelnen Elementen verbindet und daraus eine größere Zeichenkette erzeugt.

11.9 Zusammenfassung

Wir haben die verschiedenen eingebauten Datenstrukturen von Python ausführlich untersucht. Diese Datenstrukturen sind unverzichtbar für das Schreiben von Programmen mit angemessener Größe.

Da wir nun viele der Grundlagen von Python behandelt haben, werden wir als Nächstes sehen, wie man ein reales Python-Programm entwirft und schreibt.

CHAPTER 12

Problemlösung

Wir haben verschiedene Teile der Python-Sprache untersucht und nun werden wir einen Blick darauf werfen, wie all diese Teile zusammenpassen, indem wir ein Programm entwerfen und schreiben, das etwas Nützliches tut. Die Idee besteht darin, zu lernen, wie man ein Python-Skript selbstständig schreibt.

12.1 Das Problem

Das Problem, das wir lösen wollen, lautet:

> Ich möchte ein Programm, das ein Backup all meiner wichtigen Dateien erstellt.

Obwohl dies ein einfaches Problem ist, gibt es nicht genügend Informationen, um mit der Lösung zu beginnen. Es ist ein wenig mehr *Analyse* erforderlich. Zum Beispiel: Wie geben wir an, *welche* Dateien gesichert werden sollen? *Wie* werden sie gespeichert? *Wohin* werden sie gespeichert?

Nachdem wir das Problem richtig analysiert haben, *entwerfen* (design) wir unser Programm. Wir erstellen eine Liste von Dingen, wie unser Programm funktionieren soll. In diesem Fall habe ich die folgende Liste erstellt, wie *ich* möchte, dass es funktioniert. Wenn Sie selbst das Design erstellen, kommen Sie möglicherweise zu einer anderen Analyse, da jede Person ihre eigene Art hat, Dinge zu tun – das ist völlig in Ordnung.

- Die zu sichernden Dateien und Verzeichnisse werden in einer Liste angegeben.
- Das Backup muss in einem Haupt-Backup-Verzeichnis gespeichert werden.
- Die Dateien werden in eine ZIP-Datei gesichert.
- Der Name des ZIP-Archivs ist das aktuelle Datum und die Uhrzeit.
- Wir verwenden den standardmäßigen `zip`-Befehl, der in jeder üblichen GNU/Linux- oder Unix-Distribution verfügbar ist. Beachten Sie, dass Sie jeden beliebigen Archivierungsbefehl verwenden können, solange er über eine Kommandozeilenschnittstelle verfügt.

Für Windows-Benutzer

Windows-Benutzer können den `zip`-Befehl über die GnuWin32-Projektseite installieren und `C:\Program Files\GnuWin32\bin` zu ihrer `PATH`-Umgebungsvariable hinzufügen, ähnlich wie wir es getan haben, um den python-Befehl selbst zu erkennen (siehe *Installation*).

12.2 Die Lösung

Da das Design unseres Programms nun ausreichend stabil ist, können wir den Code schreiben, der eine Implementierung unserer Lösung darstellt.

Beispiel backup_ver1_de.py

Quellcode

```
import os
import time

# 1. Die Dateien und Ordner zum sichern sind in dieser Liste:
# Beispiel für Windows:
# quell_ordner = ['"C:\\Meine Dokumente"']
# Beachte die doppelten Anführungszeichen im String ( '"C:\\My Documents"' )
# Weil der String (bzw der Dateiname) Leerzeichen enthält.
# Alternativ kann man auch einen raw string benutzen:
# [r'C:\Meine Dokumente'].

# Beispiel für Mac OS X and Linux:
quell_ordner = ['/home/horst/Documents/rudi']  # hier eigenen Pfad eingeben


# 2. Die Sicherheitskopie ("backup") muss in einem eigenen Ordner
# gespeichert werden
# Beispiel für Windows:
# ziel_ordner = 'E:\\Backup'
# Beispiel für Mac OS X und Linux:
ziel_ordner = '/home/horst/Backups'  # hier eigenen Pfad eingeben!
# Nicht vergessen: Ordnernamen (Path) an die eigenen Bedürfnisse anpassen

# 3. Die Dateien werden in ein zip-file gepackt.
# 4. Der Name vom zip-file ist das aktuelle Datum und die Uhrzeit
ziel = ziel_ordner + os.sep + \
         time.strftime('%Y%m%d%H%M%S') + '.zip'

# Erzeuge den Ziel-Ordner falls er noch nicht existiert
if not os.path.exists(ziel_ordner):
    os.mkdir(ziel_ordner)  # mdkir steht für 'make directory'

# 5. Das zip kommando benutzen um die Dateien ins zip-archiv zu kopieren
zip_command = 'zip -r {0} {1}'.format(ziel,' '.join(quell_ordner))

# Run the backup
print('Das Zip command lautet:')
print(zip_command)
```

```
39 print('Ausführung:')
40 if os.system(zip_command) == 0:
41     print('Backup erfolgreich in Datei:', ziel)
42 else:
43     print('Backup abgebrochen (Fehler)')
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
$ python backup_de.py
Das Zip command lautet:
zip -r /home/horst/Backups/20260512143235.zip /home/horst/Documents/rudi
Ausführung:
  adding: home/horst/Documents/rudi/ (stored 0%)
  adding: home/horst/Documents/rudi/gerburtstagseinladung4.pdf (deflated 1%)
  adding: home/horst/Documents/rudi/gerburtstagseinladung4.svg (deflated 25%)
  adding: home/horst/Documents/rudi/spielmusik_anmeldung_2.pdf (deflated 0%)
  adding: home/horst/Documents/rudi/Elternbrief neues Anmeldesystem _1_.pdf (deflated␣
↪8%)
  adding: home/horst/Documents/rudi/spielemusik_anmeldung_rudolf_JENS.pdf (deflated 0
↪%)
  adding: home/horst/Documents/rudi/spielmusik_anmeldung_1.pdf (deflated 0%)
Backup erfolgreich in Datei: /home/horst/Backups/20260512143235.zip
```

Nun befinden wir uns in der Testphase, in der wir prüfen, ob unser Programm korrekt funktioniert. Wenn es sich nicht wie erwartet verhält, müssen wir unser Programm debuggen, d. h. die Fehler (*bugs*) aus dem Programm entfernen.

Wenn das obige Programm bei Ihnen nicht funktioniert, kopieren Sie die Zeile, die nach der Ausgabe des `Zip command ist` erscheint, fügen Sie sie in die Shell (unter GNU/Linux und macOS) bzw. in cmd (unter Windows) ein, sehen Sie nach, was der Fehler ist, und versuchen Sie, ihn zu beheben. Prüfen Sie auch im Handbuch des `zip`-Befehls, was falsch sein könnte. Wenn dieser Befehl funktioniert, könnte das Problem im Python-Programm selbst liegen. Prüfen Sie dann, ob es exakt mit dem oben aufgeführten Programm übereinstimmt.

Wie es funktioniert:

Sie werden bemerken, wie wir unser Design Schritt für Schritt in Code umgesetzt haben.

Wir verwenden die Module `os` und `time`, indem wir sie zuerst importieren. Dann geben wir die zu sichernden Dateien und Verzeichnisse in der Liste `quell_ordner` an. Das Zielverzeichnis, in dem wir alle Backup-Dateien speichern, wird in der Variablen `ziel_ordner` angegeben. Der Name des ZIP-Archivs, das wir erstellen wollen, ist das aktuelle Datum und die Uhrzeit, die wir mit der Funktion `time.strftime()` erzeugen. Es erhält auch die Endung `.zip` und wird im Verzeichnis `ziel_ordner` gespeichert.

Beachten Sie die Verwendung der Variablen `os.sep` – sie liefert das für Ihr Betriebssystem passende Verzeichnistrennzeichen, d. h. `'/'` für GNU/Linux, Unix und macOS bzw. `'\\'` für Windows. Die Verwendung von `os.sep` anstelle dieser Zeichen direkt macht unser Programm portabel und funktionstüchtig auf allen diesen Systemen.

Die Funktion `time.strftime()` verwendet ein Format wie das, das wir im obigen Programm benutzt haben. Die Angabe `%Y` wird durch das Jahr mit Jahrhundert ersetzt. Die Angabe `%m` wird durch den Monat als Dezimalzahl zwischen 01 und 12 ersetzt usw. Die vollständige Liste solcher Angaben finden Sie im Python Reference Manual.

Wir erstellen den Namen der Ziel-ZIP-Datei mit dem Addition-Operator, der Zeichenketten *konkateniert* (concatenates_), d. h. sie miteinander verbindet und eine neue Zeichenkette zurückgibt. Dann erstellen wir eine Zeichenkette `zip_command`, die den Befehl enthält, den wir ausführen werden. Sie können prüfen, ob dieser Befehl funktioniert, indem Sie ihn in der Shell (GNU/Linux-Terminal oder DOS-Eingabeaufforderung) ausführen.

Der von uns verwendete `zip`-Befehl verfügt über einige Optionen, darunter die Option `-r`. Die Option `-r` gibt an, dass der zip-Befehl *rekursiv* für Verzeichnisse arbeiten soll, d. h. er soll alle Unterverzeichnisse und Dateien einbeziehen. Auf die Optionen folgen der Name des zu erstellenden ZIP-Archivs sowie die Liste der zu sichernden Dateien und Verzeichnisse. Wir wandeln die Liste `quell_ordner` mit der Methode `join` von Strings in eine Zeichenkette um, deren Verwendung wir bereits kennengelernt haben.

Dann führen wir den Befehl schließlich mit der Funktion `os.system` aus, die den Befehl so ausführt, als wäre er vom System aufgerufen worden, d. h. in der Shell – sie gibt 0 zurück, wenn der Befehl erfolgreich war, andernfalls eine Fehlernummer.

Abhängig vom Ergebnis des Befehls geben wir die entsprechende Meldung aus, ob das Backup fehlgeschlagen oder erfolgreich war.

Das war's – wir haben ein Skript erstellt, um eine Sicherung unserer wichtigen Dateien zu erstellen!

Hinweis für Windows-Benutzer

Anstelle doppelter Backslash-Escape-Sequenzen können Sie auch Raw-Strings verwenden. Beispiel: `'C:\\Documents'` oder `r'C:\Documents'`. Verwenden Sie jedoch nicht `'C:\Documents'`, weil Sie sonst eine unbekannte Escape-Sequenz `\D` verwenden.

Nun, da wir ein funktionierendes Backup-Skript haben, können wir es verwenden, wann immer wir ein Backup der Dateien anlegen möchten. Dies nennt man die Betriebsphase oder *Deployment*-Phase der Software.

Das obige Programm funktioniert, aber (normalerweise) funktionieren Erstprogramme nicht genau so, wie Sie es erwarten. Beispielsweise kann es Probleme geben, wenn Sie das Programm nicht korrekt entworfen haben oder wenn Sie einen Tippfehler im Code gemacht haben. In diesem Fall müssen Sie zum Design zurückkehren oder das Programm debuggen.

12.3 Zweite Version

Die erste Version unseres Skripts funktioniert. Dennoch können wir einige Verfeinerungen daran vornehmen, damit es im täglichen Einsatz besser funktioniert. Dies nennt man die Wartungsphase der Software.

Eine der sinnvollen Verfeinerungen ist ein besserer Mechanismus zur Dateibenennung – die Verwendung der Uhrzeit als Dateiname innerhalb eines Verzeichnisses, dessen Name das aktuelle Datum ist, innerhalb des Haupt-Backup-Verzeichnisses. Der erste Vorteil besteht darin, dass Ihre Backups hierarchisch gespeichert werden und somit viel einfacher zu verwalten sind. Der zweite Vorteil ist, dass die Dateinamen viel kürzer sind. Der dritte Vorteil besteht darin, dass separate Verzeichnisse Ihnen helfen können zu prüfen, ob Sie für jeden Tag ein Backup erstellt haben, da das Verzeichnis nur erstellt wird, wenn Sie für diesen Tag ein Backup gemacht haben.

Beispiel backup_ver2_de.py

Quellcode

```
import os
import time

# 1. Die Dateien und Ordner zum sichern sind in dieser Liste:
# Beispiel für Windows:
# quell_ordner = ['"C:\\Meine Dokumente"', 'C:\\Code']
# Beachte die doppelten Anführungszeichen im String ( '"C:\\Meine Dokumente"' )
# Weil der String (bzw der Dateiname) Leerzeichen enthält.
# Alternativ kann man auch einen raw string benutzen:
```

```
# [r'C:\Meine Dokumente'].

# Beispiel für Mac OS X and Linux:
quell_ordner = ['/home/horst/Documents/rudi'] # Pfad anpassen!


# 2. Die Sicherheitskopie ("backup") muss in einem eigenen Ordner
# gespeichert werden
# Beispiel für Windows:
# ziel_ordner = 'E:\\Backup'
# Beispiel für Mac OS X und Linux:
ziel_ordner = '/home/horst/Backups'
# Nicht vergessen: Ordnernamen (Path) an die eigenen Bedürfnisse anpassen

# 3. Die Dateien werden in ein zip-file gepackt.
# 4. Der Name vom zip-file ist das aktuelle Datum und die Uhrzeit
heute =  time.strftime('%Y%m%d') # year, month, day
jetzt =  time.strftime('%H%M%S') # hour, minute, second
ziel = ziel_ordner + os.sep + heute + os.sep + jetzt + ".zip"


# Erzeuge den Ziel-Ordner (Backups) falls er noch nicht existiert
if not os.path.exists(ziel_ordner):
    os.mkdir(ziel_ordner)  # mdkir steht für 'make directory'
# Erzeuge (innerhalb von Backups) einen Ordner (YMT) falls er noch nicht existiert
if not os.path.exists(ziel_ordner + os.sep + heute):
    os.mkdir(ziel_ordner + os.sep + heute )

# 5. Das zip kommando benutzen um die Dateien ins zip-archiv zu kopieren
zip_command = 'zip -r {0} {1}'.format(ziel,' '.join(quell_ordner))

# Run the backup
print('Das Zip command lautet:')
print(zip_command)
print('Ausführung:')
if os.system(zip_command) == 0:
    print('Backup erfolgreich in Datei:', ziel)
else:
    print('Backup abgebrochen (Fehler)')
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
$ python backup_ver2_de.py
Successfully created directory /Users/swa/backup/20140329
Das Zip command lautet::
zip -r /Users/swa/backup/20140329/073201.zip /Users/swa/notes
Ausführung:
  adding: Users/swa/notes/ (stored 0%)
  adding: Users/swa/notes/blah1.txt (stored 0%)
  adding: Users/swa/notes/blah2.txt (stored 0%)
  adding: Users/swa/notes/blah3.txt (stored 0%)
Backup erfolgreich in Datei: /Users/swa/backup/20140329/073201.zip
```

Wie es funktioniert:

Der größte Teil des Programms bleibt gleich. Die Veränderungen bestehen darin, dass wir mit der Funktion `os.path.exists` prüfen, ob ein Verzeichnis mit dem aktuellen Tagesdatum als Namen im Haupt-Backup-Verzeichnis existiert. Wenn es nicht existiert, erstellen wir es mit der Funktion `os.mkdir`.

12.4 Dritte Version

Die zweite Version funktioniert gut, wenn ich viele Backups mache, aber wenn es sehr viele Backups gibt, fällt es mir schwer zu unterscheiden, wofür die Backups waren! Wenn ich zum Beispiel größere Änderungen an einem Programm oder einer Präsentation vorgenommen habe, möchte ich diese Änderungen mit dem Namen des ZIP-Archivs verknüpfen. Dies kann leicht erreicht werden, indem man dem Namen des ZIP-Archivs einen vom Benutzer angegebenen Kommentar hinzufügt.

Warnung

Das folgende Programm funktioniert nicht, also seien Sie nicht beunruhigt – bitte folgen Sie weiter, da hier eine wichtige Lektion enthalten ist.

Beispiel backup_ver3_de.py

Quellcode

```
import os
import time

# 1. Die Dateien und Ordner zum sichern sind in dieser Liste:
# Beispiel für Windows:
# quell_ordner = ['"C:\\Meine Dokumente"', 'C:\\Code']
# Beachte die doppelten Anführungszeichen im String ( '"C:\\Meine Dokumente"' )
# Weil der String (bzw der Dateiname) Leerzeichen enthält.
# Alternativ kann man auch einen raw string benutzen:
# [r'C:\Meine Dokumente'].

# Beispiel für Mac OS X and Linux:
quell_ordner = ['/home/horst/Documents/rudi'] # Pfad anpassen!


# 2. Die Sicherheitskopie ("backup") muss in einem eigenen Ordner
# gespeichert werden
# Beispiel für Windows:
# ziel_ordner = 'E:\\Backup'
# Beispiel für Mac OS X und Linux:
ziel_ordner = '/home/horst/Backups'
# Nicht vergessen: Ordnernamen (Path) an die eigenen Bedürfnisse anpassen

# 3. Die Dateien werden in ein zip-file gepackt.
# 4. Der Name vom zip-file ist das aktuelle Datum und die Uhrzeit
```

```
heute =  time.strftime('%Y%m%d') # year, month, day
jetzt =  time.strftime('%H%M%S') # hour, minute, second
# Benutzer nach Kommentar fragen
kommentar = input("Gib einen Kommentar ein (und drücke die ENTER Taste) >>>")
# wurde wirklich etwas eingegeben ?
if len(kommentar) == 0:                    # es wurde nur auf die Enter Taste␣
↪gedrückt
    ziel = ziel_ordner + os.sep + heute + os.sep + jetzt + ".zip"
else:                                      # es wurde ein Kommentar eingegeben
    kommentar = kommentar.replace(" ", "_") # Leerzeichen durch Unterstriche ersetzten
    ziel = ziel_ordner + os.sep + heute + os.sep + jetzt + "_" +
           kommentar + ".zip"

# Erzeuge den Ziel-Ordner (Backups) falls er noch nicht existiert
if not os.path.exists(ziel_ordner):
    os.mkdir(ziel_ordner)  # mdkir steht für 'make directory'
# Erzeuge (innerhalb von Backups) einen Ordner (YMT) falls er noch nicht existiert
if not os.path.exists(ziel_ordner + os.sep + heute):
    os.mkdir(ziel_ordner + os.sep + heute )

# 5. Das zip kommando benutzen um die Dateien ins zip-archiv zu kopieren
zip_command = 'zip -r {0} {1}'.format(ziel,' '.join(quell_ordner))

# Run the backup
print('Das Zip command lautet:')
print(zip_command)
print('Ausführung:')
if os.system(zip_command) == 0:
    print('Backup erfolgreich in Datei:', ziel)
else:
    print('Backup abgebrochen (Fehler)')
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
$ python3 backup_ver3_de.py
File "/home/horst/code/byte-of-python_deutsch_horst/programs/backup_ver3_de.py", line␣
↪33
    ziel = ziel_ordner + os.sep + heute + os.sep + jetzt + "_" +
                                                             ^
SyntaxError: invalid syntax
```

Warum es (nicht) funktioniert:

Dieses Programm funktioniert nicht! Python meldet einen Syntaxfehler (*syntax error*), was bedeutet, dass das Skript nicht der Struktur entspricht, die Python erwartet. Wenn wir die von Python ausgegebene Fehlermeldung betrachten, teilt sie uns auch die Stelle mit, an der der Fehler erkannt wurde. Daher beginnen wir an dieser Zeile mit dem Debugging unseres Programms.

Bei genauer Beobachtung stellen wir fest, dass eine logische Zeile in zwei physische Zeilen aufgeteilt wurde, wir aber nicht angegeben haben, dass diese beiden physischen Zeilen zusammengehören. Python hat im Grunde den Additionsoperator (+) ohne Operanden in dieser logischen Zeile gefunden und weiß daher nicht, wie es fortfahren soll. Denken Sie daran, dass wir angeben können, dass die logische Zeile in der nächsten physischen Zeile weitergeht,

indem wir am Ende der physischen Zeile einen Backslash verwenden. Also nehmen wir diese Korrektur an unserem Programm vor. Diese Korrektur, die wir durchführen, wenn wir Fehler finden, nennt man Bugfixing.

12.5 Vierte Version

Beispiel backup_ver4_de.py

Quellcode

```
import os
import time

# 1. Die Dateien und Ordner zum sichern sind in dieser Liste:
# Beispiel für Windows:
# quell_ordner = ['"C:\\Meine Dokumente"', 'C:\\Code']
# Beachte die doppelten Anführungszeichen im String ( '"C:\\Meine Dokumente"' )
# Weil der String (bzw der Dateiname) Leerzeichen enthält.
# Alternativ kann man auch einen raw string benutzen:
# [r'C:\Meine Dokumente'].

# Beispiel für Mac OS X and Linux:
quell_ordner = ['/home/horst/Documents/rudi'] # Pfad anpassen!


# 2. Die Sicherheitskopie ("backup") muss in einem eigenen Ordner
# gespeichert werden
# Beispiel für Windows:
# ziel_ordner = 'E:\\Backup'
# Beispiel für Mac OS X und Linux:
ziel_ordner = '/home/horst/Backups'
# Nicht vergessen: Ordnernamen (Path) an die eigenen Bedürfnisse anpassen

# 3. Die Dateien werden in ein zip-file gepackt.
# 4. Der Name vom zip-file ist das aktuelle Datum und die Uhrzeit
heute =  time.strftime('%Y%m%d') # year, month, day
jetzt =  time.strftime('%H%M%S') # hour, minute, second
# Benutzer nach Kommentar fragen
kommentar = input("Gib einen Kommentar ein (und drücke die ENTER Taste) >>>")
# wurde wirklich etwas eingegeben ?
if len(kommentar) == 0:                    # es wurde nur auf die Enter Taste␣
↪gedrückt
    ziel = ziel_ordner + os.sep + heute + os.sep + jetzt + ".zip"
else:                                      # es wurde ein Kommentar eingegeben
    kommentar = kommentar.replace(" ", "_") # Leerzeichen durch Unterstriche ersetzten
    ziel = ziel_ordner + os.sep + heute + os.sep + jetzt + "_"  + \
           kommentar + ".zip"

# Erzeuge den Ziel-Ordner (Backups) falls er noch nicht existiert
if not os.path.exists(ziel_ordner):
    os.mkdir(ziel_ordner)  # mdkir steht für 'make directory'
# Erzeuge (innerhalb von Backups) einen Ordner (YMT) falls er noch nicht existiert
if not os.path.exists(ziel_ordner + os.sep + heute):
    os.mkdir(ziel_ordner + os.sep + heute )
```

```
44
45  # 5. Das zip kommando benutzen um die Dateien ins zip-archiv zu kopieren
46  zip_command = 'zip -r {0} {1}'.format(ziel,' '.join(quell_ordner))
47
48  # Run the backup
49  print('Das Zip command lautet:')
50  print(zip_command)
51  print('Ausführung:')
52  if os.system(zip_command) == 0:
53      print('Backup erfolgreich in Datei:', ziel)
54  else:
55      print('Backup abgebrochen (Fehler)')
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
$ python3 backup_ver4_de.py
Gib einen Kommentar ein (und drücke die ENTER Taste) >>>neue Beispieldatei
Das Zip command lautet:
zip -r /home/horst/Backups/20260512/144752_neue_Beispieldatei.zip /home/horst/
↪Documents/rudi
Ausführung:
  adding: home/horst/Documents/rudi/ (stored 0%)
  adding: home/horst/Documents/rudi/gerburtstagseinladung4.pdf (deflated 1%)
  adding: home/horst/Documents/rudi/gerburtstagseinladung4.svg (deflated 25%)
  adding: home/horst/Documents/rudi/spielmusik_anmeldung_2.pdf (deflated 0%)
  adding: home/horst/Documents/rudi/Elternbrief neues Anmeldesystem _1_.pdf (deflated␣
↪8%)
  adding: home/horst/Documents/rudi/spielemusik_anmeldung_rudolf_JENS.pdf (deflated 0
↪%)
  adding: home/horst/Documents/rudi/spielmusik_anmeldung_1.pdf (deflated 0%)
Backup erfolgreich in Datei: /home/horst/Backups/20260512/144752_neue_Beispieldatei.
↪zip
```

Wie es funktioniert:

Dieses Programm funktioniert jetzt! Gehen wir die tatsächlichen Erweiterungen durch, die wir in Version 3 vorgenommen hatten. Wir holen die Kommentare des Benutzers mit der Funktion `input` ein und prüfen dann, ob der Benutzer tatsächlich etwas eingegeben hat, indem wir die Länge der Eingabe mit der Funktion `len` bestimmen. Wenn der Benutzer lediglich die Eingabetaste gedrückt hat, ohne etwas einzugeben (vielleicht war es nur ein Routine-Backup oder es wurden keine besonderen Änderungen vorgenommen), fahren wir wie zuvor fort.

Wenn jedoch ein Kommentar eingegeben wurde, wird dieser dem Namen des ZIP-Archivs unmittelbar vor der `.zip`-Endung hinzugefügt. Beachten Sie, dass wir Leerzeichen im Kommentar durch Unterstriche ersetzen – das liegt daran, dass das Verwalten von Dateinamen ohne Leerzeichen wesentlich einfacher ist.

12.6 Weitere Verfeinerungen

Die vierte Version ist für die meisten Benutzer ein zufriedenstellend funktionierendes Skript, aber es gibt immer Raum für Verbesserungen. Sie könnten zum Beispiel für den zip-Befehl eine Ausführlichkeitsstufe hinzufügen, indem Sie die Option `-v` (für *verbose*, geschwätzig) festlegen, um Ihr Programm gesprächiger zu machen, oder die Option `-q` (für *quiet*, ruhig), um es ruhig zu machen.

Eine weitere mögliche Verbesserung wäre, dem Skript das Übergeben zusätzlicher Dateien und Verzeichnisse über die Befehlszeile zu erlauben. Wir können diese Namen aus der Liste `sys.argv` erhalten und sie unserer source-Liste mit der Methode `extend`, die von der Klasse `list` bereitgestellt wird, hinzufügen.

Die wichtigste Verfeinerung wäre es, nicht den Weg über `os.system` zum Erstellen von Archiven zu verwenden, sondern stattdessen die eingebauten Module zipfile oder tarfile zum Erstellen dieser Archive zu verwenden. Sie sind Teil der Standardbibliothek und bereits ohne externe Abhängigkeiten verfügbar.

Ich habe jedoch in den obigen Beispielen die Verwendung von `os.system` zum Erstellen eines Backups ausschließlich zu didaktischen Zwecken gewählt, damit das Beispiel einfach genug bleibt, um von jedem verstanden zu werden, aber real genug, um nützlich zu sein.

Können Sie versuchen, die fünfte Version zu schreiben, die das Modul `zipfile` anstelle des Aufrufs `os.system` verwendet?

12.7 Der Softwareentwicklungsprozess

Wir haben nun die verschiedenen Phasen des Schreibens einer Software durchlaufen. Diese Phasen lassen sich wie folgt zusammenfassen:

1. Was (Analyse)
2. Wie (Design)
3. Tun (Implementierung)
4. Testen (Test und Debugging)
5. Verwenden (Betrieb oder Deployment)
6. Pflegen (Verfeinerung)

Eine empfohlene Methode beim Schreiben von Programmen ist die Vorgehensweise, die wir beim Erstellen des Backup-Skripts verwendet haben: Führen Sie Analyse und Design durch. Beginnen Sie mit einer einfachen Version der Implementierung. Testen und debuggen Sie sie. Verwenden Sie sie, um sicherzustellen, dass sie wie erwartet funktioniert. Fügen Sie dann die gewünschten Funktionen hinzu und wiederholen Sie den Tun-Testen-Verwenden-Zyklus so oft wie nötig.

Denken Sie daran:

> Software wird gepflegt und weiterentwickelt, nicht „gebaut“.
>
> – *Bill de hÓra*

12.8 Zusammenfassung

Wir haben gesehen, wie wir unsere eigenen Python-Programme/Skripte erstellen können und welche verschiedenen Schritte beim Schreiben solcher Programme beteiligt sind. Es könnte für Sie hilfreich sein, ein eigenes Programm zu entwickeln, genau wie wir es in diesem Kapitel getan haben, damit Sie sowohl mit Python als auch mit Problemlösung vertraut werden.

Als Nächstes werden wir objektorientierte Programmierung behandeln.

CHAPTER 13

Objektorientierte Programmierung

( **oop**: **o**bject **o**riented **p**rogramming)

In all den Programmen, die wir bisher geschrieben haben, haben wir unser Programm um Funktionen herum entworfen, d. h. Blöcke von Anweisungen, die Daten verarbeiten. Dies wird die *prozedur-orientierte* Art des Programmierens genannt. Es gibt eine andere Art, Ihr Programm zu organisieren, nämlich Daten und Funktionalität zu kombinieren und sie in etwas einzupacken, das ein Objekt genannt wird. Dies wird das *objektorientierte* Programmierparadigma genannt. Die meiste Zeit können Sie prozedurale Programmierung verwenden, aber beim Schreiben großer Programme oder wenn Sie ein Problem haben, das sich besser für diese Methode eignet, können Sie objektorientierte Programmiertechniken einsetzen.

Klassen und Objekte sind die beiden Hauptaspekte der objektorientierten Programmierung. Eine **Klasse** (class) erstellt einen neuen Typ, während **Objekte Instanzen** der Klasse sind. Eine Analogie ist, dass Sie Variablen des Typs `int` haben können, was bedeutet, dass Variablen, die ganze Zahlen speichern, Variablen sind, die Instanzen (Objekte) von `int` Klasse sind.

Hinweis für Programmierer statischer Sprachen

Beachten Sie, dass selbst Ganzzahlen als Objekte (der Klasse int) behandelt werden. Dies unterscheidet sich von C++ und Java (vor Version 1.5), wo Ganzzahlen primitive, native Typen sind.

Siehe `help(int)` für weitere Details zur Klasse.

C#- und Java-1.5-Programmierer werden dies mit dem Konzept des *Boxing und Unboxing* vergleichen können.

Objekte können Daten mithilfe gewöhnlicher Variablen speichern, die zum Objekt *gehören*. Variablen, die zu einem Objekt oder einer Klasse gehören, werden *Felder* genannt. Objekte können auch Funktionalität besitzen, indem Funktionen verwendet werden, die zu einer Klasse gehören. Solche Funktionen heißen **Methoden** der Klasse. Diese Terminologie ist wichtig, weil sie uns hilft, zwischen Funktionen und Variablen zu unterscheiden, die unabhängig sind, und solchen, die zu einer Klasse oder einem Objekt gehören. Gemeinsam können die Felder und Methoden als die **Attribute** dieser Klasse bezeichnet werden.

Felder gibt es in zwei Arten – sie können zu jeder einzelenen Instanz der Klasse gehören oder sie können zur ganzen Klasse (allen Instanzen gemeinsam) gehören. Sie heißen entsprechend Instanzvariablen und Klassenvariablen.

Eine Klasse wird mit dem Schlüsselwort `class` erzeugt. Die Felder und Methoden der Klasse werden in einem eingerückten Block aufgelistet.

13.1 Das `self`

Klassenmethoden unterscheiden sich nur in einem Punkt von gewöhnlichen Funktionen – sie müssen einen zusätzlichen ersten Namen haben, der am Anfang der Parameterliste hinzugefügt werden muss, aber Sie geben beim Aufruf der Methode keinen Wert für diesen Parameter an; Python stellt ihn bereit. Diese spezielle Variable bezieht sich auf das Objekt selbst, und nach Konvention wird ihr der Name `self` gegeben.

Obwohl Sie diesem Parameter einen beliebigen Namen geben können, wird dringend empfohlen, den Namen `self` zu verwenden – jeder andere Name wird eindeutig missbilligt. Es gibt viele Vorteile bei der Verwendung eines standardisierten Namens – jeder Leser Ihres Programms erkennt ihn sofort, und sogar spezialisierte IDEs (Integrated Development Environments) können Ihnen helfen, wenn Sie `self` verwenden.

Hinweis für C++/Java/C#-Programmierer

Das `self` in Python entspricht dem `this`-Pointer in C++ sowie der `this`-Referenz in Java und C#.

Sie fragen sich sicher, wie Python den Wert für `self` bereitstellt und warum Sie keinen Wert dafür angeben müssen. Ein Beispiel wird dies verdeutlichen. Angenommen, Sie haben eine Klasse namens `MyClass` und eine Instanz dieser Klasse namens `myobject`. Wenn Sie eine Methode dieses Objekts wie `myobject.method(arg1, arg2)` aufrufen, wird dies von Python automatisch in `MyClass.method(myobject, arg1, arg2)` umgewandelt – das ist alles, worum es beim speziellen `self` geht.

Das bedeutet auch, dass wenn Sie eine Methode haben, die keine Argumente benötigt, Sie dennoch ein Argument haben müssen – das `self`.

13.2 Klassen

( classes)

Die einfachstmögliche Klasse wird im folgenden Beispiel gezeigt:

Beispiel oop_simplestclass_de.py

Quellcode

```
1 class Person:
2     pass  # Ein leerer Block
3
4 p = Person()
5 print(p)
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
$ python3 oop_simplestclass_de.py
<__main__.Person object at 0x758768cf4c20>
```

Wie es funktioniert:

Wir erstellen eine neue Klasse mit der `class`-Anweisung und dem Namen der Klasse. Darauf folgt ein eingerückter Block von Anweisungen, die den Rumpf (body) der Klasse bilden. In diesem Fall haben wir einen leeren Block, der mit der `pass`-Anweisung gekennzeichnet ist.

Als Nächstes erstellen wir eine Instanz dieser Klasse, indem wir den Namen der Klasse gefolgt von einem Klammerpaar verwenden. (Wir werden später mehr über Instanziierung lernen.) Zu unserer Überprüfung bestätigen wir den Typ der Variable, indem wir sie einfach ausdrucken. Es sagt uns, dass wir eine Instanz der Klasse `Person` im Modul `__main__` haben.

Beachten Sie, dass auch die Adresse des Computerspeichers ausgegeben wird, an der Ihr Objekt gespeichert ist. Die Adresse wird auf Ihrem Computer einen anderen Wert haben, da Python das Objekt dort speichert, wo es freien Platz findet.

13.3 Methoden

( methods)

Wir haben bereits besprochen, dass Klassen/Objekte Methoden haben können, genau wie Funktionen, außer dass wir eine zusätzliche Variable self haben. Jetzt werden wir ein Beispiel sehen:

Beispiel oop_method_de.py

Quellcode

```
class Person:
    def sag_hallo(self):
        print("Hallo, wie geht's?")

p = Person()
p.sag_hallo()
# Die vorigen beiden Zeilen könnte man auch schreiben als:
# Person().sag_hallo()
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
$ python3 oop_method_de.py
Hallo, wie geht's?
```

Wie es funktioniert:

Hier sehen wir `self` in Aktion. Beachten Sie, dass die Methode `sag_hallo` keine Parameter übernimmt, aber dennoch `self` in der Funktionsdefinition hat.

13.4 Die Methode `__init__`

Es gibt viele Methodennamen, die in Python-Klassen eine besondere Bedeutung haben. Wir werden nun die Bedeutung der Methode `__init__` sehen.

Die `__init__`-Methode wird ausgeführt, sobald ein Objekt einer Klasse instanziiert (d. h. erzeugt) wird. Die Methode ist nützlich, um jede Initialisierung vorzunehmen (d. h. Anfangswerte an Ihr Objekt zu übergeben), die Sie mit Ihrem

Objekt durchführen wollen. Beachten Sie die doppelten Unterstriche sowohl am Anfang als auch am Ende des Namens.

Beispiel oop_init_de.py

Quellcode

```
1  class Person:
2      def __init__(self, name):
3          self.name = name
4
5      def sag_hallo(self):
6          print('Hallo, ich heiße', self.name)
7
8  p = Person('Swaroop')
9  p.sag_hallo()
10 # Man könnte auch beide vorigen Zeilen in einer Zeile schreiben:
11 # Person('Swaroop').sag_hallo()
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
$ python3 oop_init_de.py
Hallo, ich heiße Swaroop
```

Wie es funktioniert:

Hier definieren wir die `__init__`-Methode so, dass sie einen Parameter namens `name` übernimmt (zusammen mit dem üblichen `self`). Wir erstellen einfach ein neues Feld, ebenfalls `name` genannt. Beachten Sie, dass dies zwei unterschiedliche Variablen sind, auch wenn sie beide `name` heißen. Es gibt kein Problem, weil die Punktnotation `self.name` bedeutet, dass es etwas namens `name` gibt, das Teil des Objekts `self` ist, und das andere `name` ist eine lokale Variable. Da wir explizit angeben, auf welchen Namen wir uns beziehen, gibt es keine Verwechslung.

Beim Erzeugen der neuen Instanz `p` der Klasse `Person` tun wir dies, indem wir den Klassennamen gefolgt von den Argumenten in Klammern verwenden: `p = Person('Swaroop')`.

Wir rufen die `__init__`-Methode nicht explizit auf. Das ist die besondere Bedeutung dieser Methode.

Jetzt können wir das Feld `self.name` in unseren Methoden verwenden, was in der Methode `sag_hallo` demonstriert wird.

13.5 Klassen- und Objektvariablen

- (class variables: *Klassenvariablen*)
- (object variables: *Objektvariablen*)
- (name spaces: *Namensräume*)
- (fields: *Felder*)

Wir haben bereits den funktionalen Teil von Klassen und Objekten besprochen (d. h. Methoden), nun wollen wir etwas über den Datenteil lernen. Der Datenteil, d. h. Felder, sind nichts anderes als gewöhnliche Variablen, die an

die **Namensräume** der Klassen und Objekte gebunden sind. Das bedeutet, dass diese Namen nur im Kontext dieser Klassen und Objekte gültig sind. Daher nennt man sie **Namensräume**.

Es gibt zwei Arten von *Feldern* – Klassenvariablen und Objektvariablen, je nachdem, ob die Klasse oder das Objekt die Variablen besitzt.

Klassenvariablen gehören allen gemeinsam – sie können von allen Instanzen dieser Klasse verwendet werden. Es gibt nur eine Kopie der Klassenvariable, und wenn ein Objekt eine Klassenvariable ändert, wird diese Änderung von allen anderen Instanzen gesehen.

Objektvariablen gehören zu jedem einzelnen Objekt/Instanz der Klasse. In diesem Fall hat jedes Objekt seine eigene Kopie des Feldes, d. h. sie werden nicht geteilt und stehen nicht in Beziehung zu einem Feld gleichen Namens in einer anderen Instanz. Ein Beispiel macht dies leichter verständlich:

Beispiel oop_objvar_de.py

Quellcode

```
class Robot:
    """Repräsentiert einen Roboter mit einem Namen"""

    # Eine class variable, sie zählt die Anzahl der Roboter
    population = 0

    def __init__(self, name):
        """Intitialisiere Daten"""
        self.name = name
        print("(Initializing {})".format(self.name))

        # Der "frisch gebackene" Robotor erhöht die Roboter-Population
        Robot.population += 1

    def stirb(self):
        """Ich sterbe."""
        print("{} ist zerstört!".format(self.name))

        Robot.population -= 1

        if Robot.population == 0:
            print("{} war der letzte seiner Art.".format(self.name))
        else:
            print("Es sind noch {} Roboter übrig.".format(
                Robot.population))

    def sag_hallo(self):
        """
        Roboter können grüßen.
        Wirklich!
        """
        print("Grüße, ich werde {} genannt.".format(self.name))

    @classmethod
    def wieviele_gibt_es(cls):
        """druckt die Anzahl aller Roboter aus."""
        print("Derzeit gibt es {} Roboter.".format(cls.population))
```

```
38
39
40 droid1 = Robot("R2-D2")
41 droid1.sag_hallo()
42 Robot.wieviele_gibt_es()
43
44 droid2 = Robot("C-3PO")
45 droid2.sag_hallo()
46 Robot.wieviele_gibt_es()
47
48 print("\nHier könnten die Roboter arbeiten\n")
49
50 print("Arbiet ist fertig, Roboter werden zerstört....")
51 droid1.stirb()
52 droid2.stirb()
53
54 Robot.wieviele_gibt_es()
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
$ python3 oop_objvar_de.py
(Initializing R2-D2)
Grüße, ich werde R2-D2 genannt.
Derzeit gibt es 1 Roboter.
(Initializing C-3PO)
Grüße, ich werde C-3PO genannt.
Derzeit gibt es 2 Roboter.

Hier könnten die Roboter arbeiten

Arbiet ist fertig, Roboter werden zerstört....
R2-D2 ist zerstört!
Es sind noch 1 Roboter übrig.
C-3PO ist zerstört!
C-3PO war der letzte seiner Art.
Derzeit gibt es 0 Roboter.
```

Wie es funktioniert:

Dies ist ein langes Beispiel, hilft aber dabei, die Natur von Klassen- und Objektvariablen zu verdeutlichen. Hier gehört `population` zur Klasse `Robot` und ist daher eine Klassenvariable. Die Variable `name` gehört zum Objekt (sie wird mit `self` zugewiesen) und ist daher eine Objektvariable.

Daher verweisen wir auf die Klassenvariable `population` mit `Robot.population` und nicht mit `self.population`. Wir verweisen auf die Objektvariable `name` mit der Notation `self.name` in den Methoden dieses Objekts. Merken Sie sich diesen einfachen Unterschied zwischen Klassen- und Objektvariablen. Beachten Sie auch, dass eine Objektvariable mit demselben Namen wie eine Klassenvariable die Klassenvariable verdeckt!

Anstelle von `Robot.population` könnten wir auch `self.__class__.population` verwenden, weil jedes Objekt über das Attribut `self.__class__` auf seine Klasse verweist.

Die Methode `wieviele_gibt_es` gehört tatsächlich zur Klasse und nicht zum Objekt. Das bedeutet, wir können sie entweder als classmethod oder als staticmethod definieren, abhängig davon, ob wir wissen müssen, zu welcher Klasse

wir gehören. Da wir auf eine Klassenvariable verweisen, verwenden wir`classmethod`.

Wir haben die Methode `wieviele_gibt_es` als Klassenmethode mit einem Dekorator markiert.

Dekoratoren kann man sich als eine Abkürzung zur Anwendung einer Wrapper-Funktion vorstellen (d. h. eine Funktion, die eine andere Funktion "einwickelt", sodass sie etwas vorher oder nachher tun kann). Das Anwenden des Dekorators @classmethod ist gleichbedeutend mit dem Aufruf:

```
how_many = classmethod(how_many)
```

Beachten Sie, dass die Methode `__init__` verwendet wird, um die Robot-Instanz mit einem Namen zu initialisieren. In dieser Methode erhöhen wir den `population`-Zähler um 1, da ein weiterer Roboter hinzugefügt wurde. Beachten Sie auch, dass die Werte von `self.name` spezifisch für jedes Objekt sind, was die Natur der Objektvariablen zeigt.

Denken Sie daran, dass Sie auf die Variablen und Methoden desselben Objekts ausschließlich mit `self` verweisen dürfen. Dies wird als *Attributreferenz* bezeichnet.

In diesem Programm sehen wir auch die Verwendung von *Docstrings* sowohl für Klassen als auch für Methoden. Wir können den Docstring der Klasse zur Laufzeit mit `Robot.__doc__` und den Docstring der Methode mit `Robot.sag_hallo.__doc__` abrufen.

In der Methode die verringern wir einfach den Zähler `Robot.population` um 1.

Alle Klassenmitglieder sind öffentlich. Eine Ausnahme: Wenn Sie Datenmember mit Namen verwenden, die den Doppelunterstrich-Präfix haben, wie `__privatevar`, verwendet Python *Name-Mangling*, um daraus effektiv eine private Variable zu machen.

Daher ist die Konvention, dass jede Variable, die nur innerhalb der Klasse oder des Objekts verwendet werden soll, mit einem Unterstrich beginnen sollte, und alle anderen Namen sind öffentlich und können von anderen Klassen/Objekten verwendet werden. Denken Sie daran, dass dies nur eine Konvention ist und nicht von Python erzwungen wird (außer beim Doppelunterstrich-Präfix).

Hinweis für C++/Java/C#-Programmierer

Alle Klassenmitglieder (einschließlich der Datenmember) sind öffentlich (*public*) und alle Methoden sind in Python virtuell (*virtual*).

13.6 Vererbung

( inheritance)

Einer der größten Vorteile der objektorientierten Programmierung ist die Wiederverwendung von Code, und eine der Möglichkeiten, wie dies erreicht wird, ist der Mechanismus der Vererbung. Vererbung kann man sich am besten als eine *Typ- und Subtyp*-Beziehung zwischen Klassen vorstellen.

Angenommen, Sie möchten ein Programm schreiben, das die Lehrer und Schüler in einem College verwaltet. Sie haben einige gemeinsame Merkmale wie Name, Alter und Adresse. Sie haben auch spezifische Merkmale wie Gehalt, Kurse und Urlaubstage für Lehrer sowie Noten und Gebühren für Schüler.

Sie könnten zwei unabhängige Klassen für jeden Typ erstellen und diese verarbeiten, aber das Hinzufügen eines neuen gemeinsamen Merkmals würde bedeuten, dass Sie es zu beiden unabhängigen Klassen hinzufügen müssen. Dies wird schnell unübersichtlich.

Eine bessere Methode wäre, eine gemeinsame Klasse namens `SchoolMember` zu erstellen und dann die Klassen `Teacher` und `Student` von dieser Klasse erben zu lassen, d. h. sie werden Untertypen dieses Typs (der Klasse), und dann können wir spezifische Merkmale zu diesen Untertypen hinzufügen.

Dieser Ansatz hat viele Vorteile. Wenn wir Funktionalität in SchoolMember hinzufügen oder ändern, wirkt sich dies automatisch auch auf die Untertypen aus. Beispielsweise können Sie ein neues ID-Karten-Feld sowohl für Lehrer als auch für Schüler hinzufügen, indem Sie es einfach zur Klasse SchoolMember hinzufügen. Änderungen in den Untertypen beeinflussen jedoch nicht die anderen Untertypen. Ein weiterer Vorteil ist, dass Sie ein Lehrer- oder Schülerobjekt als ein SchoolMember-Objekt betrachten können, was in manchen Situationen nützlich sein kann, z. B. beim Zählen der Anzahl von Schulmitgliedern. Dies wird **Polymorphismus** genannt, bei dem ein Untertyp in jeder Situation ersetzt werden kann, in der ein Obertyp erwartet wird, d. h. das Objekt kann als Instanz der Elternklasse behandelt werden.

Beachten Sie auch, dass wir den Code der Elternklasse wiederverwenden und ihn nicht in verschiedenen Klassen wiederholen müssen, wie wir es hätten tun müssen, wenn wir unabhängige Klassen verwendet hätten.

Die Klasse SchoolMember wird in dieser Situation als **Basisklasse** oder **Superklasse** bezeichnet. Die Klassen Teacher und Student werden die **abgeleiteten Klassen** (derived class) oder **Subklassen** genannt.

Wir sehen dieses Beispiel nun als Programm an:

Beispiel oop_subclass_de.py

Quellcode

```
class Mensch:
    '''Repräsentiert sowohl Schüler als auch Lehrer'''
    def __init__(self, name, alter):
        self.name = name
        self.alter = alter
        print('(Initialisiere Mensch: {})'.format(self.name))

    def info(self):
        '''Druckt Zusammenfassung (ohne Zeilenende!)'''
        print('Name:"{}" Alter:"{}"'.format(self.name, self.alter), end=" ")


class Lehrer(Mensch):
    '''Repräsentiert einen Lehrer'''
    def __init__(self, name, alter, gehalt):
        Mensch.__init__(self, name, alter)
        self.gehalt = gehalt
        print('(Initialisiere Lehrer: {})'.format(self.name))

    def info(self):
        Mensch.info(self)
        print('Gehalt: {}'.format(self.gehalt))


class Schüler(Mensch):
    '''Repräsentiert einen Schüler'''
    def __init__(self, name, alter, noten):
        Mensch.__init__(self, name, alter)
        self.noten = noten
        print('(Initialisiere Schüler: {})'.format(self.name))

    def info(self):
        Mensch.info(self)
        print('Noten: {}'.format(self.noten))
```

```

l = Lehrer('Mrs. Shrividya', 40, 30000)
s = Schüler('Swaroop', 25, 75)

# Drucke eine Leerzeile
print()

personen = [l, s]
for eine_person in personen:
    # funktioniert sowohl für Lehrer wie auch für Schüler
    eine_person.info()
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
$ python3 oop_subclass_de.py
(Initialisiere Mensch: Mrs. Shrividya)
(Initialisiere Lehrer: Mrs. Shrividya)
(Initialisiere Mensch: Swaroop)
(Initialisiere Schüler: Swaroop)

Name:"Mrs. Shrividya" Alter:"40" Gehalt: 30000
Name:"Swaroop" Alter:"25" Noten: 75
```

Wie es funktioniert:

Um Vererbung zu verwenden, geben wir die Namen der Basisklassen in einem Tupel nach dem Klassennamen in der Klassendefinition an (zum Beispiel `class Lehrer(Mensch)`). Als Nächstes sehen wir, dass die Methode `__init__` der Basisklasse explizit mit der Variablen `self` aufgerufen wird, sodass wir den Basisklassenanteil einer Instanz in der Unterklasse initialisieren können. Dies ist sehr wichtig zu merken – da wir eine `__init__`-Methode in den Unterklassen `Lehrer` und `Schüler` definieren, ruft Python den Konstruktor der Basisklasse `Mensch` nicht automatisch auf; Sie müssen ihn ausdrücklich selbst aufrufen.

Wenn wir dagegen keine `__init__`-Methode in einer Unterklasse definiert haben, ruft Python automatisch den Konstruktor der Basisklasse auf.

Während wir Instanzen von Lehrer oder Schüler wie eine Instanz von Mensch behandeln könnten und auf die `info`-Methode von `Mensch` zugreifen könnten, indem wir einfach `Lehrer.info` oder `Schüler.info` schreiben, definieren wir stattdessen eine weitere `info`-Methode in jeder Unterklasse (wobei wir die Methode tell der Klasse `Mensch` für einen Teil davon verwenden), um sie für diese Unterklasse anzupassen. Aufgrund dessen verwendet Python bei `Lehrer.info` die `info`-Methode der Unterklasse statt der Oberklasse. Wenn es jedoch keine `info`-Methode in der Unterklasse gäbe, würde Python die `info`-Methode der Oberklasse verwenden. Python beginnt immer damit, Methoden im tatsächlichen Typ der Unterklasse zu suchen, und wenn es dort nichts findet, sucht es in den Basisklassen der Unterklasse, eine nach der anderen, in der Reihenfolge, in der sie im Tupel der Klassendefinition angegeben sind (hier haben wir nur eine Basisklasse, aber es könnten mehrere vorhanden sein).

Ein Hinweis zur Terminologie – wenn mehr als eine Klasse im Vererbungs-Tupel aufgelistet ist, nennt man dies *Mehrfachvererbung* (multiple inheritance).

Der Parameter `end` wird in der `print`-Funktion in der `info()`-Methode der Oberklasse verwendet, um eine Zeile auszugeben und die nächste Ausgabe in derselben Zeile fortzusetzen. Dies ist ein Trick, um zu verhindern, dass print ein `\n` (Newline-Symbol) am Ende der Ausgabe druckt.

13.7 Zusammenfassung

Wir haben nun die verschiedenen Aspekte von Klassen und Objekten sowie die damit verbundenen Terminologien untersucht. Wir haben auch die Vorteile und Fallstricke der objektorientierten Programmierung betrachtet. Python ist stark objektorientiert, und das sorgfältige Verständnis dieser Konzepte wird Ihnen langfristig sehr helfen.

Als Nächstes werden wir lernen, wie man mit Eingabe/Ausgabe umgeht und wie man in Python auf Dateien zugreift.

CHAPTER 14

Eingabe und Ausgabe

- ( input: _Eingabe)
- ( output: _Ausgabe)

Es wird Situationen geben, in denen Ihr Programm mit dem Benutzer interagieren muss. Zum Beispiel möchten Sie möglicherweise Eingaben vom Benutzer entgegennehmen und dann einige Ergebnisse zurückgeben. Dies können wir mit der `input()`-Funktion bzw. der `print()`-Funktion erreichen.

Für die Ausgabe können wir auch die verschiedenen Methoden der `str` (String)-Klasse verwenden. Sie können z. B. die Methode `rjust` verwenden, um eine Zeichenkette zu erhalten, die rechtsbündig mit einer bestimmten Breite ausgerichtet ist. Weitere Details finden Sie unter `help(str)`.

Eine weitere häufige Art von Ein- und Ausgabe ist der Umgang mit Dateien. Die Fähigkeit, Dateien zu erstellen, zu lesen und zu schreiben, ist für viele Programme unerlässlich, und wir werden diesen Aspekt in diesem Kapitel untersuchen.

14.1 Benutzereingaben

( user input)

Beispiel io_input_de.py

Quellcode

```
def rückwärts(text):
    return text[::-1]


def ist_es_ein_palindrome(text):
        # klein/Großschreibung ignorieren:
        # .lower() wandelt einen Textstring in
        # Kleinbuchstaben um
```

```
    return text.lower() == rückwärts(text).lower()

# Beispiel für Palindrome: Renter, Kajak, Lagerregal
antwort_text = input("Bitte einen Text eingeben: ")
if ist_es_ein_palindrome(antwort_text):
    print("Ja, das ist ein Palindrome")
else:
    print("Nein, das ist kein Palindrom")
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
$ python3 io_input_de.py
Bitte einen Text eingeben: Rentner
Ja, das ist ein Palindrome


$ python3 io_input_de.pyBitte einen Text eingeben: Marmelade
Nein, das ist kein Palindrom
```

Wie es funktioniert

Wir verwenden die Slice-Funktion, um den Text umzukehren. Wir haben bereits gesehen, wie wir *Slices aus Sequenzen* mit dem Code `seq[a:b]` erstellen können, beginnend bei Position `a` bis Position `b`. Wir können auch ein drittes Argument angeben, das den *Schritt* (step) angibt, mit dem das Slicing durchgeführt wird. Der Standard-Schritt ist 1, wodurch ein durchgehender Teil des Textes zurückgegeben wird. Ein negativer Schritt, z. B. `-1`, gibt den Text in umgekehrter Reihenfolge zurück.

Die Funktion `input()` nimmt eine Zeichenkette als Argument entgegen und zeigt sie dem Benutzer an. Dann wartet sie darauf, dass der Benutzer etwas eingibt und die Eingabetaste drückt. Sobald der Benutzer die Eingabe getätigt und die Eingabetaste gedrückt hat, gibt die Funktion `input()` den eingegebenen Text zurück.

Wir nehmen diesen Text und kehren ihn um. Wenn der ursprüngliche Text und der umgekehrte Text gleich sind, dann ist der Text ein palindrome.

14.1.1 Hausaufgabe

Das Überprüfen, ob ein Text ein Palindrom ist, sollte auch Interpunktion, Leerzeichen und Groß-/Kleinschreibung ignorieren. Zum Beispiel ist "Rise to vote, sir." ebenfalls ein Palindrom, aber unser aktuelles Programm erkennt dies nicht. Können Sie das Programm so anpassen, dass es solche Fälle erkennt?

Einen Tipp zu dieser Hausübung finden Sie in der Fußnote[1]

14.2 Dateien

( files)

Sie können Dateien zum Lesen oder Schreiben öffnen und verwenden, indem Sie ein Objekt der Klasse `file` erstellen und dessen Methoden `read`, `readline` oder `write` verwenden, um aus der Datei zu lesen (read) oder in sie zu schreiben (write). Die Fähigkeit, auf eine Datei zuzugreifen oder sie zu verändern, hängt vom Modus ab, in dem die Datei geöffnet

[1] Verwenden Sie ein Tupel (eine Liste aller Satzzeichen finden Sie online), um alle zu ignorierenden Zeichen zu speichern. Verwenden Sie dann den Mitgliedschaftstest, um festzustellen, ob ein Zeichen entfernt werden soll, z. B. `forbidden = ('!', '?', '.', ...)`.

wurde. Wenn Sie mit der Datei fertig sind, rufen Sie `close` auf, um Python mitzuteilen, dass die Datei nicht länger benötigt wird.

Beispiel io_using_file_de.py

Quellcode

```
gedicht = '''\
Programmieren macht Spaß
wenn die Arbeit getan ist.
Wenn Du Spaß an der Arbeit haben willst:
    verwende Python!
'''

# Datei schreiben ("w" steht für "writing")
f = open('gedicht.txt', 'w')
# Text in die Datei schreiben
f.write(gedicht)
# Datei schließen
f.close()

# Wenn bei open kein Modus angegeben wird,
# öffnet Python die Datei im Lesemodus
# (r steht für "read")
f = open('gedicht.txt')
while True:
    zeile = f.readline()
    # Eine Zeile mit Länge 0 bedeutet End of File (EOF)
    if len(zeile) == 0:
        break
    # Jede Zeile in der Datei hat schon ein
    # Zeilenende-Zeichen am Ende "eingebaut"
    print(zeile, end='')
# Datei schließen
f.close()
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
python3 io_using_file_de.py
Programmieren macht Spaß
wenn die Arbeit getan ist.
Wenn Du Spaß an der Arbeit haben willst:
    verwende Python!
```

Wie es funktioniert:

Beachten Sie, dass wir ein neues Dateiobjekt einfach durch die Verwendung der Funktion `open` erzeugen können. Wir öffnen (oder erzeugen, falls die Datei noch nicht existiert) diese Datei mit der eingebauten Funktion `open`, wobei wir den Namen der Datei und den Modus angeben, in dem wir die Datei öffnen möchten. Der Modus kann ein Lesemodus (`'r'`), ein Schreibmodus (`'w'`) oder ein Anhängemodus (`'a'`) sein. Wir können außerdem angeben, ob wir

im Textmodus (`'t'`) oder im Binärmodus (`'b'`) lesen, schreiben oder anhängen. Es gibt in der Tat noch viele weitere Modi; `help(open)` gibt Ihnen die Details. Standardmäßig behandelt `open()` eine Datei als Textdatei (`'t'`) und öffnet sie im Lesemodus (`'r'`).

In unserem Beispiel öffnen wir die Datei zunächst im Schreib-Textmodus und verwenden dann die Methode `write` des Dateiobjekts, um unsere String-Variable `gedicht` in die Datei zu schreiben, und schließen anschließend die Datei mit `close`.

Danach öffnen wir dieselbe Datei erneut zum Lesen. Wir müssen keinen Modus angeben, da „Textdatei lesen“ der Standardmodus ist. Wir lesen jede Zeile der Datei mithilfe der Methode `readline` innerhalb einer Schleife ein. Diese Methode gibt eine vollständige Zeile zurück, *inklusive* des Zeilenumbruchs am Ende. Wenn eine leere Zeichenkette zurückgegeben wird, bedeutet dies, dass wir das Ende der Datei erreicht haben, und wir verlassen die Schleife mittels `break`.

Am Ende schließen wir die Datei mit `close`.

Wir können anhand der Ausgabe erkennen, dass dieses Programm tatsächlich in die Datei `gedicht.txt` geschrieben und anschließend wieder daraus gelesen hat.

14.3 Pickle

( pickle: *eindosen, konservieren*)

Python bietet ein Standardmodul namens `pickle`, mit dem Sie beliebige einfache Python-Objekte in einer Datei speichern und später wieder abrufen können. Diese Art des Speicherns wird als persistente Speicherung bezeichnet.

Beispiel io_pickle_de.py

Quellcode

```
1  import pickle
2
3  # Der Dateiname in dem das Python-Objekt gespeichert wird
4  dateiname = 'einkaufsliste.data'
5  # Ein einfaches Python-Objekt: eine Liste
6  einkaufsliste = ['Apfel', 'Mango', 'Karotte']
7
8  # In die Datei schreiben (wb bedeutet "write binary")
9  f = open(dateiname, 'wb')
10 # Das Ojbekt in die Datei reinschreiben (dump)
11 pickle.dump(einkaufsliste, f)
12 f.close()
13
14 # Die Variable aus dem Speicher löschen
15 del einkaufsliste
16
17 # Aus der Datei lesen (rb bedeutet "read binary")
18 f = open(dateiname, 'rb')
19 # Objekt aus der Datei laden
20 liste_aus_der_festplatte = pickle.load(f)
21 print(liste_aus_der_festplatte)
22 f.close()
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
```

Wie es funktioniert:

Um ein Objekt in einer Datei zu speichern, müssen wir die Datei im Schreib-Binärmodus öffnen (`'wb'`) und anschließend die Funktion `dump` des pickle-Moduls aufrufen. Dieser Prozess wird *pickling* genannt.

Wir rufen das Objekt dann mithilfe der Funktion `load` Funktion des pickle-Moduls wieder ab, die das Objekt zurückliefert. Dieser Vorgang wird *unpickling* genannt.

14.4 Unicode

Bisher haben wir beim Schreiben von Strings oder beim Lesen/Schreiben von Dateien nur einfache englische Zeichen verwendet. Sowohl englische als auch nicht-englische Zeichen können in Unicode dargestellt werden, und Python3 speichert alle Stringvariablen standardmäßig in Unicode.

Wenn Daten über das Internet übertragen werden, müssen sie als Bytes gesendet werden – etwas, das der Computer problemlos versteht. Die Regeln für die Übersetzung von Unicode (dem Format, in dem Python Strings speichert) in Bytes werden *Encoding* genannt. Eine gängige Kodierung ist UTF-8. Wir können im UTF-8-Format lesen und schreiben, indem wir beim Aufruf von `open` ein Argument zur Kodierung angeben.

Beispiel io_unicode_de.py

Quellcode

```
1 # encoding=utf-8
2 import io
3
4 f = io.open("abc.txt", "wt", encoding="utf-8")
5 f.write(u"Hier steht eine nicht-englische Sprache...")
6 f.close()
7
8 text = io.open("abc.txt", encoding="utf-8").read()
9 print(text)
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Wie es funktioniert:

Wir verwenden `io.open` und geben beim ersten Aufruf das Argument `encoding` an, um die Nachricht zu kodieren, und beim zweiten Aufruf erneut, um die Nachricht zu dekodieren. Beachten Sie, dass wir das Encoding-Argument nur im Textmodus verwenden sollten.

Wann immer wir ein Programm schreiben, das Unicode-Literale (also Strings mit einem vorangestellten u) verwendet, müssen wir sicherstellen, dass Python weiß, dass unser Programm UTF-8 benutzt. Dazu müssen wir die Zeile

```
# encoding=utf
```

an den Anfang des Programms setzen.

Weitere Informationen finden Sie in folgenden Quellen:

– The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets – Python Unicode Howto – Pragmatic Unicode (Vortrag von Nat Batchelder)

14.5 Zusammenfassung

Wir haben verschiedene Arten der Ein- und Ausgabe besprochen, den Umgang mit Dateien, das Pickle-Modul und Unicode.

Als Nächstes werden wir das Konzept der Ausnahmen (exceptions) untersuchen.

CHAPTER 15

Ausnahmen

- ( exceptions)

Exceptions (Ausnahmen) treten auf, wenn **außergewöhnliche** Situationen in Ihrem Programm auftreten. Was wäre zum Beispiel, wenn Sie eine Datei lesen möchten und die Datei nicht existiert? Oder was, wenn Sie sie versehentlich gelöscht haben, während das Programm lief? Solche Situationen werden mit **exceptions** behandelt.

Ebenso: Was wäre, wenn Ihr Programm ungültige Anweisungen enthielte? Python **meldet** dies und teilt Ihnen mit, dass ein **Fehler** vorliegt. (Python *raises* an *error*)

15.1 Fehler

- ( errors)

Betrachten wir einen einfachen Aufruf der `print`-Funktion. Was wäre, wenn wir `print` fälschlicherweise als `Print` schreiben? Beachten Sie die Großschreibung. In diesem Fall **wirft** Python einen Syntaxfehler.

```
>>> Print("Hallo Welt")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'Print' is not defined
>>> print("Hallo Welt")
Hallo World
```

Beachten Sie, dass ein `NameError` ausgelöst wird und auch der Ort, an dem der Fehler erkannt wurde, angezeigt wird. Dies ist die Aufgabe eines **Fehlerbehandlers** (*error_handlers*) für diesen Fehler.

15.2 Exceptions

Wir werden **versuchen** (*try*), eine Eingabe vom Benutzer zu lesen. Geben Sie die folgende Zeile ein und drücken Sie die **Eingabetaste**. Wenn Ihr Computer Sie zur Eingabe auffordert, drücken Sie stattdessen `[Strg+D]` auf einem Mac bzw. Linux-Rechner oder `[Strg+Z]` unter Windows und schauen Sie, was passiert. (Wenn Sie Windows verwenden und keine der Optionen funktioniert, können Sie `[Strg+C]` in der Eingabeaufforderung versuchen, um stattdessen einen `KeyboardInterrupt`-Fehler zu erzeugen.)

```
>>> s = input('Schreib was --> ')
Enter something --> Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
EOFError
```

Python löst einen Fehler namens `EOFError` (*End_of_file*) aus, was im Grunde bedeutet, dass es ein *Dateiende*-Symbol (das durch `[Strg+D]` dargestellt wird) gefunden hat, als es nicht damit gerechnet hat.

15.3 Behandlung von Ausnahmen

- ( handling Exceptions)

Wir können *Exceptions* mit der `try...except`-Anweisung behandeln. Wir setzen unsere üblichen Anweisungen in den `try`-Block und alle unsere Fehlerbehandler in den `except`-Block.

Beispiel exceptions_handle_de.py

Quellcode

```
1 try:
2     text = input('Schreibe etwas --> ')
3 except EOFError:
4     print('Warum plötzlich ein EOF (end of file) ?')
5 except KeyboardInterrupt:
6     print('Vorgang wurde abgebrochen.')
7 else:
8     print('Du hast geschrieben: {}'.format(text))
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe:

```
$ python3 exceptions_handle_de.py
# drücke STRG + d
Schreibe etwas --> Warum plötzlich ein EOF (end of file) ?
# drücke STRG + c
horst@BigBlackTower:~/code/byte-of-python_deutsch_horst/programs$ python3 exceptions_
↪handle_de.py
Schreibe etwas --> ^CVorgang wurde abgebrochen.
#
horst@BigBlackTower:~/code/byte-of-python_deutsch_horst/programs$ python3 exceptions_
↪handle_de.py
Schreibe etwas --> Keine Exceptions
Du hast geschrieben: Keine Exceptions
```

Wie es funktioniert

Wir platzieren alle Anweisungen, die Ausnahmen/Fehler auslösen könnten, innerhalb des `try`-Blocks und definieren anschließend Behandler für die entsprechenden Fehler/Ausnahmen in der `except`-Klausel bzw. im `except`-Block. Die `except`-Klausel kann entweder einen einzelnen angegebenen Fehler bzw. eine einzelne Ausnahme oder eine in Klammern angegebene Liste von Fehlern/Ausnahmen behandeln. Wenn keine Namen von Fehlern oder Ausnahmen angegeben werden, behandelt sie *alle* Fehler und Ausnahmen.

Beachten Sie, dass jeder `try`-Klausel mindestens eine `except`-Klausel zugeordnet sein muss. Andernfalls stellt sich die Frage, welchen Zweck ein `try`-Block überhaupt erfüllen soll.

Falls ein Fehler oder eine Ausnahme nicht behandelt wird, wird der Standardbehandler von Python aufgerufen, der lediglich die Ausführung des Programms beendet und eine Fehlermeldung ausgibt. Dies haben wir oben bereits in Aktion gesehen.

Sie können außerdem eine `else`-Klausel mit einem `try...except`-Block kombinieren. Die `else`-Klausel wird ausgeführt, wenn keine Ausnahme auftritt.

Im nächsten Beispiel werden wir außerdem sehen, wie das Ausnahmeobjekt abgerufen werden kann, sodass zusätzliche Informationen daraus gewonnen werden können.

15.4 Ausnahmen auslösen

( raising exceptions)

Sie können Ausnahmen mithilfe der Anweisung `raise` *auslösen*, indem Sie den Namen des Fehlers/der Ausnahme sowie das Ausnahmeobjekt angeben, das *geworfen* werden soll.

Der Fehler bzw. die *Exception*, die Sie auslösen können, muss eine Klasse sein, die direkt oder indirekt von der Klasse `Exception` abgeleitet ist.

Beispiel exceptions_raise_de.py'

Quellcode

```
class ZuKurzeEingabetException(Exception):
    '''Eine user-definierte Exception Klasse'''
    def __init__(self, länge, minimum):
        Exception.__init__(self)
        self.länge = länge
        self.minimum = minimum

try:
    text = input('Schreib etwas --> ')
    if len(text) < 3:
        raise ZuKurzeEingabetException(len(text), 3)
    # weitere Programmierzeilen ....
except EOFError:
    print('Warum End of File (EoF) ?')
except ZuKurzeEingabetException as ex:
    print(('ZuKurzeEingabeException: Die Eingabe war: ' +
           '{0} Zeichen lang, sollte aber mindestens {1} Zeichen haben')
          .format(ex.länge, ex.minimum))
```

```
else:
    print('No exception was raised.')
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Wie es funktioniert

Hier erstellen wir unseren eigenen Ausnahmetyp. Dieser neue Ausnahmetyp heißt `ZuKurzeEingabetException`. Er besitzt zwei Felder – `länge`, welches die Länge der gegebenen Eingabe darstellt, und `minimum`, welches die minimale Länge angibt, die das Programm erwartet hat.

In der `except`-Klausel geben wir die Fehlerklasse an, die mithilfe von `as` unter einem Variablennamen gespeichert wird, welcher das entsprechende Fehler-/Ausnahmeobjekt enthält. Dies ist analog zu Parametern und Argumenten bei einem Funktionsaufruf. Innerhalb dieser speziellen `except`-Klausel verwenden wir die Felder `länge` und `minimum` des Ausnahmeobjekts, um dem Benutzer eine geeignete Meldung auszugeben.

15.5 Try ... Finally

- ( try: *versuchen, probieren*)
- ( finally: *abschließend, schlussendlich*)

Angenommen, Sie lesen in Ihrem Programm eine Datei. Wie stellen Sie sicher, dass das Dateiobjekt ordnungsgemäß geschlossen wird – unabhängig davon, ob eine Ausnahme ausgelöst wurde oder nicht? Dies kann mithilfe des `finally`-Blocks erreicht werden.

Um zu funktionieren, braucht das folgende Program eine Datei namens poem.txt im selben Verzeichnis. Es ist nicht so wichtig, *was* genau in `poem.txt` drinsteht, so lange es *mehrere* Textzeilen sind.

Beispiel exceptions_finally_de.py

Quellcode

```
import sys
import time

f = None   # f für file
try:
    f = open("poem.txt")
    # Textdatei öffnen und Inhalt lesen
    while True:
        zeile  = f.readline()
        if len(zeile) == 0:
            break
        print(zeile, end='')
        sys.stdout.flush()
        print("Drücke jetzt STRG+c")
        # Sicherstellen daß es eine Weile läuft
        time.sleep(2) # 2 Sekunden nichts tun
except IOError:
    print("Ich konnte die Datei poem.txt nicht finden")
except KeyboardInterrupt:
    print("!! Lesevorgang wurde unterbochen mit Tastatur")
```

```
21 finally:
22     if f:
23         f.close()
24     print("(Aufräumen: Datei wurde geschlossen)")
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
$ python3 exceptions_finally_de.py
Programming is fun
Drücke jetzt STRG+c
When the work is done
Drücke jetzt STRG+c
if you wanna make your work also fun:
Drücke jetzt STRG+c
^C!! Lesevorgang wurde unterbochen mit Tastatur
(Aufräumen: Datei wurde geschlossen)
```

Wie es funktioniert

Wir führen die üblichen Datei-Leseoperationen durch, haben jedoch zusätzlich künstlich eine Pause von 2 Sekunden nach der Ausgabe jeder Zeile mithilfe der Funktion `time.sleep` eingefügt, damit das Programm langsam ausgeführt wird (Python ist von Natur aus sehr schnell). Während das Programm noch läuft, drücken Sie `ctrl + c`, um das Programm zu unterbrechen bzw. abzubrechen.

Beachten Sie, dass die *Exception* `KeyboardInterrupt` ausgelöst wird und das Programm beendet wird. Bevor das Programm jedoch beendet wird, wird die `finally`-Klausel ausgeführt und das Dateiobjekt wird stets geschlossen.

Beachten Sie außerdem, dass eine Variable mit dem Wert `0` oder `None` oder eine Variable, die eine leere Sequenz oder Sammlung enthält, von Python als `False` betrachtet wird. Deshalb können wir im obigen Code `if f:` verwenden.

Beachten Sie ebenfalls, dass wir nach `print` den Aufruf `sys.stdout.flush()` verwenden, damit die Ausgabe sofort auf dem Bildschirm erscheint.

15.6 Das `with` statement

- ( with: *mit*)

Das Erwerben einer Ressource innerhalb des `try`-Blocks und das anschließende Freigeben der Ressource im `finally`-Block ist ein häufig auftretendes Muster. Daher existiert außerdem die Anweisung `with`, die dies auf elegante Weise ermöglicht:

Das folgende Programm benötigt ebenfalls eine Textdatei namens `poem.txt` im gleichen Verzeichnis.

Beispiel exceptions_using_with.py

Quellcode

```
1 with open("poem.txt") as f:
2     for line in f:
3         print(line, end='')
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Wie es funktioniert

Die Ausgabe sollte dieselbe sein wie im vorherigen Beispiel. Der Unterschied besteht hier darin, dass wir die Funktion `open` zusammen mit der Anweisung `with` verwenden – wir überlassen das Schließen der Datei der automatischen Behandlung durch `with open`.

Hinter den Kulissen geschieht Folgendes: Es existiert ein Protokoll, das von der Anweisung `with` verwendet wird. Dabei wird das Objekt abgerufen, das durch die Anweisung `open` zurückgegeben wird; nennen wir es in diesem Fall „thefile“.

Es wird *immer* die Funktion `thefile.__enter__` aufgerufen, bevor der darunterliegende Codeblock ausgeführt wird, und *immer* `thefile.__exit__`, nachdem der Codeblock beendet wurde.

Daher sollte der Code, den wir andernfalls in einem `finally`-Block geschrieben hätten, automatisch von der Methode `__exit__` übernommen werden. Dies hilft uns dabei, die wiederholte Verwendung expliziter `try...finally`-Anweisungen zu vermeiden.

Eine weiterführende Diskussion dieses Themas liegt außerhalb des Umfangs dieses Buches. Bitte lesen Sie daher PEP 343 für eine umfassende Erklärung.

15.7 Zusammenfassung

Wir haben die Verwendung der Anweisungen `try..except` und `try..finally` besprochen. Außerdem haben wir gesehen, wie eigene Ausnahmetypen erstellt und wie Ausnahmen ausgelöst werden können.

Als Nächstes werden wir die Python-Standardbibliothek untersuchen.

CHAPTER 16

Standardbibliothek

Die Python-Standardbibliothek enthält eine große Anzahl nützlicher Module und ist Teil jeder Standard-Python-Installation. Es ist wichtig, sich mit der Python-Standardbibliothek vertraut zu machen, da viele Probleme schnell gelöst werden können, wenn man mit den Möglichkeiten dieser Bibliotheken vertraut ist.

Wir werden einige der häufig verwendeten Module in dieser Bibliothek erkunden. Vollständige Details zu allen Modulen der Python-Standardbibliothek finden Sie im Abschnitt „Library Reference“ der Dokumentation, die mit Ihrer Python-Installation geliefert wird.

Lassen Sie uns einige nützliche Module erkunden.

Achtung

Falls Sie die Themen in diesem Kapitel zu fortgeschritten finden, können Sie dieses Kapitel überspringen. Ich empfehle jedoch dringend, zu diesem Kapitel zurückzukehren, wenn Sie sich mit der Programmierung in Python wohler fühlen.

16.1 Das sys-Modul

Das `sys`-Modul enthält systemspezifische Funktionen. Wir haben bereits gesehen, dass die Liste `sys.argv` die Befehlszeilenargumente enthält.

Angenommen, wir möchten die Version der verwendeten Python-Software überprüfen, das `sys`-Modul gibt uns diese Information.

```
>>> import sys
>>> sys.version_info
sys.version_info(major=3, minor=6, micro=0, releaselevel='final', serial=0)
>>> sys.version_info.major == 3
True
```

Wie es funktioniert:

Das sys-Modul hat ein version_info-Tuple, das uns die Versionsinformationen liefert. Der erste Eintrag ist die Hauptversionsnummer. Wir können diese Information extrahieren, um sie zu verwenden.

16.2 Das logging-Modul

Was tun Sie, wenn Sie Debug-Nachrichten oder wichtige Nachrichten irgendwo speichern möchten, um zu überprüfen, ob Ihr Programm so läuft, wie Sie es erwarten? Wie speichern Sie diese Nachrichten „irgendwo“? Dies kann mit dem logging-Modul erreicht werden.

Beispiel stdlib_logging_de.py

Quellcode

```
import os
import platform
import logging

if platform.platform().startswith('Windows'):
    logging_file = os.path.join(os.getenv('HOMEDRIVE'),
                                os.getenv('HOMEPATH'),
                                'test.log')
else:
    logging_file = os.path.join(os.getenv('HOME'),
                                'test.log')

print("Logging in Datei:", logging_file)

logging.basicConfig(
    level=logging.DEBUG,
    format='%(asctime)s : %(levelname)s : %(message)s',
    filename=logging_file,
    filemode='w',
)

logging.debug("Start des Programms")
logging.info("macht etwas")
logging.warning("stirbt gerade")
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
$ python3 stdlib_logging_de.py
Logging in Datei: /home/horst/test.log

$ cat /home/horst/test.log
2026-05-14 00:02:23,822 : DEBUG : Start des Programms
2026-05-14 00:02:23,823 : INFO : macht etwas
2026-05-14 00:02:23,823 : WARNING : stirbt gerade
```

Der Befehl cat wird in der Kommandozeile verwendet, um die Datei test.log zu lesen. Falls der Befehl cat nicht verfügbar ist, können Sie stattdessen die Datei test.log in einem Texteditor öffnen.

Wie es funktioniert:

Wir verwenden drei Module aus der Standardbibliothek – das `os`-Modul für die Interaktion mit dem Betriebssystem, das `platform`-Modul für Informationen über die Plattform, d. h. das Betriebssystem, und das `logging`-Modul, um Informationen zu protokollieren.

Zuerst prüfen wir, welches Betriebssystem wir benutzen, indem wir den von `platform.platform()` zurückgegebenen String überprüfen (für weitere Informationen siehe `import platform; help(platform)`). Falls es Windows ist, ermitteln wir das Home-Laufwerk, den Home-Ordner und den Dateinamen, in dem wir die Informationen speichern wollen. Durch das Zusammenfügen dieser drei Teile erhalten wir den vollständigen Speicherort der Datei. Für andere Plattformen müssen wir lediglich den Home-Ordner des Benutzers kennen, und daraus erhalten wir ebenfalls den vollständigen Speicherort der Datei.

Wir verwenden die Funktion `os.path.join()`, um diese drei Teile des Speicherorts zusammenzufügen. Der Grund für die Verwendung einer speziellen Funktion anstatt die Strings einfach zu addieren besteht darin, dass diese Funktion sicherstellt, dass der vollständige Speicherort dem vom Betriebssystem erwarteten Format entspricht. Hinweis: die hier verwendete Methode `join()` aus dem `os`-Modul ist eine andere als die String-Methode `join()`, die wir an anderer Stelle in diesem Buch verwendet haben.

Wir konfigurieren das `logging`-Modul so, dass alle Meldungen in einem bestimmten Format in die von uns angegebene Datei geschrieben werden.

Schließlich können wir Meldungen einfügen, die entweder für das Debugging, zur Information, als Warnung oder sogar als kritische Meldungen gedacht sind. Sobald das Programm ausgeführt wurde, können wir diese Datei überprüfen und werden feststellen, was im Programm geschehen ist, selbst wenn dem Benutzer, der das Programm ausgeführt hat, keine Informationen angezeigt wurden.

CHAPTER 17

Module der Woche

( module-of-the-Week)

Es gibt noch viel mehr in der Standardbibliothek zu entdecken, wie zum Beispiel [Debugging](debugging, Umgang mit Kommandozeilenoptionen, reguläre Ausdrücke (*regular expressions*) und vieles mehr.

Der beste Weg, die Standardbibliothek weiter zu erkunden, ist, Doug Hellmanns ausgezeichnete Serie Python Module of the Week zu lesen (auch als Buch verfügbar) sowie in der Python-Dokumentation zu lesen.

17.1 Zusammenfassung

Wir haben einige der Funktionalitäten vieler Module der Python-Standardbibliothek erkundet. Es wird sehr empfohlen, die Dokumentation der Python-Standardbibliothek durchzublättern, um eine Vorstellung von allen verfügbaren Modulen zu bekommen.

Als Nächstes behandeln wir verschiedene Aspekte von Python, die unsere Rundreise durch Python vollständiger machen werden.

CHAPTER 18

Weitere Themen

Bisher haben wir die meisten verschiedenen Aspekte von Python abgedeckt, die Sie verwenden werden. In diesem Kapitel werden wir einige weitere Aspekte behandeln, die unser Wissen über Python abrunden.

18.1 Tupel zurückgeben und weitergeben

Haben Sie sich schon einmal gewünscht, zwei verschiedene Werte aus einer Funktion zurückgeben zu können? Das ist möglich. Alles, was Sie tun müssen, ist ein *Tupel* zu verwenden.

```
>>> def get_error_details():
...     return (2, 'details')
...
>>> errnum, errstr = get_error_details()
>>> errnum
2
>>> errstr
'details'
```

Beachten Sie, dass die Verwendung von `a, b = <irgendein Ausdruck>` das Ergebnis des Ausdrucks als ein Tupel mit zwei Werten interpretiert.

Dies bedeutet auch, dass der schnellste Weg, zwei Variablen in Python zu vertauschen, folgender ist:

```
>>> a = 5; b = 8
>>> a, b
(5, 8)
>>> a, b = b, a
>>> a, b
(8, 5)
```

18.2 Spezielle Methoden

Es gibt bestimmte Methoden wie `__init__` und `__del__`, die in Klassen eine besondere Bedeutung haben.

Hinweis

Spezielle Methoden werden manchmal als *dunder_methods* bezeichnet, verballhornt von *double_underscore*

- ( double: *doppelt*)
- ( underscore: *Unterstrich*)

Spezielle Methoden werden verwendet, um bestimmte Verhaltensweisen von eingebauten Typen nachzuahmen. Wenn Sie z. B. die Indexoperation `x[key]` für Ihre Klasse verwenden möchten (so wie Sie sie für Listen und Tupel verwenden), müssen Sie nur die Methode `__getitem__()` implementieren, und Ihre Aufgabe ist erledigt. Wenn Sie darüber nachdenken, ist das genau das, was Python für die `list`-Klasse selbst tut!

Einige nützliche spezielle Methoden sind in der folgenden Tabelle aufgelistet. Wenn Sie mehr über alle speziellen Methoden erfahren möchten, sehen Sie sich das Handbuch an.

- `__init__(self, ...)`
  - Diese Methode wird aufgerufen, kurz bevor das neu erstellte Objekt für die Verwendung zurückgegeben wird.
- `__del__(self)`
  - Wird aufgerufen, kurz bevor das Objekt zerstört wird (der Zeitpunkt ist unvorhersehbar, daher sollten Sie diese Methode vermeiden).
- `__str__(self)`
  - Wird aufgerufen, wenn wir die `print`-Funktion verwenden oder wenn `str()` verwendet wird.
- `__lt__(self, other)`
  - Wird aufgerufen, wenn der *kleiner als*-Operator (<) verwendet wird. Ähnlich gibt es spezielle Methoden für alle Operatoren (+, >, usw.).
- `__getitem__(self, key)`
  - Wird aufgerufen, wenn `x[key]` indexiert wird.
- `__len__(self)`
  - Wird aufgerufen wenn die *build_in* Python-Funktion `len()`mit dem Objekt aufgerufen wird.

18.3 Einzelne Anweisungsblöcke (single statement blocks)

Wir haben gesehen, dass jeder Block von Anweisungen durch seine eigene Einrückungsebene vom Rest getrennt ist. Nun, es gibt eine Ausnahme. Wenn Ihr Block von Anweisungen nur eine einzige Anweisung enthält, dann können Sie diese in derselben Zeile wie beispielsweise eine Bedingungsanweisung oder eine Schleifenanweisung angeben. Das folgende Beispiel sollte dies verdeutlichen:

```
>>> flag = True
>>> if flag: print('Yes')
...
Yes
```

Beachten Sie, dass die einzelne Anweisung direkt verwendet wird und nicht als separater Block. Obwohl Sie dies verwenden können, um Ihr Programm *kleiner* zu machen, empfehle ich dringend, diese Abkürzung zu vermeiden – außer beim Prüfen von Fehlern –, hauptsächlich weil es viel einfacher ist, eine zusätzliche Anweisung hinzuzufügen, wenn Sie die korrekte Einrückung verwenden.

18.4 Lambda-Formen

Eine lambda-Anweisung (auch: anonyme Funktion) wird verwendet, um neue Funktionsobjekte zu erzeugen. Im Wesentlichen nimmt lambda einen Parameter gefolgt von einem einzigen Ausdruck. Die Lambda wird zum Rumpf der Funktion. Der Wert dieses Ausdrucks wird von der neuen Funktion zurückgegeben.

Beispiel more_lambda_de.py

Quellcode

```
1 points = [{'x': 2, 'y': 3},
2           {'x': 4, 'y': 1}]
3 points.sort(key=lambda i: i['y'])
4 print(points)
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
$ python more_lambda.py
[{'y': 1, 'x': 4}, {'y': 3, 'x': 2}]
```

Wie es funktioniert:

Beachten Sie, dass die Methode `sort` einer `list` einen Parameter namens `key` akzeptiert, der bestimmt, wie die Liste sortiert wird (gewöhnlich wissen wir nur etwas über aufsteigende oder absteigende Ordnung). In unserem Fall möchten wir eine benutzerdefinierte Sortierung durchführen, und dafür müssen wir eine Funktion schreiben. Anstatt einen separaten `def`-Block für eine Funktion zu schreiben, die nur an dieser einen Stelle verwendet wird, benutzen wir einen Lambda-Ausdruck, um eine neue Funktion zu erzeugen.

18.5 List Comprehension

List Comprehensions werden verwendet, um eine neue Liste aus einer bestehenden Liste abzuleiten. Angenommen, Sie haben eine Liste von Zahlen und Sie möchten eine entsprechende Liste erhalten, in der alle Zahlen mit 2 multipliziert wurden, jedoch nur dann, wenn die Zahl selbst größer als 2 ist. List Comprehensions sind ideal für solche Situationen.

Beispiel (speichern als more_list_comprehension_de.py):

Beispiel more_list_comprehension_de.py

Quellcode

```
1 listone = [2, 3, 4]
2 listtwo = [2*i for i in listone if i > 2]
3 print(listtwo)
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
$ python more_list_comprehension.py
[6, 8]
```

Wie es funktioniert:

Hier leiten wir eine neue Liste ab, indem wir die auszuführende Manipulation angeben (`2*i`), wenn eine bestimmte Bedingung erfüllt ist (`if i > 2`). Beachten Sie, dass die ursprüngliche Liste unverändert bleibt.

Der Vorteil von List Comprehensions besteht darin, dass sie die Menge an Boilerplate-Code reduzieren, den wir benötigen, wenn wir Schleifen verwenden, um jedes Element einer Liste zu verarbeiten und es in einer neuen Liste zu speichern.

18.6 Entgegennahme von Tupeln und Dictionaries in Funktionen

- () ***args**: steht für (beliebig viele) **arg**uments)
- () ****kwargs**: steht für (beliebig viele) **k**ey**w**ord **arg**uments)

Es gibt eine besondere Möglichkeit, Parameter an eine Funktion als Tupel oder Dictionary zu übergeben, indem man vor dem Variablennamen das Präfix * bzw. ** verwendet. Dies ist nützlich, wenn die Funktion eine variable Anzahl von Argumenten annimmt.

```
>>> def potenzsumme(power, *args):
...     '''Gibt die Summe aller Elemente hoch power zurück'''
...     total = 0
...     for element in args:
...         total += i**power
...         #oder: total += pow(i, power)
...     return total
...
>>> potenzsumme(2, 3, 4) # 3^2 + 4^2
25
>>> potenzsumme(3, 10, 5 ) # 10^3 + 5^3
1125
```

Da wir ein *-Präfix vor der Variable `args` haben, werden alle zusätzlichen Argumente, die an die Funktion übergeben werden, in `args` als Tupel gespeichert. Wenn stattdessen ein **-Präfix verwendet worden wäre, würden die zusätzlichen Parameter als Schlüssel/Wert-Paare eines Dictionaries betrachtet.

18.7 Die `assert` - Anweisung

Die assert-Anweisung wird verwendet, um sicherzustellen, dass etwas wahr ist. Wenn Sie z. B. sehr sicher sind, dass Sie mindestens ein Element in einer Liste haben, die Sie verwenden, und Sie dies überprüfen möchten und einen Fehler auslösen wollen, wenn es nicht zutrifft, dann ist die `assert`-Anweisung für diese Situation ideal. Wenn die Assert-Anweisung fehlschlägt, wird ein `AssertionError` ausgelöst. Die Methode `pop()` entfernt letzte Element einer Liste und gibt es zurück.

```
>>> mylist = ['item']
>>> assert len(mylist) >= 1
>>> mylist.pop()
'item'
>>> assert len(mylist) >= 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AssertionError
```

Die `assert`-Anweisung sollte mit Bedacht eingesetzt werden. Die meiste Zeit ist es besser, Ausnahmen abzufangen, entweder das Problem zu behandeln oder dem Benutzer eine Fehlermeldung anzuzeigen und dann zu beenden.

18.8 Dekoratoren

- ( decorators)

Dekoratoren sind eine Abkürzung, um Wrapper-Funktionen anzuwenden. Dies ist hilfreich, um Funktionalität immer wieder mit demselben Code zu „umhüllen“. Zum Beispiel habe ich für mich selbst einen `retry`-Dekorator erstellt, den ich einfach auf jede Funktion anwenden kann, und wenn während eines Aufrufs eine Ausnahme ausgelöst wird, wird der Aufruf erneut ausgeführt – bis zu maximal 5 Mal und mit einer Verzögerung zwischen jedem Versuch. Dies ist besonders nützlich für Situationen, in denen Sie versuchen, einen Netzwerkaufruf zu einem entfernten Computer durchzuführen:

Beispiel more_decorator_de.py

Quellcode

```
#from time import sleep
import time
#from functools import wraps
import functools
import logging
logging.basicConfig()
log = logging.getLogger("retry")

# retry: probier's nochmal
# *args: beliebig viele argumente
# **kwargs: beliebig viele keyword-arguments

def retry(f):
    @functools.wraps(f)
    def wrapper_function(*args, **kwargs):
        MAXIMALE_VERSUCHE = 5
        for versuch in range(1, MAXIMALE_VERSUCHE + 1):
            try:
                return f(*args, **kwargs)
            except Exception:
                log.exception("Versuch {} von {} schlug fehl: {}".format(
                              versuch,
                              MAXIMALE_VERSUCHE,
                              (args, kwargs)))
                time.sleep(10 * versuch)
```

```
        log.critical("Alle {} versuche schlugen fehl: {}",
                     MAXIMALE_VERSUCHE,
                     (args, kwargs))
    return wrapper_function


zähler = 0


@retry
def save_to_database(arg):
    print("Schreibt in eine Datenbank, überträgt Daten per Netzwerk etc.")
    print("Die Funktion wird automatisch nocheinmal ausgeführt wennn")
    print("eine Exception auftritt")
    global zähler
    zähler += 1
    print("Dies ist Versuch", zähler)
    # Beim 1. Versuch tritt eine Exception auf, beim
    # 2. Versuch wird die Funktion ausgeführt
    if zähler < 2:
        raise ValueError(arg)
    print("Funktion erfolgreich ausgeführt")


if __name__ == '__main__':
    save_to_database("Irgendein schlechter Wert")
```

Die Zeilennummern sind nicht Bestandteil des Quellcodes

Ausgabe

```
$ python3 more_decorator_de.py
Schreibt in eine Datenbank, überträgt Daten per Netzwerk etc.
Die Funktion wird automatisch nocheinmal ausgeführt wennn
eine Exception auftritt
Dies ist Versuch 1
ERROR:retry:Versuch 1 von 5 schlug fehl: (('Irgendein schlechter Wert',), {})
Traceback (most recent call last):
  File "/home/horst/code/byte-of-python_deutsch_horst/programs/more_decorator_de.py",␣
↪line 19, in wrapper_function
    return f(*args, **kwargs)
           ^^^^^^^^^^^^^^^^^^
  File "/home/horst/code/byte-of-python_deutsch_horst/programs/more_decorator_de.py",␣
↪line 46, in save_to_database
    raise ValueError(arg)
ValueError: Irgendein schlechter Wert
Schreibt in eine Datenbank, überträgt Daten per Netzwerk etc.
Die Funktion wird automatisch nocheinmal ausgeführt wennn
eine Exception auftritt
Dies ist Versuch 2
Funktion erfolgreich ausgeführt
```

Wie es funktioniert:

Siehe auch

- Video : Python Decorators Made Easy
- http://toumorokoshi.github.io/dry-principles-through-python-decorators.html

18.9 Unterschiede zwischen Python 2 und Python 3

Siehe auch

- Porting to Python 3 Redux by Armin
- Python 3 experience by PyDanny
- Discussion on What are the advantages to python 3.x?

18.10 Zusammenfassung

Wir haben in diesem Kapitel einige weitere Merkmale von Python behandelt, und trotzdem haben wir noch nicht alle Merkmale von Python abgedeckt. Allerdings haben wir an diesem Punkt das meiste behandelt, was Sie in der Praxis jemals verwenden werden. Dies ist ausreichend, damit Sie mit den Programmen beginnen können, die Sie erstellen möchten.

Als Nächstes werden wir besprechen, wie man Python weiter erkunden kann.

CHAPTER 19

Wie geht's weiter

Wenn du dieses Buch bis jetzt gründlich gelesen und viele Programme geschrieben hast, dann musst du inzwischen mit Python vertraut und vertraulich geworden sein. Du hast wahrscheinlich einige Python-Programme erstellt, um Dinge auszuprobieren und deine Python-Fähigkeiten zu üben. Falls du das noch nicht getan hast, solltest du es tun. Die Frage ist jetzt: „Was kommt als Nächstes?".

Ich würde vorschlagen, dass du dieses Problem angehst:

> Erstelle dein eigenes Kommandozeilen-Adressbuch-Programm (command_line address book), mit dem du deine Kontakte wie Freunde, Familie und Kollegen sowie deren Informationen wie E-Mail-Adresse und/oder Telefonnummer durchsuchen, hinzufügen, verändern, löschen oder durchsuchen kannst. Die Daten müssen für eine spätere Wiederverwendung gespeichert werden.

Das ist ziemlich einfach, wenn du in Bezug auf all die verschiedenen Dinge denkst, denen wir bisher begegnet sind. Wenn du dennoch eine Anleitung möchtest, wie du fortfahren sollst, dann hier ein Hinweis[1].

Wenn du in der Lage bist, dies zu tun, kannst du behaupten, ein Python-Programmierer zu sein. Jetzt, sofort, schreib dem Autor Swaroop Chitlur und bedanke dich für dieses großartige Buch . Dieser Schritt ist optional, aber empfohlen.

Man kann das Buch übrigens auch kaufen: https://store.pothi.com/book/swaroop-c-h-byte-python/

19.1 Nächste Projekte

Wenn du die obigen Programme einfach zu erstellen fandest, dann sieh dir diese umfassende Liste von Projekten an und versuche, deine eigenen Programme zu schreiben: https://github.com/thekarangoel/Projects#numbers (die gleiche Liste befindet sich auch in Martyr2's Mega Project List).

[1] Erstelle eine Klasse, um die Informationen einer Person darzustellen. Verwende ein Dictionary, um Personenobjekte mit ihrem Namen als Schlüssel zu speichern. Verwende das pickle-Modul, um die Objekte dauerhaft auf deiner Festplatte zu speichern. Verwende die eingebauten Methoden des Dictionary, um Personen hinzuzufügen, zu löschen und zu verändern.

19.2 Beispielcode

Der beste Weg, eine Programmiersprache zu lernen, ist, viel Code zu schreiben und viel Code zu lesen:

- Python Cookbook ist eine äußerst wertvolle Sammlung von Rezepten oder Tipps, wie man bestimmte Arten von Problemen mit Python löst. Dies ist ein Pflichttext für jeden Python-Benutzer.
- Python Module of the Week ist ein weiterer ausgezeichnete, unbedingt lesenswerter Leitfaden zur *Standard Library*.

(mehr Links auf https://python.swaroopch.com/what_next.html)

19.3 Zusammenfassung

Wir sind nun am Ende dieses Buches angekommen, aber wie man so sagt, dies ist der Anfang vom Ende! Du bist jetzt ein begeisterter Python-Benutzer und zweifellos bereit, viele Probleme mithilfe von Python zu lösen. Du kannst anfangen, deinen Computer zu automatisieren, um alle möglichen früher unvorstellbaren Dinge zu tun oder deine eigenen Spiele zu schreiben und vieles, vieles mehr. Also, fang an!

CHAPTER 20

Quizfragen

20.1 Aufgaben für das Kapitel *Basic*:

20.1.1 Aufgabe basic 1

- Erstelle eine Variable mit dem Namen `sentence` und weise ihr den Wert eines Strings mit elf Wörtern zu (beliebige Wörter).
- Gib (den Wert der Variable) `sentence` aus.
- Füge diese Codezeile am Ende deines Programms hinzu:

```
print(len(sentence.split()))
```

Hinweis: Die Python-`split()`-Funktion zählt die Wörter in einem String, indem sie die Leerzeichen zwischen den Wörtern zählt, daher lautet die genaue Aufgabe: „Schreibe 11 Wörter mit 10 Leerzeichen zwischen ihnen“.

mögliche Lösung

```
# solution for basic 1
sentence = "Fischers Fritz fischt frische Fische. Frische Fische fischt Fischer's Fritz."
print(sentence)
print(len(sentence.split()))
```

20.1.2 Aufgabe basic 2

- Erstelle eine Variable mit dem Namen `gedicht` und weise ihr einen String mit einigen Wörtern zu.
- Der String soll über drei Textzeilen gehen.
- Gib den Wert der Variable poem aus.
- Füge diese Codezeile am Ende deines Programms hinzu:

```
print(len(gedicht.splitlines()))
```

Können Sie diese Aufgabe (dass der String über mehrere Zeilen geht) auf verschiedene Arten lösen?

mögliche Lösungen

Variante A

```
# solution for chapter basic,  task 2, variant A
gedicht = """Eins
Zwo
g'suffa!"""
print(gedicht)
print(len(gedicht.splitlines()))
```

Variante B

```
# solution for chapter basic, task 2, variant B
gedicht="Eins\nZwo\ng'suffa!"
print(gedicht)
print(len(gedicht.splitlines()))
```

Variante C

```
# solution for chapter basic,  task 2, variant C
poem="\n".join(["Eins","Zwo","g'suffa!"])
print(poem)
print(len(gedicht.splitlines()))
```

20.1.3 Aufgabe basic 3

Erstelle eine Variable mit dem Namen `gehalt` und weise ihr den Wert `4000` zu. Gib den Wert von `salary` aus.

mögliche Lösungen

Variante A

```
# solution for chapter basic,  task 3, variant A
# most common variant
gehalt = 4000
print(gehalt)
```

Variante B

```
# solution for chapter basic,  task 3, variant B
# use underscores for better readability
gehalt = 4_000
print(gehalt)
```

Variante C

```
# solution for chapter basic, task 3, variant C
print(gehalt:=4000) # walrus operator,
# siehe <https://peps.python.org/pep-0572/>
```

20.1.4 Aufgabe basic 4

- Erstelle eine Variable mit dem Namen `gehalt` und weise ihr den Wert `4000` zu.
- Erstelle eine Variable namens `output`. Der Wert von `output` soll dieser String sein: `"Mein Gehalt ist X Euro pro Monat"`.
- Modifiziere den Code so, dass Python `X` durch den Wert der Variable `gehalt` ersetzt.
- Gib den Wert von `gehalt` aus.

Können Sie diese Aufgabe auf verschiedene Arten lösen?

mögliche Lösungen:

mögliche Lösungen

Variante A

```
# solution for chapter basic,  task 4, variant A
# using .format()
gehalt=4000
output = 'Mein Gehalt ist {} Euro pro Monat'.format(gehalt)
print(output)
```

Variante B

```
# solution for chapter basic,  task 4, variant B
# using f-strings
gehalt = 4_000 # Unterstriche verbessern die Lesbarkeit
output = f'Mein Gehalt ist {gehalt} Euro pro Monat'
print(output)
```

Variante C

```
# solution for chapter basic,  task 4, variant C
# using str()
gehalt=4000
output = "Mein Gehalt ist " + str(gehalt) + " Euro pro Monat"
print(output)
```

Variante D

```
# solution for chapter basic,  task 4, variant D
# using %
gehalt=4000
output = 'Mein Gehalt ist %i Euro pro Monat' % income
print(output)
```

20.1.5 Aufgabe basic 5

Bezeichner für Variablen

-Welche dieser Namen sind gültige Bezeichner (namen) für Variablen in Python?

1) `a`
2) `A`
3) `aaaa`
4) `a123`
5) `123a`
6) `_123a`
7) `_a123`
8) `a_123`
9) `1_a23`
10) `!abc`
11) `a-b`
12) `a_minus_b`

Korrekte Antworten

1, 2, 3, 4, 6, 7, 8, 12

20.2 Aufgaben für das Kapitel *Operatoren und Ausdrücke*

20.2.1 Aufgabe op_exp 1

- Welche dieser Python-Ausdrücke ergeben den Wert True?

1. `True == False`
2. `True == True`
3. `False == False`
4. `5 > 2`
5. `len("Michael") > len("Mike")`
6. `5 != 7`
7. `6 >= 6`
8. `"abc" * 3 == "abcabcabc"`
9. `5**2 == 25`
10. `0 == False`
11. `1 == True`
12. `2 == True`
13. `2 == False`
14. `True == 1`

15. `None == None`
16. `None != 0`
17. `None == ""`

Korrekte Antworten

2,3,4,5,6,7,8,9,10,11,14,15,16

20.2.2 Aufgabe op_exp 2

Welche der folgenden Python-Ausdrücke ergeben den Wert False?

1. `10 / 5 == 2.0`
2. `10 // 5 == 2`
3. `10 % 3 == 1`
4. `( 5 > 1) und ( 5 > 7)`
5. `( 5 > 1) oder (5 > 7)`
6. `not (5 > 7)`

Korrekte Antwort(en)

4

20.2.3 Aufgabe op_exp 3

Was wird die Ausgabe (der Wert der Variable `x`) dieses Python-Programms sein?

```
x = 5
x = 5+1
x += 1
x = x * 2
x /= 2
print(x)
```

Korrekte Antwort

7.0

20.3 Aufgaben für das Kapitel Kontrollfluss

20.3.1 Aufgabe control_flow 1

In den Aufgaben für dieses Kapitel verwende (einige der) Python-Anweisungen, die im Kapitel *Kontrollfluss* beschrieben sind (wie `if`, `elif`, `else`, `for`, `while`, `continue`, `break`).

- Schreibe ein Programm, das den Benutzer ein Passwort eingeben lässt und eine Antwort über das Passwort gibt:
- Wenn das Passwort SeCrEt ist, soll das Programm `Correct` ausgeben und beenden.
- Wenn der Benutzer ein falsches Passwort eingibt, soll das Programm `Wrong` ausgeben und erneut nach einem Passwort fragen.
- Nach 3 fehlgeschlagenen Versuchen soll das Programm `You failed 3 times` ausgeben und beenden.

- Bevor das Programm endet, soll es `bye!` ausgeben.

Beispielausgabe:

```
Please enter password: >>>secret
Wrong
Please enter password: >>>Secret
Wrong
Please enter password: >>>SeCrEt
Correct
bye!
```

mögliche Lösungen

Variante A

```
# solution for chapter control flow,  task 1, variant A
for a in range(3):
    text = input("Please enter password: >>>")
    if text == "SeCrEt":
        print("Correct")
        break
    else:
        print("Wrong")
else:
    print("You failed 3 times")
print("bye!")
```

Variante B

```
# solution for chapter control flow,  task 1, variant B
for _ in range(3):
    if input("Please enter password: >>>") == "SeCrEt":
        print("Correct")
        break
    print("Wrong")
else:
    print("You failed 3 times")
print("bye!")
```

Variante C

```
# solution for chapter control flow,  task 1, variant C
attempt = 1
max_attempts = 3
valid_password = "SeCrEt"
while True:
    print(f"this is attempt {attempt} of {max_attempts}")
    text = input("Please enter password: >>>")
    if text == valid_password:
        print("Correct")
        break
    else:
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        print("Wrong")
    attempt += 1
    if attempt > 3:
        print("You failed 3 times")
        break
print("bye!")
```

Variante D

```
# solution for chapter control flow,  task 1, variant D
attempt = 0
while attempt < 3:
    attempt += 1
    if input("Please enter password: >>>") == "SeCrEt":
        print("Correct")
        break
    print("Wrong")
else:
    print("You failed 3 times")
print("bye!")
```

Variante E

```
# solution for chapter control flow,  task 1, variant E
attempt = 1
while input("Please enter password: >>>") != "SeCrEt":
    print("wrong")
    attempt += 1
    if attempt > 3:
        print("You failed 3 times")
        break
else:
    print("correct")
print("bye!")
```

20.3.2 Aufgabe control_flow 2

Das folgende Programm funktioniert **nicht** wie beabsichtigt.

Was das Programm eigentlich tun sollte:

- Das Programm soll den Benutzer ein Passwort eingeben lassen.
- Wenn der Benutzer das richtige Passwort (secret) eingibt, soll das Programm correct ausgeben und beenden.
- Wenn der Benutzer ein falsches Passwort eingibt, soll das Programm erneut fragen.
- Wenn der Benutzer 3-mal ein falsches Passwort eingibt, soll das Programm `You failed 3 times` ausgeben und beenden.

Deine Aufgaben nach der Analyse des Programms:

- Finde heraus, warum dieses Programm nicht wie beabsichtigt funktioniert.
- Schlage vor, wie man das Programm ändern kann, damit es wie gewünscht funktioniert.

Hier der Quellcode des nicht funktionierenden Programms:

```
# problem  control flow,  task 2,
password = "secret"
max_attempts = 3
attempt = 1
while attempt < max_attempts:
    print(f"Attempt {attempt} of {max_attempts}")
    guess = input("enter password: >>>")
    if guess == password:
        print("correct")
        break
    attempt += 1
else:
    print("You failed 3 times")
print("bye")
```

mögliche Lösung

Das Problem liegt in Zeile 5.

Korrektur von Zeile 5:

```
while attempt <= max_attempts:
```

20.3.3 Aufgabe control_flow 3

Bitte korrigiere das folgende Programm so, dass es nur eine einzige Antwort ausgibt.

```
# problem control_flow task 3
income = input("please enter your monthly (netto) income in €")
income = int(income)
if income < 1000:
    print("you do not earn much...")
if income < 2000:
    print("it could be better....")
elif income < 3000:
    print("nice")
if income < 4000:
    print("very nice")
if income < 5000:
    print("fantastic")
else:
    print("really??")
```

mögliche Lösung

```
income = input("please enter your monthly (netto) income in €")
income = int(income)
if income < 1000:
    print("you do not earn much...")
elif income < 2000:
    print("it could be better....")
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
elif income < 3000:
    print("nice")
elif income < 4000:
    print("very nice")
elif income < 5000:
    print("fantastic")
else:
    print("really??")
```

20.3.4 Aufgabe control flow 4

Übungen zu `for`-Schleife und zum Befehl `range`

Lies die Python-Dokumentation über den `range-Befehl`: https://docs.python.org/3/library/stdtypes.html#range um die Parameter `start`, `stop` und `step` zu verstehen. (oder tippe im Python-Interpreter `help(range)` ein)

Versuche diese Aufgabe im Kopf (ohne Computer) zu lösen: Was wird die Ausgabe dieser Python-Anweisung sein?

```
print(list(range(5)))
```

Lösung

Die Ausgabe wird sein: `[0,1,2,3,4]`

20.3.5 Aufgabe control flow 5

Schreibe eine Python-Anweisung (mit `range`), die folgende Ausgabe erzeugt:

`[1,2,3,4,5]`

Lösung

```
# solution control flow, task 5
print(list(range(1,6)))
```

20.3.6 Aufgabe control_flow 6

Schreibe eine Python-Anweisung (mit `range`), die folgende Ausgabe erzeugt:

`[10,20,30,40,50]`

Lösung

```
# solution control flow task 6
print(list(range(10,51,10)))
```

20.3.7 Aufgabe control_flow 7

Schreibe eine Python-Anweisung (mit `range`), die folgende Ausgabe erzeugt:

`[50,40,30,20,10,0]`

Lösung

```
# solution control flow task 7
print(list(range(50,-1,-10)))
```

20.3.8 Aufgabe control_flow 8

- Schreibe eine Python-Anweisung (mit `range`), die alle Zahlen von 1 bis (inklusive) 10 ausgibt.
- Jede Zahl soll in einer eigenen Zeile stehen.

Lösung

```
# solution control flow task 8
for x in range(1,11):
    print(x)
```

20.3.9 Aufgabe control_flow 9

- Schreibe eine Python-Anweisung (mit `range`), die alle Zahlen von 1 bis inklusive 10 in einer einzigen Zeile ausgibt.
- Die Zahlen sollen durch ein Komma getrennt sein. (Es darf ein abschließendes Komma vorhanden sein.)

Lösung

```
# solution control flow task 9
for x in range(1,11):
    print(x, end=",")
```

20.3.10 Aufgabe control_flow 10

- Schreibe ein Python-Programm (mit range), das jede Zahl von 2 bis inklusive 5 mit jeder anderen Zahl in diesem Bereich multipliziert.
- Das Programm soll für jede Berechnung eine eigene Zeile ausgeben, wie in diesem (gekürzten) Beispiel:

(*Die 3 Punkte (...) symbolisieren daß nur ein Ausschnitt gezeigt wird*)

```
...
 2 x 2 =  4
 2 x 3 =  6
 2 x 4 =  8
 2 x 5 = 10
 3 x 2 =  6
 3 x 3 =  9
 ...
```

Lösung

```
# solution chapter control flow,  task 10
for a in range(1,6):
    for b in range(1,6):
        print(f"{a} x {b} = {a*b:>2}")
```

20.3.11 Aufgabe control_flow 11

- Gegeben ist dieses Python-Programm:

```
# problem control flow task 11
for x in "abcde":
    for y in "wxyz":
        print(x,y)
```

- Wie viele Zeilen wird dieses Python-Programm ausgeben?

Lösung

20 Zeilen

20.3.12 Aufgabe control_flow 12

- Gegeben ist dieses Python-Programm:

```
# problem control flow task 12
for a in ("abc","def","ghi"):
    for b in (100,200,300,400):
        for c in "yz":
            print(a,b,c)
```

Wie viele Zeilen wird dieses Python-Programm ausgeben?

Lösung

24 Zeilen

20.3.13 Aufgabe control_flow 13

- Gegeben ist dieses Python-Programm:

```
# problem control flow task 13
for a in range(-10,11):
    for b in range(-10,11):
        print(f"{a} + {b} = {a+b}")
        print(f"{a} - {b} = {a-b}")
        print(f"{a} x {b} = {a*b}")
        print(f"{a} / {b} = {a/b}")
```

- Warum funktioniert dieses Programm nicht?

Lösung

Wegen eines Division-durch-Null-Fehlers in Zeile 6. Die Zeile:

```
print(f"{a} / {b} = {a/b}")
```

versucht, `a` durch `b` zu teilen. Aber `b` stammt aus `range(-10, 11)`, und dieser Bereich enthält auch die 0.

20.3.14 Aufgabe control_flow 14

-Bitte lies in im Kapitel *Kontrollstrukturen* über die Befehle `break` und `continue` nach. (Beide Befehle können innerhalb einer `while`-Schleife oder innerhalb einer `for`-Schleife verwendet werden.) -Überprüfe außerdem, ob du die offizielle Python-Dokumentation dazu verstehst: https://docs.python.org/3/tutorial/controlflow.html#break-and-continue-statements

- Wenn du das Gefühl hast, `break` und `continue` verstanden zu haben, analysiere folgends (korrekt funktionierende) Programm:

```
# problem control flow task 14 A
while True:
    print("What is the answer to THE question?")
    print("type 'help' to see a help text")
    print("type 'quit' to exit this game")
    command = input(">>>")
    if command == "help":
        print("See 'The hitchhikers guide to the galaxy")
        print(" by Douglas Adams")
    elif command == "quit":
        break
    elif command == "42":
        print("Congratulations, you know THE answer")
        print("But what was the question exactly..... ? ")
        break

print("bye-bye")
```

- Versuche zu verstehen, was das Programm macht (probiere es aus).
- Versuche, dieses Programm nicht nur anhand des Codes zu verstehen, sondern auch anhand des Flussdiagramms:

Du kannst Flussdiagramme mit Stift und Papier oder mit Computerprogrammen erstellen (z. B. MS Word oder LibreOffice Draw). Das obige Diagramm wurde mit dem Charting-Tool „mermaid“ erstellt: https://mermaid.js.org

- Gegeben ist dieses Python-Programm:

```
# problem control flow task 14 B
# password creation

while True:
    print("type 'help' to display a help text")
    command = input("please enter a new password >>>")
    if command == "help":
        print("The password must have: ")
        print("At least one digit (0-9) ")
        print("At least one lower-case character (a-z)")
        continue
    # test the password

    for char in "abcdefghijklmnopqrstuvwxyz":
        if char in command:
            break
    else:
        print("no lowercase character (a-z) found. please try again")
        continue

    for number in "0123456789":
        if number in command:
            break
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    else:
        print("no digit (0-9) found. please try again")
        continue
    print("Congratulation, your password is accepted")
    break
print("bye bye")
```

- Analysiere das obige Python-Programm und erstelle dafür ein Flussdiagramm! (mit einem Werkzeug deiner Wahl)

mögliche Lösung

20.4 Aufgaben für das Kapitel *Funktionen*

20.4.1 Aufgabe functions 1

- Schreibe eine Python-Funktion mit dem Namen `greeting`.
- Die Funktion soll einen Hotelangestellten simulieren. Wenn die Funktion die Parameter `hour_of_day` (0–24) erhält, soll sie entsprechend dieser Tabelle einen Gruß zurückgeben:

Stunde von	Stunde bis	Gruß
0	6	Gute Nacht
6	11	Guten Morgen
11	14	Guten Tag
14	18	Guten Nachmittag
18	22	Guten Abend
22	24	Gute Nacht

- Füge Code hinzu, um die Funktion `greeting` zu testen:
  - Gib jede Stunde von 1 bis 24 und den entsprechenden Gruß für diese Stunde aus (eine Zeile pro Stunde)

mögliche Lösungen

Variante A

```
# solution chapter function task 1, variant A
def greeting(time_of_day):
    if 6 <= time_of_day <11:
        return "Good morning"
    elif 11 <= time_of_day < 14:
        return "Good day"
    elif 14 <= time_of_day < 18:
        return "Good afternoon"
    elif 18 <= time_of_day < 22:
        return "Good evening"
    elif (22 <= time_of_day) or (time_of_day <6):
        return "Good night"
# test
for h in range(1,25):
    print(f"hour: {h:>2} greeting: {greeting(h)}")
```

Variante B

```
# solution chapter function task 1, variant B
def greeting(hour_of_day):
    #  dictionary: key: hour_of_day value: greeting
    timetable = {6:"Good night",
                 11:"Good morning",
                 14:"Good day",
                 18:"Good afternoon",
                 22:"Good evening",
                 24.1:"Good night", # special case to catch 24
                }
    for key in timetable:
        if hour_of_day < key:
            return timetable[key]

# test
for h in range(1,25):
    print(f"hour: {h:>2} greeting: {greeting(h)}")
```

20.4.2 Aufgabe control flow 2

- Schreibe eine Funktion namens `improved_greeting`.
- Die Funktion soll wie in *Aufgabe control flow 1* einen Hotelangestellten simulieren.
- Die Funktion soll zwei Parameter haben:
  - `hour_of_day` (eine Zahl von 1 bis inklusive 24)
  - `gender` (kann die Werte `"male"` oder `"female"` haben)
- Die Funktion soll je nach `hour_of_day` einen Gruß zurückgeben (siehe Tabelle in *Aufgabe control flow 1*), allerdings abhängig vom Wert des Arguments `gender`:
  - füge `"Sir"` zum Gruß hinzu, wenn `gender`den Wert `"male"` hat
  - füge `"Madam"` zum Gruß hinzu, wenn das `gender` den Wert `"female"` hat.

Beispiele:

```
Good morning, Sir
Good afternoon, Madam
```

-Füge außerdem – wie in der vorherigen Aufgabe – Code hinzu, um die Funktion für jede Stunde des Tages (0–24) und für beide Geschlechter („male“ und „female“) zu testen.

mögliche Lösungen

Variante A

```
# solution chapter function task 2, variant A
def improved_greeting(hour_of_day, gender):
    suffix = ""
    if gender == "male":
        suffix = ", Sir"
    elif gender == "female":
        suffix = ", Madam"
    if 6 <= hour_of_day <11:
        return "Good morning" + suffix
    elif 11 <= hour_of_day < 14:
        return "Good day" + suffix
    elif 14 <= hour_of_day < 18:
        return "Good afternoon" + suffix
    elif 18 <= hour_of_day < 22:
        return "Good evening" + suffix
    elif (22 <= hour_of_day) or (hour_of_day <6):
        return "Good night" + suffix
# test
for h in range(1,25):
    for g in ("male", "female"):
        print(f"hour: {h:>2} gender: {g:<6} greeting: {improved_greeting(h,g)}")
```

Variante B

```
# solution chapter function task 1, variant B
def improved_greeting(hour_of_day, gender):
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    suffix = {"male": "Sir",
              "female":"Madam",
              }

    #  dictionary: key: hour_of_day value: greeting
    timetable = {6:"Good night",
                 11:"Good morning",
                 14:"Good day",
                 18:"Good afternoon",
                 22:"Good evening",
                 24.1:"Good night", # special case to catch 24
                }
    for key in timetable:
        if hour_of_day < key:
            return timetable[key] + ", " + suffix[gender]
# test
for h in range(1,25):
    for g in ("male", "female"):
        print(f"hour: {h:>2} gender: {g:<6} greeting: {improved_greeting(h,g)}")
```

20.4.3 Aufgabe control flow 3

- Benutze das Beispiel aus der vorherigen Aufgabe (*Aufgabe control flow 2*), aber mache folgende Änderungen:
  - Benenne die Funktion `improved_greeting` in `complex_greeting` um
  - Füge einen zusätzlichen Parameter mit dem Namen `child` hinzu (kann `True` oder `False` sein)
  - Ändere die Rückgabewerte der Funktion so, dass – wenn `child` den Wert `True` hat – die Funktion anstatt `"Sir"` den Text `"young man"` zurückgibt und anstatt `"Madam"` den Text `"young lady"`.
- Füge Code hinzu, um die Funktion für alle Kombinationen von `hour_of_day` (0–24), `gender` (`"male"`, `"female"`) und `child` (`True`, `False`) zu testen

mögliche Lösungen

Variante A

```
# solution chapter function task 3, variant A
def complex_greeting(hour_of_day, gender, child):
    suffix = ""
    if gender == "male":
        suffix = ", Sir"
        if child: #  if child == True:
            suffix = ", young man"
    elif gender == "female":
        suffix = ", Madam"
        if child:
            suffix = ", young lady"
    if 6 <= hour_of_day <11:
        return "Good morning" + suffix
    elif 11 <= hour_of_day < 14:
        return "Good day" + suffix
    elif 14 <= hour_of_day < 18:
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        return "Good afternoon" + suffix
    elif 18 <= hour_of_day < 22:
        return "Good evening" + suffix
    elif (22 <= hour_of_day) or (hour_of_day <6):
        return "Good night" + suffix
# test
for h in range(1,25):
    for g in ("male", "female"):
        for c in (True, False):
            print(f"hour: {h:>2} gender: {g:<6} child: {str(c):<5} "
                  f"greeting: {complex_greeting(h,g,c)}")
```

Variante B

```
# solution chapter function task 3, variant B
def complex_greeting(hour_of_day, gender, child):

    # dictionary: key: gender value: (greeting_adult, greeting_child)
    suffix = {"male": ("Sir","young man"),
              "female":("Madam","young lady"),
              }

    #  dictionary: key: hour_of_day value: greeting
    timetable = {6:"Good night",
                 11:"Good morning",
                 14:"Good day",
                 18:"Good afternoon",
                 22:"Good evening",
                 24.1:"Good night", # special case to catch 24
                }
    for key in timetable:
        if hour_of_day < key:
            return timetable[key] + ", " + suffix[gender][child]
            # True has the value of 1, False has the value of 0
            # Therefore, child can be used as index for first/second element
# test
for h in range(1,25):
    for g in ("male", "female"):
        for c in (True, False):
            print(f"hour: {h:>2} gender: {g:<6} child: {str(c):<5} "
                  f"greeting: {complex_greeting(h,g,c)}")
```

20.4.4 Aufgabe control flow 4

- Schreibe eine Funktion mit dem Namen `greeter1`:
- Die Funktion soll keinerlei Parameter haben
- Die Funktion soll immer den String `"Good morning"` zurückgeben
- Füge Code hinzu, der das Ergebnis eines Funktionsaufrufs von `greeter1` ausgibt

Lösung

```
# solution chapter function, task4
def greeter1():
    return "Good morning"

# function call (calling greeter1 without arguments)
print("----- calling greeter1 -------")
print(greeter1())
```

20.4.5 Aufgabe control flow 5

-Erstelle eine Funktion mit dem Namen `greeter2`:

- Die Funktion soll einen Parameter mit dem Namen `adjective` haben
- Der Standardwert von `adjective` soll `"good"` sein
- Die Funktion soll einen String zurückgeben, bestehend aus `adjective` und dem Text `" Morning"`
- Teste die Funktion, indem du sie mit verschiedenen Argumenten (und ohne Argumente) aufrufst. Gib jeweils den Rückgabewert aus.

Lösung

```
# solution function task 5
def greeter2(adjective="good"):
    return adjective + " Morning"
#function call (calling greeter2 with different arguments)
print("---- calling greeter2 -----")
print(greeter2("bad"))
print(greeter2("excellent"))
print(greeter2())
print(greeter2(" "))
```

20.4.6 Aufgabe control flow 6

-Schreibe eine Funktion mit dem Namen `greeter3`:

- Die Funktion soll 2 Parameter haben: `adjective` und `time_of_day` (beide sind Strings)
- Beide Parameter sollen Standardwerte haben (z. B. `"good"` und `"morning"`)
- Die Funktion soll einen einzigen String zurückgeben, bestehend aus dem Wert von `adjective`, einem Leerzeichen und dem Wert von `time_of_day` -Teste die Funktion, indem du sie mit verschiedenen Argumenten (und ohne Argumente) für beide Parameter aufrufst und jeweils den Rückgabewert ausgibst

mögliche Lösung

```
#solution function task 6
def greeter3(adjective="good", time_of_day="Morning"):
    return adjective + " " + time_of_day
#function call calling greeter3
print("---- calling greeter3 -----")
print(greeter3())
print(greeter3("bad"))
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
print(greeter3("bad", "evening"))
print(greeter3(time_of_day= "evening"))
```

20.4.7 Aufgabe control flow 7

- Schreibe eine Funktion mit dem Namen `greeter4`:
- Die Funktion soll einen Parameter mit dem Namen `time_of_day` haben
- Der Standardwert von `time_of_day` soll `"morning"` sein -Die Funktion soll BELIEBIG viele weitere Argumente akzeptieren (auch keine Argumente) -Die Funktion soll einen String zurückgeben, bestehend aus allen zusätzlichen Argumenten (durch Komma getrennt), einem Leerzeichen und dem Wert von `time_of_day` (alle Parameter sind Strings) -Teste die Funktion, indem du sie mehrfach aufrufst – jedes Mal mit einer unterschiedlichen Anzahl an Argumenten (und auch ohne Argumente). Gib bei jedem Funktionsaufruf den Rückgabewert aus.

Beispiel:

```
print(greeter4("evening", "lovely", "mild", "wonderful"))
```

mögliche Lösungen

Variante A

```
# solution functions task 7 variant A
# function that accepts any number of parameters and returns them all
def greeter4(time_of_day="Morning", *args):
    text = ""
    for a in args:
        text += a + ","
    if len(args) > 0:
        text = text[:-1]  # remove last comma
        text += " "
    text += time_of_day
    return text

print("------ calling greeter4 ----")
print(greeter4())
print(greeter4("night"))
print(greeter4("evening", "wonderful", "lovely", "heroic", "romantic"))
print(greeter4("night", "good"))
print(greeter4("day", "sunny", "warm", "emotional"))
```

Variante B

```
# solution functions task 7 variant B
# function that accepts any number of parameters and returns them all
def greeter4(time_of_day="Morning", *args):
    text = ",".join(args)
    if len(text) > 0:
        text += " "
    text += time_of_day
    return text
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
print("------ calling greeter4 ----")
print(greeter4())
print(greeter4("night"))
print(greeter4("evening", "wonderful", "lovely", "heroic", "romantic"))
print(greeter4("night", "good"))
print(greeter4("day", "sunny", "warm", "emotional"))
```

20.4.8 Aufgabe control flow 8

-Schreibe eine Funktion mit dem Namen `greeter5`:

- Die Funktion soll einen Parameter mit dem Namen `time_of_day` haben
- Der Standardwert von `time_of_day` soll "morning" sein
- Die Funktion soll BELIEBIG viele zusätzliche Keyword-Argumente akzeptieren, z. B.:

```
greeter5("morning", air="wonderful", weather="sunny")
```

- Die Funktion soll einen mehrzeiligen String zurückgeben (siehe unten), der alle Argumente in folgender Form enthält:

Funktionsaufruf:

```
greeter5("morning", air="wonderful", weather="sunny")
```

Rückgabewert:

```
"What a morning!\nThe air is wonderful.\nThe weather is sunny."
```

-Teste die Funktion indem du sie mehrfach mit unterschiedlichen Argumenten aufrufst (auch ohne Argumente). Gib jeweils den Rückgabewert aus.

mögliche Lösung

```
#solution function task 8
def greeter5(time_of_day="morning", **kwargs):
    text = "What a " + time_of_day + "!\n"  # \n makes a new line
    for key, value in kwargs.items():
        text += "The " + key + " is " + value + ".\n"
    return text
print("------- calling greeter5 ---------")
print(greeter5())
print(greeter5(air="wonderful", mood="joyful", future="bright"))
print(greeter5("day", temperature="freezing", wind="strong"))
```

20.4.9 Aufgabe control flow 9

- Schreibe eine Funktion mit dem Namen `greeter6`:
- Die Funktion soll einen Parameter mit dem Namen `time_of_day` haben Der Standardwert von `time_of_day` soll `"morning"` sein -Die Funktion soll BELIEBIG viele zusätzliche Argumente akzeptieren (deren Werte Strings sind), z. B.:

```
greeter6("day","good","early","successfull")
```

- Die Funktion soll BELIEBIG viele zusätzliche Keyword-Argumente akzeptieren, z. B.:

```
greeter6("day", "good", air="wonderful", weather="sunny")
```

- Die Funktion soll einen mehrzeiligen String zurückgeben, der alle Argumente und alle Keyword-Arguments in dieser Form enthält:

Funktionsaufruf:

```
greeter6("morning", "good", "nice", air="wonderful", weather="sunny")
```

Rückgabewert:

```
"What a good, nice morning!\nThe air is wonderful.\nThe weather is sunny."
```

-Teste die Funktion, indem du sie mehrfach mit verschiedenen Argumenten und Keyword-Argumenten aufrufst, und gib jeweils den Rückgabewert aus.

mögliche Lösung

```
# solution function task 9
def greeter6(time_of_day="morning", *args, **kwargs):
    text = "What a "
    # iterate over *args
    for a in args:
        text += a + ", "
    if len(args) > 0:
        text = text[:-2] + " "
    text += time_of_day + "!\n"
    # iterate over **kwargs
    for key,value in kwargs.items():
        text += "The " + key + " is " + value + ".\n"
    return text

print("------ calling greeter6 ----")
print(greeter6())
print(greeter6("evening"))
print(greeter6("morning", "sunny", "warm", "emotional"))
print (greeter6("morning", "sunny", "warm", "emotional",
                  air="wonderful", mood="joyful", future="bright"))
print(greeter6(air="smelly"))
```

20.5 Aufgaben für das Kapitel *Objektorientierte Programmierung*

20.5.1 Aufgabe oop 1

- Schreibe eine Klasse mit dem Namen `Game`.
  - Die Klasse soll eine *Klassenvariable* mit dem Namen `player` haben. Der *Wert* dieser Variable soll `"Bugs Bunny"` sein.
  - Die Klasse soll eine *Klassenvariable* mit dem Namen `highscore` haben. Der *Wert* dieser Variable soll `1000` sein.

  - Die Klasse soll eine *Klassenvariable* mit dem Namen `credit` haben. Der *Wert* dieser Variable soll `2` sein.
- Schreibe Python-Code, der alle Klassenvariablen der Klasse `Game` und ihre Werte ausgibt.

mögliche Lösungen

Variante A

```
# Lösung OOP Aufgabe 1 Variante A
class Game:
    player = "Bugs Bunny"
    highscore = 1000
    credit = 2

print("Game.player", Game.player)
print("Game.highscore", Game.highscore)
print("Game.credit", Game.credit)
```

Variante B

```
# Lösung OOP Aufgabe 1 Variante B
class Game:
    player = "Bugs Bunny"
    highscore = 1000
    credit = 2

for key, value in Game.__dict__.items():
    if key[:2] != "__":
        print(key, value)
```

20.5.2 Aufgabe oop 2

- Schreibe eine Klasse mit dem Namen `Toy`
- Die Klasse soll eine `__init__`-Methode besitzen.
- Schreibe die Klasse so, dass jede Instanz dieser Klasse folgende Attribute (auch Objektvariablen genannt) hat:
  - `price` mit dem Wert `10`
  - `color` mit dem Wert `"green"`
- Erstelle eine Variable mit dem Namen `teddy`.
- Der Wert von `teddy` soll eine Instanz der Klasse `Toy` sein.
- Schreibe Code, um das Attribut `height` von teddy auf `7` zu setzen.
- Schreibe Code, um die Namen und Werte aller Attribute von `teddy` auszugeben.

mögliche Lösungen

Variante A

```
# Lösung OOP Aufgabe 2 Variante A
class Toy:
    def __init__(self):
        self.price = 10
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        self.color = "green"

teddy = Toy()
teddy.height = 7
print("price", teddy.price)
print("color", teddy.color)
print("height", teddy.height)
```

Variante B

```
# Lösung OOP Aufgabe 2 Variante B
class Toy:
    def __init__(self):
        self.price = 10
        self.color = "green"

teddy = Toy()
teddy.height = 7
for key, value in teddy.__dict__.items():
    print(key, value)
```

20.5.3 Aufgabe oop 3

- Erstelle eine Klasse mit dem Namen `Walker`.
  - Gib dieser Klasse eine Methode mit dem Namen `walk`.
    - Der Rückgabewert dieser Methode soll der String `"i'm walking"` sein.
- Erstelle eine Klasse mit dem Namen `Swimmer`.
  - Gib dieser Klasse eine Methode mit dem Namen `swim`.
    - Der Rückgabewert dieser Methode soll `"i'm swimming"` sein.
- Erstelle eine Klasse namens `Flyer`.
  - Gib der Klasse eine Methode names `fly`.
    - Mit dem Rückgabewert: `"i'm flying"`
- Erstelle eine Klasse `Diver`. - Mit einer Methode `dive`. - Mit dem Rückgabewert: `"i'm diving"` -Erstelle Klassen mit den Namen und Methoden gemäß der Tabelle unten. Verwende *Vererbung* (*inheritance*) von den bestehenden Klassen für die neuen Klassen. Schreibe nur ein `pass` Statement in die Klassen, schreibe keine Methoden oder Attribute hinein.

Klassenname	walk()	swim()	fly()	dive()
Falcon	Ja	Nein	Ja	Nein
Penguin	Ja	Ja	Nein	Ja
Duck	Ja	Ja	Ja	Ja
Eurasian_swift	Nein	Nein	Ja	Nein

- Erstelle eine Variable `tux`. Der Wert dieser Variablen soll eine *Instanz* der Klasse `Penguin` sein.
- Schreibe Python-Code, der das Ergebnis von `tux.dive()` ausgibt.

mögliche Lösung

```
#Lösung OOP Aufgabe 3 Variante A
#Elternklassen

class Walker:
   def walk(self):
      return "i'm walking"

class Swimmer:
    def swim(self):
        return "i'm swimming"

class Flyer:
    def fly(self):
        return "i'm flying"

class Diver:
    def dive(self):
        return "i'm diving"

# Kindklassen
class Falcon(Walker, Flyer):
    pass

class Penguin(Walker, Swimmer, Diver):
    pass

class Duck(Walker, Swimmer, Flyer, Diver):
    pass

class Eurasian_swift(Flyer):
    pass

tux = Penguin()     # Klasseninstanz erzeugen
print(tux.dive())
```

20.5.4 Aufgabe oop 4

- Folgender Code ist gegeben:

```
class Bird:
    def __init__(self, name):
        self.name = name
        self.can_fly = True
        self.can_walk = True
        self.can_swim = False
        self.can_dive = False

    def __str__(self):
        """Diese Funktion wird aufgerufen, wenn die Klasseninstanz gedruckt wird"""
        return f"i am a {self.__class__.__name__}"
```

- Erweitere obigen Python-Code, um eine Variable tux zu erstellen.

- Der Wert dieser Variable soll eine Klasseninstanz von `Bird` sein. -Das Attribut `name` der Instanz soll `"Duck"` sein.
- Schreibe eine neue Zeile Python-Code, um das Attribut `can_swim` von `tux` auf `True` zu setzen.

mögliche Lösung

```
# ...
# <hier die class Bird einfügen aus der Angabe>
# ...
# Lösung Kapitel OOP, Aufgabe 4
tux = Bird(name="Duck")
tux.can_swim = True
```

20.5.5 Aufgabe oop 5

- Gegeben ist die Klasse `Bird` aus *Aufgabe oop 4*.
- Schreibe Python-Code für eine Klasse mit dem Namen `Penguin`.
  - Diese Klasse soll eine Kindklasse von `Bird` sein.
  - Diese Klasse soll alle Methoden von `Bird` erben, inklusive der `__init__`-Methode.
  - Ändere die `__init__`-Methode der Klasse `Penguin` so, dass das Attribut `continent` immer den Wert `"Antarctic"` hat.
  - Ändere die `__init__`-Methode so, dass die Attribute `can_swim` und `can_dive` immer den Wert `True` haben und das Attribut `can_fly` immer den Wert `False` hat.

mögliche Lösungen

Variante A

```
# Lösung Kapitel OOP Aufgabe 5 Variante A
class Penguin(Bird):
    def __init__(self, name):
        # Wenn man den Elternklassennamen verwendet, muss self angegeben werden
        Bird.__init__(self, name)
        self.continent = "Antarctic"
        self.can_swim = True
        self.can_dive = True
        self.can_fly = False
```

Variante B

```
# Lösung Kapitel OOP Aufgabe 5 Variante B
class Penguin(Bird):
    def __init__(self, name):
        # super() verweist auf die Elternklasse
        super().__init__(name)  # self wird hier nicht benötigt
        self.continent = "Antarctic"
        self.can_swim = True
        self.can_dive = True
        self.can_fly = False
```

20.6 Aufgaben für das Kapitel *Eingabe und Ausgabe*

20.6.1 Aufgabe io 1

- Schreibe ein Python-Programm, das den String `"Hello World!"` in eine Datei mit dem *Dateinamen* `hello.txt` schreibt.
- Gib den Text `File written to disk` aus.

mögliche Lösungen

Variante A

```
# Lösung Kapitel IO Aufgabe 1 Variante A
text = "Hello World!"
myfile = open("hello.txt", "w")  # write mode
myfile.write(text)
myfile.close()
print("File written to disk")
```

Variante B

```
# Lösung Kapitel IO Aufgabe 1 Variante B
text = "Hello World!"
with open("hello.txt", "w") as myfile:
    myfile.write(text)
# schließt automatisch!
print("File written to disk")
```

20.6.2 Aufgabe io 2

- Falls du Aufgabe 1 nicht bearbeitet hast:
  - Erstelle (mit einem Texteditor) eine neue Textdatei, die eine einzige Textzeile enthält.
  - Das letzte Zeichen der Textzeile soll ein *Ausrufezeichen* (`!`) sein.
  - Speichere diese Textdatei unter dem Dateinamen `hello.txt`.
- Überprüfe mit Hilfe eines Datei-Managers, ob sich die Datei `hello.txt` im aktuellen Ordner befindet.
- Schreibe ein Python-Programm, das die bestehende Textdatei `hello.txt` so verändert, dass:
  - Zwei leere Zeilen am Ende des bestehenden Textes hinzugefügt werden.
  - Nach diesen 2 leeren Zeilen eine neue Textzeile hinzugefügt wird mit dem Text: `How do you do?`
  - Der Text mit einem *Zeilenumbruch* endet.
- Schreibe Python-Code, der auf dem Bildschirm die Worte `Textline added` ausgibt.

mögliche Lösung

```
# Lösung Kapitel IO Aufgabe 2 Variante A
text = "\n\n\nhow do you do?\n"
with open("hello.txt", "a") as myfile:
    myfile.write(text)
print("Textline added")
```

20.6.3 Aufgabe io 3

- Schreibe ein Programm, das eine vorhandene Textdatei mit dem Namen `hello.txt` öffnet.
- Das Programm soll herausfinden, wie viele Textzeilen (wie viele Zeilenumbrüche) in dieser Datei enthalten sind.
- Das Programm soll den String `lines found:` und anschließend die Anzahl der Textzeilen ausgeben.

mögliche Lösung

```
# Lösung Kapitel IO Aufgabe 3 Variante A
with open("hello.txt") as myfile:
    lines = myfile.readlines()
print("lines found:", len(lines))
```

20.7 Aufgaben für das Kapitel *Ausnahmen*

20.7.1 Aufgabe exceptions 1

- gegeben ist dieses Python-Programm:

```
# question chapter exception task 1
print("please enter your year of birth in the format YYYY")
year_text = input(">>>")
year = int(year_text)
print("in the year 2050, you will be ", 2050-year, "years old")
```

- Ändere das Programm so, dass es **nicht** mit einem `ValueError` endet, wenn der Benutzer eine falsche Eingabe macht (zum Beispiel Buchstaben anstelle einer Zahl eingibt). Stattdessen soll das Programm den Benutzer erneut um eine Eingabe bitten, bis die Eingabe eine Zahl ist.

mögliche Lösungen

Variante A

```
# Lösung Kapitel Exceptions Aufgabe 1 Variante A
while True:
    print("please enter your year of birth in the format YYYY")
    year_text = input(">>>")
    try:
        year = int(year_text)
    except ValueError:
        print("not a number, please try again")
        continue
    # Eingabe war korrekt
    break
print("in the year 2050, you will be ", 2050-year, "years old")
```

Variante B

```
# Lösung Kapitel Exceptions Aufgabe 1 Variante B
while True:
    print("please enter your year of birth in the format YYYY")
    year_text = input(">>>")
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    if year_text.isdigit():
        year = int(year_text)
        break
    print("not a number, please try again")
print("in the year 2050, you will be ", 2050-year, "years old")
```

20.8 Aufgabe 2

-gegeben ist dieser Python-Code:

```
# question chapter exceptions task 2
chessboard = [
              ["white","black","white","black","white","black","white","black"],
              ["black","white","black","white","black","white","black","white"],
              ["white","black","white","black","white","black","white","black"],
              ["black","white","black","white","black","white","black","white"],
              ["white","black","white","black","white","black","white","black"],
              ["black","white","black","white","black","white","black","white"],
              ["white","black","white","black","white","black","white","black"],
              ["black","white","black","white","black","white","black","white"],
             ]

def get_color(row, column):
    row = int(row)
    column = int(column)
    return chessboard[row][column]

while True:
    r = input("enter row number: (0-7) >>>")
    c = input("enter column number (0-7) >>>")
    result = get_color(r,c)
    print(f"The color the field (row {r} column {c}) is: {result}")
```

- Ändere den Code der Funktion `get_color` so, dass:
  - Die Funktion den String `"no numbers"` zurückgibt, wenn `row` oder `column` (oder beide) keine Ganzzahlen sind.
  - Die Funktion den String `"bad index"` zurückgibt, wenn `row` oder `column` (oder beide) kleiner als 0 oder größer als 7 sind.

mögliche Lösung

```
# Lösung Kapitel Exceptions Aufgabe 2
def get_color(row, column):
    try:
        r = int(row)
        c = int(column)
    except ValueError:
        return "no numbers"
    if (c < 0) or (c > 7) or (r < 0) or (r > 7):
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        return "bad index"
    return chessboard[r][c]
```

CHAPTER 21

Anhang: FLOSS

Warnung

Bitte beachten Sie, dass dieser Abschnitt im Jahr 2003 verfasst wurde. Einige Inhalte mögen daher veraltet oder nostalgisch wirken. :-)

"Free/Libre and Open Source Software", kurz FLOSS, basiert auf dem Konzept einer Gemeinschaft, die ihrerseits auf dem Konzept des Teilens – und insbesondere des Teilens von Wissen – aufbaut. FLOSS-Software ist kostenlos für die Nutzung, Modifikation und Weiterverbreitung verfügbar.

Wenn Sie dieses Buch bereits gelesen haben, dann sind Sie bereits mit FLOSS vertraut, da Sie die ganze Zeit über *Python* verwendet haben – und Python ist eine Open-Source-Software!

Hier sind einige Beispiele für FLOSS, um eine Vorstellung davon zu geben, welche Art von Dingen durch gemeinschaftliches Teilen und Aufbauen entstehen können:

21.1 Beispiele für FLOSS

- **Linux:** Dies ist ein FLOSS-Betriebssystemkern, der im GNU/Linux-Betriebssystem verwendet wird. Linux, der Kernel, wurde von Linus Torvalds als Student gestartet. Android basiert auf Linux. Fast alle Websites, die Sie heutzutage nutzen, laufen wahrscheinlich auf Linux.
- **Ubuntu:** Dies ist eine gemeinschaftlich getragene Distribution, die von Canonical gesponsert wird und heute die beliebteste GNU/Linux-Distribution ist. Sie ermöglicht es Ihnen, eine Fülle von verfügbarer FLOSS-Software zu installieren – und das auf einfache und benutzerfreundliche Weise. Am besten: Sie können Ihren Computer neu starten und GNU/Linux direkt von der CD ausführen! Dadurch können Sie das neue Betriebssystem vollständig ausprobieren, bevor Sie es auf Ihrem Computer installieren. Allerdings ist Ubuntu nicht vollständig freie Software, da es proprietäre Treiber, Firmware und Anwendungen enthält.
- **LibreOffice:** Dies ist eine hervorragende, gemeinschaftlich entwickelte und gepflegte Büro-Suite mit Textverarbeitung, Präsentations-, Tabellenkalkulations- und Zeichenkomponenten unter anderem. Es kann sogar MS Word- und MS PowerPoint-Dateien problemlos öffnen und bearbeiten. Es läuft auf fast allen Plattformen und ist vollständig kostenlos, frei und Open-Source-Software.

- **Mozilla Firefox:** Dies ist **der** beste Webbrowser. Er ist extrem schnell und hat für seine sinnvollen und beeindruckenden Funktionen große Anerkennung erhalten. Das Erweiterungs-Konzept ermöglicht die Verwendung aller Arten von Plugins.
- **Mono:** Dies ist eine Open-Source-Implementierung der Microsoft .NET-Plattform. Sie ermöglicht die Erstellung und Ausführung von .NET-Anwendungen auf GNU/Linux, Windows, FreeBSD, Mac OS und vielen anderen Plattformen.
- **Apache Webserver:** Dies ist der beliebte Open-Source-Webserver. Tatsächlich ist er **der** beliebteste Webserver auf dem Planeten! Er betreibt fast mehr als die Hälfte aller Websites. Ja, das stimmt – Apache bedient mehr Websites als alle Konkurrenten (einschließlich Microsoft IIS) zusammen.
- **VLC Player:** Dies ist ein Videoplayer, der alles von DivX über MP3, Ogg bis hin zu VCDs und DVDs abspielen kann – wer sagt denn, dass Open Source keinen Spaß macht? ;-)

Diese Liste soll Ihnen nur einen kurzen Überblick geben – es gibt noch viele weitere hervorragende FLOSS-Projekte, wie z. B. die Programmiersprache Perl, die Programmiersprache PHP, das Content-Management-System Drupal für Websites, den Datenbankserver PostgreSQL, das Rennspiel TORCS, die IDE KDevelop, den Filmplayer Xine, den Editor VIM, den Editor Quanta+, den Audio-Player Banshee, das Bildbearbeitungsprogramm GIMP, … Diese Liste könnte endlos weitergeführt werden.

21.1.1 Aktuelle Nachrichten aus der FLOSS-Welt

Besuchen Sie die folgenden Websites, um die neuesten Nachrichten aus der FLOSS-Welt zu erfahren:

- OMG! Ubuntu!
- Web Upd8
- DistroWatch
- Planet Debian

21.1.2 Weitere Informationen zu FLOSS

Besuchen Sie die folgenden Websites für weitere Informationen zu FLOSS:

- GitHub Explore
- Code Triage
- SourceForge
- FreshMeat

Also, machen Sie sich auf den Weg und erkunden Sie die weite, freie und offene Welt von FLOSS!

CHAPTER 22

Anhang: Impressum

Fast die gesamte Software, die ich für dieses Buch verwendet habe, ist freie Software (*FLOSS* *F*ree, *L*ibre, *O*pen-*S*ource).

22.1 Entstehung des Buches

Der ersten Entwurf dieses Buches entstand auf einem Red Hat Linux 9.0 System, der sechsten Etwurf entsand auf einem Fedora Core 3 Linux.

Zum Schreiben der Texte wurd am Anfang das Programm KWord verwendet (siehe auch*Revision History*).

22.2 Die Anfänge

Später wechselte der Autor zu DocBook XML mit [Kate](https://de.wikipedia.org/wiki/Kate_(Texteditor), fand dies aber zu umständlich. Danach erfolgte ein Wechsel zu OpenOffice, das hinsichtlich der Formatierungsmöglichkeiten und der PDF-Erstellung hervorragend war, jedoch sehr unsauberen HTML-Code erzeugte.

Schließlich entdeckte der Autor XEmacs und schrieb das Buch (erneut) komplett in DocBook XML-Format neu, nachdem er sich für dieses Format entschieden hatte.

Im sechsten Entwurf folgte ein Wechsel zu Quanta+ als Bearbeitungswerkzeug. Dabei kamen die standardmäßigen XSL-Stylesheets von Fedora Core 3 Linux zum Einsatz. Zusätzlich wurde ein CSS-Dokument erstellt, um den HTML-Seiten Farbe und Stil zu verleihen. Außerdem hatte schrieb der Autor einen einfachen lexikalischen Analysator in Python, der automatisch Syntaxhervorhebung für alle Programmlisten bereitstellte.

Für den siebten Entwurf wurde MediaWiki als Grundlage verwendet. Die Bearbeitung erfolgte online, und die Leser konnten direkt auf der Wiki-Website lesen, bearbeiten und diskutieren. Letztendlich kostete die Spam-Bekämpfung mehr Zeit als das Schreiben selbst.

Für den achten Entwurf verwendete der Autor Vim (siehe auch "A byte of Vim", ebenfalls von Swaroop CH), Pandoc und Mac OS X.

Für den neunten Entwurf wurde ins AsciiDoc-Format gewechselt unter Verwendung von Emacs 24.3 als Texteditor, mit dem Tomorrow-Theme (sowie Firefox-Mod-Theme) und die Schriftart Fira Mono.

22.3 Aktuell

2016: AsciiDoc nervte den Autor mit kleineren Darstellungsfehlern, Beispielsweise verschwanden die ++ Zeichen in C/C++, und es war schwierig, den Überblick über die korrekte Markierung solcher Kleinigkeiten zu behalten.

Für den zehnten Entwurf wechselte der Autor zum Markdown-Format mit GitBook und dem Spacemacs-Editor.

November 2020: Da GitBook seine Open-Source-Software eingestellt hat, migrierte der Autor ich zu Honkit, einem von der Community gepflegten Fork des GitBook-Legacy.

22.4 Über den Autor

Siehe https://www.swaroopch.com/

22.5 Über diese Übersetzung

Diese deutsche Übersetzung von "A byte of Python" wurde von Horst JENS mittels Sphinx, Myst und Sphinxbook-theme erzeugt auf einem Linux Mint System.

CHAPTER 23

Anhang: Geschichtsstunde

Ich begann mit Python, als ich ein Installationsprogramm für meine Software „Diamond“ schreiben musste, um die Installation zu vereinfachen. Ich musste mich zwischen Python- und Perl-Bindings für die Qt-Bibliothek entscheiden. Bei meiner Recherche im Internet stieß ich auf einen Artikel von Eric S. Raymond (http://www.python.org/about/success/esr/), einem bekannten und angesehenen Hacker, in dem er beschrieb, wie Python zu seiner Lieblingsprogrammiersprache geworden war. Ich fand auch heraus, dass die PyQt-Bindings im Vergleich zu Perl-Qt ausgereifter waren. Daher entschied ich mich für Python.

Anschließend suchte ich nach einem guten Buch über Python. Ich wurde jedoch nicht fündig! Zwar fand ich einige Bücher von O'Reilly, aber diese waren entweder zu teuer oder eher Nachschlagewerke als praktische Anleitungen. Schließlich begnügte ich mich mit der mitgelieferten Python-Dokumentation. Diese war jedoch zu kurz und unvollständig. Sie vermittelte zwar einen guten Überblick über Python, war aber nicht vollständig. Ich kam damit zurecht, da ich bereits Programmiererfahrung hatte, aber für Anfänger war es ungeeignet.

Etwa sechs Monate nach meinen ersten Berührungen mit Python installierte ich das (damals) neueste Red Hat 9.0 Linux und experimentierte mit KWord. Ich war begeistert und hatte plötzlich die Idee, etwas über Python zu schreiben. Ich begann mit ein paar Seiten, doch der Text wurde schnell 30 Seiten lang. Daraufhin beschloss ich, ihn in Buchform verständlicher zu gestalten. Nach unzähligen Überarbeitungen ist er nun ein nützlicher Leitfaden zum Erlernen der Programmiersprache Python. Ich betrachte dieses Buch als meinen Beitrag und meine Hommage an die Open-Source-Community.

Dieses Buch begann als meine persönlichen Notizen zu Python und ich betrachte es immer noch als solche, obwohl ich mir viel Mühe gegeben habe, es verständlicher zu gestalten.

Ganz im Sinne von Open Source habe ich viele konstruktive Vorschläge, Kritikpunkte und Feedback von begeisterten Lesern erhalten, was mir sehr geholfen hat, dieses Buch zu verbessern.

23.1 Status des Buches

Das Buch benötigt die Hilfe seiner Leser, um auf Stellen hinzuweisen, die unbrauchbar, unverständlich oder fehlerhaft sind. Bitte senden Sie Ihre Kommentare und Vorschläge (idealerweise direkt via Github Issues/Pull requests) an den Swaroop CH (Autor der englischen Originalausgabe) oder die *jeweiligen Übersetzer*.

- Für di deutsche Überestzung welche Sie gerade lesen: Horst JENS

CHAPTER 24

Anhang: Versionsverlauf

- Keine Versionsänderung
- 6. Nov. 2020
- Migration von GitBook (eingestellt) zu Community-Projekt Honkit, einem Fork von GitBook
- 4.0
- 19. Jan. 2016
- Umstellung zurück auf Python 3
- Umstellung zurück auf Markdown mit GitBook und Spacemacs
- 3.0
- 31. März 2014
- Neu geschrieben für Python 2 mit AsciiDoc und adoc-mode
- 2.1
- 3. August 2013
- Neu geschrieben mit Markdown und Jason Blevins' Markdown Mode
- 2.0
- 20. Oktober 2012
- Neu geschrieben im Pandoc-Format, vielen Dank an meine Frau, die den Großteil der Konvertierung vom MediaWiki-Format übernommen hat.
- Vereinfachung des Textes, Entfernung nicht notwendiger Abschnitte wie `nonlocal` und Metaklassen
- 1.90
- 4. September 2008 und noch in Bearbeitung
- Wiederbelebung nach 3,5 Jahren Pause!
- Neu geschrieben für Python 3.0

- Neu geschrieben mit MediaWiki (erneut)
- 1.20
- 13. Januar 2005
- Komplett neu geschrieben mit Quanta+ auf Fedora Core 3 mit vielen Korrekturen und Aktualisierungen. Viele neue Beispiele. Meine DocBook-Konfiguration wurde komplett neu geschrieben.
- 1.15
- 28. März 2004
- Kleinere Überarbeitungen
- 1.12
- 16. März 2004
- Ergänzungen und Korrekturen
- 1.10
- 9. März 2004
- Weitere Tippfehlerkorrekturen, vielen Dank an die vielen engagierten und hilfsbereiten Leser.
- 1.00
- 8. März 2004
- Nach dem zahlreichen Feedback und den Anregungen der Leser habe ich den Inhalt grundlegend überarbeitet und Tippfehler korrigiert.
- 0.99
- 22. Feb. 2004
- Neues Kapitel über Module hinzugefügt. Details zur variablen Anzahl von Argumenten in Funktionen ergänzt.
- 0.98
- 16. Feb. 2004
- Python-Skript und CSS-Stylesheet zur Verbesserung der XHTML-Ausgabe geschrieben, inklusive eines einfachen, aber funktionalen lexikalischen Analysators für die automatische Syntaxhervorhebung der Programmlisten (ähnlich wie bei VIM).
- 0.97
- 13. Feb. 2004
- Komplett überarbeiteter Entwurf, erneut in DocBook XML. Das Buch ist deutlich verbessert – kohärenter und lesbarer.
- 0.93
- 25. Jan. 2004
- Erläuterungen zu IDLE und weitere Windows-spezifische Informationen hinzugefügt.
- 0.92
- 5. Jan. 2004
- Änderungen an einigen Beispielen.
- 0.91
  - 30. Dezember 2003

  - Tippfehler korrigiert. Viele Themen improvisiert.
- 0.90
  - 18. Dezember 2003
  - Zwei weitere Kapitel hinzugefügt. OpenOffice-Format mit Revisionen.
- 0.60
  - 21. November 2003
  - Komplett neu geschrieben und erweitert.
- 0.20
  - 20. November 2003
  - Einige Tippfehler und Fehler korrigiert.
- 0.15
  - 20. November 2003
  - Mit XEmacs in DocBook XML konvertiert.
- 0.10
  - 14. November 2003
  - Erster Entwurf mit KWord.

CHAPTER 25

Übersetzungen

Dank der unermüdlichen Arbeit vieler Freiwilliger ist das Buch in viele verschiedene menschliche Sprachen übersetzt worden!

Wenn Sie bei diesen Übersetzungen mithelfen möchten, werfen Sie bitte einen Blick auf die Liste der Freiwilligen und Sprachen unten und entscheiden Sie, ob Sie eine neue Übersetzung beginnen oder bei bestehenden Übersetzungsprojekten mithelfen möchten.

Falls Sie planen, eine neue Übersetzung zu beginnen, lesen Sie bitte die *Übersetzungsanleitung*.

25.1 Arabisch

Unten ist der Link für die arabische Version. Dank an Ashraf Ali Khalaf für die Übersetzung des Buches, du kannst es von sourceforge.net herunterladen und weitere Informationen unter http://itwadi.com/byteofpython_arabi finden.

25.2 Aserbaidschanisch

Jahangir Shabiyev (c.shabiev@gmail.com) hat sich freiwillig gemeldet, das Buch ins Aserbaidschanische zu übersetzen.

25.3 Brasilianisches Portugiesisch

Es gibt zwei Übersetzungen in unterschiedlichen Stadien der Fertigstellung und Zugänglichkeit. Die ältere Übersetzung fehlt bzw. ist verloren gegangen, und die neuere Übersetzung ist unvollständig.

Samuel Dias Neto (samuel.arataca@gmail.com) hat die erste brasilianisch-portugiesische Übersetzung (ältere Übersetzung) dieses Buches angefertigt, als Python in Version 2.3.5 war. Diese ist nicht mehr öffentlich zugänglich.

Rodrigo Amaral (rodrigoamaral@gmail.com) hat sich freiwillig gemeldet, das Buch ins brasilianische Portugiesisch zu übersetzen (neuere Übersetzung), die noch fertiggestellt werden muss.

25.4 Katalanisch

Moises Gomez (moisesgomezgiron@gmail.com) hat sich freiwillig gemeldet, das Buch ins Katalanische zu übersetzen. Die Übersetzung ist in Arbeit.

> Moisès Gómez – Ich bin Entwickler und auch Lehrer für Programmierung (normalerweise für Menschen ohne jegliche Vorkenntnisse).
>
> Vor einiger Zeit musste ich lernen, wie man in Python programmiert, und Swaroops Werk war wirklich hilfreich. Klar, präzise und ausreichend vollständig. Genau das, was ich brauchte.
>
> Nach dieser Erfahrung dachte ich, dass auch andere Menschen in meinem Land davon profitieren könnten. Aber die englische Sprache kann eine Barriere sein.
>
> Also, warum nicht versuchen, es zu übersetzen? Und das habe ich für eine frühere Version von BoP getan.
>
> In meinem Land gibt es zwei offizielle Sprachen. Ich habe die katalanische Sprache gewählt, in der Annahme, dass andere es in die weiter verbreitete spanische Sprache übersetzen werden.

25.5 Frühere chinesische Übersetzung

Im Jahr 2005 übersetzte Shen Jieyuan dieses Buch in der Version 1.20 ins Chinesische und veröffentlichte es im Internet. Dies ist die erste chinesische Ausgabe. Auf der offiziellen BoP-Seite wurde er Juan Shen genannt, mit der E-Mail-Adresse orion_val@163.comDiese Ausgabe wurde weit im Netzwerk verbreitet, und die Links, die von der offiziellen BoP-Seite bereitgestellt wurden, sind nicht mehr verfügbar, sodass die ursprüngliche Quelle nicht mehr gefunden werden kann. Daher kann hier keine bestimmte Adresse angegeben werden. Aber du kannst versuchen, nach Schlüsselwörtern wie „Python “ zu suchen, um eine Kopie zu finden.

Juan Shen sagt:

> Ich bin Postgraduierter an der Wireless Telecommunication Graduate School der Beijing University of Technology, China. Mein aktuelles Forschungsinteresse liegt in der Synchronisation, Kanalschätzung und Multi-User-Erkennung von Multicarrier-CDMA-Systemen. Python ist meine wichtigste Programmiersprache für tägliche Simulationen und Forschungsarbeit, tatsächlich mit Hilfe von Python Numeric. Ich habe Python erst vor einem halben Jahr gelernt, aber wie du siehst, ist es wirklich leicht verständlich, leicht zu benutzen und produktiv. Genau wie in Swaroops Buch versichert wird: „Es ist jetzt meine Lieblingsprogrammiersprache.“
>
> „A Byte of Python“ ist mein Tutorial, um Python zu lernen. Es ist klar und effektiv, dich in kürzester Zeit in die Welt von Python zu führen. Es ist nicht zu lang, deckt aber effizient fast alle wichtigen Dinge in Python ab. Ich denke, „A Byte of Python“ sollte Anfängern als ihr erstes Python-Tutorial dringend empfohlen werden. Ich widme meine Übersetzung den potenziell Millionen von Python-Nutzern in China.

25.6 Chinesisch (Traditionell)

Fred Lin (gasolin@gmail.com) hat sich freiwillig gemeldet, das Buch ins traditionelle Chinesisch zu übersetzen.

Es ist verfügbar unter http://code.google.com/p/zhpy/wiki/ByteOfZhpy.

Ein aufregendes Merkmal dieser Übersetzung ist, dass sie auch die ausführbaren chinesischen Python-Quellcodes Seite an Seite mit den ursprünglichen Python-Quellcodes enthält.

> Fred Lin – Ich arbeite als Netzwerk-Firmware-Ingenieur bei Delta Network und bin außerdem ein Beitragender zum TurboGears-Webframework.
>
> Als Python-Evangelist (:-p) brauche ich Material, um die Programmiersprache Python zu fördern. Ich fand, dass „A Byte of Python“ genau den richtigen Punkt trifft – sowohl für Anfänger als auch für erfahrene Programmierer. „A Byte of Python“ erläutert die wichtigsten Python-Grundlagen in angenehmer Größe.

Die Übersetzung basierte ursprünglich auf der vereinfachten chinesischen Version, und bald wurden viele Überarbeitungen vorgenommen, um sie an die aktuelle Wiki-Version und die Lesbarkeit anzupassen.

Die aktuelle traditionelle chinesische Version enthält außerdem ausführbare chinesische Python-Quellen, die durch mein neues „zhpy"-Projekt (Python auf Chinesisch) erreicht wurden (gestartet im August 07).

zhpy (ausgesprochen (Z.H.?, oder „zippy")) baut eine Schicht über Python auf, um Python in Chinesisch (traditionell oder vereinfacht) zu übersetzen oder damit zu interagieren. Dieses Projekt richtet sich hauptsächlich an den Bildungsbereich.

25.7 Französisch

Gregory (coulix@ozforces.com.au) hat sich freiwillig gemeldet, das Buch ins Französische zu übersetzen.

Gérard Labadie (gerard.labadie@gmail.com) hat die Übersetzung des Buches ins Französische abgeschlossen.

Diese Übersetzung wurde später ins Markdown-Format übertragen, aktualisiert, um der neuesten Version des Buches zu entsprechen, und auf GitBook von Romain Gilliotte (rgilliotte@gmail.com) veröffentlicht.

Sie ist verfügbar unter https://rgilliotte.gitbook.io/byte-of-python/

25.8 Deutsch

Horst JENS (horst.jens@spielend-programmieren.at) hat eine deutsche Übersetzung (basierend auf Python3) gemacht die auf https://spielend-programmieren.at/byte_of_python_deutsch/ zu finden ist.

25.8.1 ältere Versionen (basierend auf Python2)

Lutz Horn (lutz.horn@gmx.de), Bernd Hengelein (bernd.hengelein@gmail.com) und Christoph Zwerschke (cito@online.de) haben sich freiwillig gemeldet, das Buch ins Deutsche zu übersetzen.

Die Übersetzung ist verfügbar unter http://cito.github.io/byte_of_python/

Lutz Horn sagt:

> Ich bin 32 Jahre alt und habe einen Abschluss in Mathematik von der Universität Heidelberg, Deutschland. Zurzeit arbeite ich als Softwareingenieur in einem öffentlich geförderten Projekt zum Aufbau eines Webportals für alle Themen rund um die Informatik in Deutschland. Die Hauptsprache, die ich beruflich benutze, ist Java, aber ich versuche, so viel wie möglich im Hintergrund mit Python zu erledigen. Besonders Textanalyse und -konvertierung sind mit Python sehr einfach. Ich bin nicht sehr vertraut mit GUI-Toolkits, da sich der Großteil meiner Programmierung um Webanwendungen dreht, bei denen die Benutzeroberfläche mit Java-Frameworks wie Struts erstellt wird. Momentan versuche ich, die funktionalen Programmiermerkmale von Python und Generatoren stärker zu nutzen. Nachdem ich einen kurzen Blick auf Ruby geworfen habe, war ich sehr beeindruckt von der Nutzung von Blöcken in dieser Sprache. Generell mag ich die dynamische Natur von Sprachen wie Python und Ruby, da sie mir Dinge ermöglichen, die in statischeren Sprachen wie Java nicht möglich sind. Ich habe nach einer Art Einführung ins Programmieren gesucht, die sich eignet, einem völligen Nicht-Programmierer das Programmieren beizubringen. Ich fand das Buch „How to Think Like a Computer Scientist: Learning with Python" und „Dive into Python". Das erste ist gut für Anfänger, aber zu lang zum Übersetzen. Das zweite ist nicht für Anfänger geeignet. Ich denke, „A Byte of Python" liegt genau zwischen diesen beiden, da es nicht zu lang ist, auf den Punkt geschrieben und gleichzeitig ausführlich genug, um einem Anfänger etwas beizubringen. Außerdem gefällt mir die einfache DocBook-Struktur, die es zu einem Vergnügen macht, den Text zu übersetzen und Ausgaben in verschiedenen Formaten zu erzeugen.
>
> Bernd Hengelein sagt:
>
> Lutz und ich werden die deutsche Übersetzung gemeinsam machen. Wir haben gerade erst mit der Einleitung und dem Vorwort begonnen, aber wir werden dich über unsere Fortschritte auf dem Laufenden

halten. Okay, jetzt ein paar persönliche Dinge über mich. Ich bin 34 Jahre alt und spiele seit den 1980ern mit Computern, als der „Commodore C64“ die Kinderzimmer beherrschte. Nach dem Studium der Informatik begann ich als Softwareingenieur zu arbeiten. Zurzeit arbeite ich im Bereich der medizinischen Bildgebung für ein großes deutsches Unternehmen. Obwohl C++ die Hauptsprache ist, die ich (zwangsweise) bei der täglichen Arbeit benutze, bin ich ständig auf der Suche nach neuen Dingen, die ich lernen kann. Letztes Jahr habe ich mich in Python verliebt, was eine wunderbare Sprache ist – sowohl wegen ihrer Möglichkeiten als auch ihrer Schönheit. Ich las irgendwo im Netz über jemanden, der sagte, dass er Python mag, weil der Code so schön aussieht. Meiner Meinung nach hat er absolut recht. Als ich mich entschied, Python zu lernen, bemerkte ich, dass es nur sehr wenig gute Dokumentation auf Deutsch gibt. Als ich auf dein Buch stieß, kam mir spontan die Idee einer deutschen Übersetzung. Glücklicherweise hatte Lutz dieselbe Idee, und wir können nun die Arbeit aufteilen. Ich freue mich auf eine gute Zusammenarbeit!

25.9 Griechisch

Die griechische Ubuntu-Community hat das Buch ins Griechische übersetzt, zur Verwendung in unseren Online-asynchronen Python-Lektionen, die in unseren Foren stattfinden. Kontaktiere @savvasradevic für weitere Informationen.

25.10 Indonesisch

Daniel (daniel.mirror@gmail.com) übersetzt das Buch ins Indonesische unter http://python.or.id/moin.cgi/ByteofPython.

Wisnu Priyambodo (cibermen@gmail.com) hat sich ebenfalls freiwillig gemeldet, das Buch ins Indonesische zu übersetzen. http://python.or.id/moin.cgi/ByteofPython

Außerdem hat sich Bagus Aji Santoso (baguzzzaji@gmail.com) freiwillig gemeldet.

25.11 Italienisch (erste)

Enrico Morelli (mr.mlucci@gmail.com) und Massimo Lucci (morelli@cerm.unifi.it) haben sich freiwillig gemeldet, das Buch ins Italienische zu übersetzen.

Massimo Lucci und Enrico Morelli – wir arbeiten an der Universität Florenz (Italien) – Fachbereich Chemie. Ich (Massimo) als Servicetechniker und Systemadministrator für Kernspinresonanzspektrometer; Enrico als Servicetechniker und Systemadministrator für unser CED und parallele/clusterbasierte Systeme. Wir programmieren seit etwa sieben Jahren in Python und haben seit zehn Jahren Erfahrung mit Linux-Plattformen. In Italien sind wir verantwortlich für die Websites www.gentoo.it für die Gentoo/Linux-Distribution und www.nmr.it (derzeit im Aufbau) für Anwendungen der Kernspinresonanz und Kongressorganisation und -verwaltung. Das ist alles! Wir sind beeindruckt von der klaren Sprache in deinem Buch, und wir denken, dies ist wesentlich, um Python neuen Nutzern näherzubringen (wir denken an Hunderte von Studenten und Forschern in unseren Labors).

25.12 Italienisch (zweite)

Eine italienische Übersetzung wurde erstellt von Calvina Bice & Kollegen unter http://besthcgdropswebsite.com/translate/a-byte-of-python/.

25.13 Japanisch

Shunro Dozono (dozono@gmail.com) übersetzt das Buch ins Japanische.

25.14 Koreanisch

25.14.1 Epsimatt (2019)

Epsimatt hat eine neue koreanische Übersetzung begonnen:

- Online lesen unter https://epsimatt.gitbook.io/byte-of-python/
- Fortschritt verfolgen unter https://github.com/epsimatt/byte-of-python/issues/16

25.14.2 Ältere Versionen

Jeongbin Park (pjb7687@gmail.com) hat das Buch ins Koreanische übersetzt – https://github.com/pjb7687/byte_of_python

> Ich bin Jeongbin Park, zurzeit arbeite ich als Forscher für Biophysik & Bioinformatik in Korea.
>
> Vor einem Jahr suchte ich nach einem guten Tutorial/Leitfaden für Python, um es meinen Kollegen vorzustellen, da die Nutzung von Python in solchen Forschungsbereichen immer unvermeidlicher wird, weil die Nutzerbasis immer weiter wächst.
>
> Aber zu dieser Zeit gab es nur wenige Python-Bücher auf Koreanisch, also entschied ich mich, dein E-Book zu übersetzen, weil es wie einer der besten Leitfäden aussah, die ich je gelesen habe!
>
> Zurzeit ist das Buch fast vollständig ins Koreanische übersetzt, außer einige Texte im Einführungskapitel und in den Anhängen.
>
> Nochmals vielen Dank für das Schreiben eines so guten Leitfadens!

25.15 Mongolisch

Ariunsanaa Tunjin (luftballons2010@gmail.com) hat sich freiwillig gemeldet, das Buch ins Mongolische zu übersetzen.

Update vom 22. November 2009: Ariunsanaa steht kurz vor der Fertigstellung der Übersetzung.

25.16 Norwegisch (Bokmål)

Eirik Vågeskar ist ein Schüler an der Sandvika videregående skole in Norwegen, ein Blogger und übersetzt derzeit das Buch ins Norwegische (Bokmål).

> *Eirik Vågeskar*: Ich wollte schon immer programmieren, aber da ich eine kleine Sprache spreche, war der Lernprozess viel schwieriger. Die meisten Tutorials und Bücher sind in sehr technischem Englisch geschrieben, sodass die meisten Schulabgänger nicht einmal das Vokabular besitzen, um zu verstehen, worum es in dem Tutorial geht. Als ich dieses Buch entdeckte, waren all meine Probleme gelöst. „A Byte of Python“ verwendet einfache, nicht-technische Sprache, um eine Programmiersprache zu erklären, die ebenso einfach ist, und diese beiden Dinge machen das Lernen von Python Spaß. Nachdem ich die Hälfte des Buches gelesen hatte, entschied ich, dass das Buch eine Übersetzung wert ist. Ich hoffe, die Übersetzung wird Menschen helfen, die sich in derselben Situation befanden wie ich (insbesondere junge Leute) und vielleicht Interesse an der Sprache bei Menschen mit weniger technischem Wissen wecken.

25.17 Polnisch

Dominik Kozaczko (dominik@kozaczko.info) hat sich freiwillig gemeldet, das Buch ins Polnische zu übersetzen.

Update : Die Übersetzung ist abgeschlossen und seit dem 2. Oktober 2009 bereit. Dank an Dominik, seine zwei Studenten und ihren Freund für ihre Zeit und Mühe!

> Dominik Kozaczko – Ich bin Lehrer für Informatik und Informationstechnologie.

25.18 Portugiesisch

Artur Weber (arturweberguimaraes@gmail.com) hat eine Übersetzung dieses Buches ins Portugiesische abgeschlossen (Stand 21. Februar 2018) unter https://www.homeyou.com/~edu/introducao.

> *Artur Weber*: Meine Studenten studieren an der polytechnischen Fakultät der Ökologischen Universität in der Stadt Curitiba (Brasilien), und einige von ihnen interessieren sich für verschiedene Dokumente.
>
> Da sie Kurs- und akademische Arbeiten schreiben, suchen sie immer nach interessanten Artikeln und Seiten. Ich bemühe mich ebenfalls, interessante Materialien zu finden, die Quellen für ihre universitären Arbeiten sein können.
>
> Ich fand die Materialien auf deiner Seite nützlich für einige meiner Studenten, die Arbeiten im Bereich Python-Programmierung schreiben. Deshalb habe ich beschlossen, eine portugiesische Übersetzung anzufertigen, um meinen Studenten, die kein Englisch können, spannende Artikel in ihrer Muttersprache (auf Portugiesisch) zugänglich zu machen.

25.19 Russisch

Vladimir Smolyar (v_2e@ukr.net) hat eine russische Übersetzung unter http://wombat.org.ua/AByteOfPython/ abgeschlossen.

25.20 Ukrainisch

Daria JENS (jensdarya@gmail.com) hat eine Übersetzung in die ukrainische Sprache abgeschlossen: https://spielend-programmieren.at/byte_of_python_ukraine/

Averkiev Andrey (averkiyev@ukr.net) hat sich freiwillig gemeldet, das Buch ins Russische und vielleicht auch ins Ukrainische zu übersetzen (wenn es die Zeit erlaubt).

25.21 Serbisch

„BugSpice“ (amortizerka@gmail.com) hat eine serbische Übersetzung abgeschlossen:

> Dieser Download-Link ist nicht mehr zugänglich.

Mehr Details unter http://forum.ubuntu-rs.org/Thread-zagrljaj-pitona.

25.22 Slowakisch

Albertio Ward (albertioward@gmail.com) hat das Buch ins Slowakische übersetzt.

> Wir sind eine gemeinnützige Organisation namens „Übersetzung für Bildung“. Wir vertreten eine Gruppe von Menschen, hauptsächlich Studenten und Professoren der slawischen Universität. Unter uns sind Studenten aus verschiedenen Fachbereichen: Linguistik, Chemie, Biologie usw. Wir versuchen,

> interessante Veröffentlichungen im Internet zu finden, die für uns und unsere Universitätskollegen relevant sein können. Manchmal finden wir Artikel selbst; andere Male helfen uns unsere Professoren, das Material auszuwählen. Nachdem wir die Erlaubnis von Autoren erhalten haben, übersetzen wir Artikel und veröffentlichen sie in unserem Blog, der für unsere Kollegen und Freunde zugänglich ist. Diese übersetzten Veröffentlichungen helfen Studenten oft in ihrem täglichen Studienalltag.

25.23 Spanisch

Alfonso de la Guarda Reyes (alfonsodg@ictechperu.net), Gustavo Echeverria (gustavo.echeverria@gmail.com), David Crespo Arroyo (davidcrespoarroyo@hotmail.com) und Cristian Bermudez Serna (crisbermud@hotmail.com) haben sich freiwillig gemeldet, das Buch ins Spanische zu übersetzen.

Gustavo Echeverria sagt:

> Ich arbeite als Softwareingenieur in Argentinien. Ich benutze hauptsächlich C# und .Net-Technologien bei der Arbeit, aber ausschließlich Python oder Ruby in meinen persönlichen Projekten. Ich lernte Python vor vielen Jahren kennen und war sofort begeistert. Nicht lange nachdem ich Python kennengelernt hatte, entdeckte ich dieses Buch, und es half mir, die Sprache zu lernen. Dann meldete ich mich freiwillig, das Buch ins Spanische zu übersetzen. Nun habe ich, nachdem ich einige Anfragen erhalten habe, begonnen, „A Byte of Python“ mit Hilfe von Maximiliano Soler zu übersetzen.

Cristian Bermudez Serna sagt:

> Ich bin Student der Telekommunikationstechnik an der Universität von Antioquia (Kolumbien). Vor einigen Monaten begann ich, Python zu lernen, und fand dieses wunderbare Buch, also meldete ich mich freiwillig, um die spanische Übersetzung zu erstellen.

25.24 Schwedisch

Mikael Jacobsson (leochingkwake@gmail.com) hat sich freiwillig gemeldet, das Buch ins Schwedische zu übersetzen.

25.25 Türkisch

Türker SEZER (tsezer@btturk.net) und Bugra Cakir (bugracakir@gmail.com) haben sich freiwillig gemeldet, das Buch ins Türkische zu übersetzen. „Wo ist die türkische Version? Wir würden sie so gerne lesen.“

25.26 Persisch

Najmeh Ghaderi (najmeh.gh.7.2008@gmail.com) hat sich freiwillig gemeldet, das Buch ins Persische zu übersetzen. Die Übersetzung hat gerade erst begonnen und ist derzeit in Arbeit.

CHAPTER 26

Übersetzungsanleitung

1. Der vollständige Quellcode des Original-Buches ist unter https://github.com/swaroopch/byte-of-python verfügbar.
2. Bitte forken Sie das Repository. (Legen Sie sich bei Github einen Account an und klicken Sie auf den "Fork" Button open rechts)
3. Laden Sie das Repository anschließend auf Ihren Computer herunter. Sie benötigen dafür Kenntnisse in der Verwendung von Git (bzw. Sie klicken auf den grünen "Code" sodaß er aufklappt und dann auf "Download as zip")
4. Bearbeiten Sie die `.md`-Dateien, um sie in Ihre Sprache zu übersetzen. Idealerweise übersetzten Sie auch die `.py` und die `.txt`-Dateien im Unterordner `programs`
5. Aus den übersetzten Dateien bilden Sie eine vollständige Website, ein pdf oder ein Ebook im Epub Format. Dies ist auf mehrere Arten möglich, z.B. mittels Honkit oder Sphinx
6. Informationen zum Generieren der Website, des PDFs und des EPUBs finden Sie in der Datei INSTALL.md.

CHAPTER 27

Feedback

Das Buch benötigt die Unterstützung seiner Leser:innen, wie Sie es sind, um auf Teile des Buches hinzuweisen, die nicht gut, nicht verständlich oder schlichtweg falsch sind.

Bitte schreiben Sie an den Hauptautor Swaroop CH für Feedback und für Vorschläge zur (englischen) Originalversion oder an den Übersetzer Horst JENS für Feedback und für Vorschläge zur deutschen Übersetzung.

Besser noch, falls Sie einen Account bei <Github.com> haben, machen Sie direkt *Pull-Requests* oder schreiben Sie *issues* auf der Github-Projektseite:

- englisches Original: https://github.com/swaroopch/byte-of-python
- deutsche Übersetzung: https://github.com/horstjens/byte-of-python-german