Python
Der Grundkurs

TEIL I

Python lernen

Kapitel 1

Hello, World!

Traditionell beginnt fast jeder Programmierkurs mit einem *Hello-World*-Programm. Die Aufgabe dieses simplen Programms ist es, die Zeichenkette `Hello, World!` auf den Bildschirm auszugeben.

Das klingt trivial – und ist es natürlich auch. In Wirklichkeit geht es bei *Hello, World!* gar nicht um den erforderlichen Programmcode, sondern vielmehr darum, dass Sie das zur Programmentwicklung notwendige Umfeld einrichten.

Bevor Sie loslegen können, müssen Sie nur zwei Dinge erledigen: Die Programmiersprache Python installieren und sich für einen geeigneten Editor oder eine Entwicklungsumgebung entscheiden. In diesem Kapitel erläutere ich Ihnen die Vorgehensweise für verschiedene Betriebssysteme.

1.1 Python installieren

Linux

Beginnen wir mit dem einfachsten Fall: Unter Linux ist Python fast ausnahmslos schon installiert. Sie müssen nur noch feststellen, um welche Version es sich handelt. Dazu öffnen Sie ein Terminalfenster und führen das folgende Kommando aus (siehe Abbildung 1.1):

```
python3 --version
  Python 3.12.2
```

Achten Sie auf die richtige Schreibweise des `python`-Kommandos! Unter Linux war es lange Zeit üblich, das veraltete Python 2 (Kommando `python`) und die aktuelle Version 3 (Kommando `python3`) parallel zu installieren. In aktuellen Distributionen gibt es nur noch Python 3. Bei manchen Distribu-

tionen bezieht sich das Kommando `python` sogar schon auf Version 3. Viele Distributionen sind aber beim etablierten Kommando `python3` geblieben. Bei Debian und Ubuntu haben Sie die Wahl: Wenn Sie das Zusatzpaket `python-is-python3` installieren, können Sie anschließend gleichermaßen die Kommandos `python` und `python3` verwenden, um Python 3 auszuführen.

Abbildung 1.1 Test der Python-Installation unter Ubuntu

Python-Version

Für dieses Buch habe ich alle Beispielprogramme mit Python 3.12 getestet. Nahezu alle Beispiele laufen aber ebenso gut auf älteren Python-Installationen ab Version 3.7. Das liegt daran, dass Python in den kleinen Versionsnummernsprüngen jeweils nur um relativ wenige Details verändert bzw. um Zusatzfunktionen erweitert wird. Diese Features spielen für dieses Buch nur eine untergeordnete Rolle.

Es ist höchst unwahrscheinlich, dass Python 3 nicht installiert ist. Das Kommando `python3` liefert dann die Fehlermeldung `command not found`. Dieses Manko beheben Sie durch die Installation des betreffenden Pakets. Unter Ubuntu müssen Sie z. B. in einem Terminalfenster `sudo apt install python3` ausführen.

Windows

Unter Windows ist Python standardmäßig nicht installiert. Auf der folgenden Webseite finden Sie Links zu den gerade aktuellsten Python-Releases für Windows:

https://python.org/downloads/windows

Nachdem Sie die gewünschte Versionsnummer ausgewählt haben, müssen Sie sich noch für das richtige Installationsprogramm entscheiden. In der Regel ist das der *Windows installer (64 bit)*.

Achten Sie darauf, dass Sie im Installationsprogramm die Option ADD PYTHON.EXE TO PATH aktivieren (siehe Abbildung 1.2), bevor Sie auf INSTALL NOW klicken! Diese Option stellt sicher, dass Sie Python später unkompliziert im Terminal starten können.

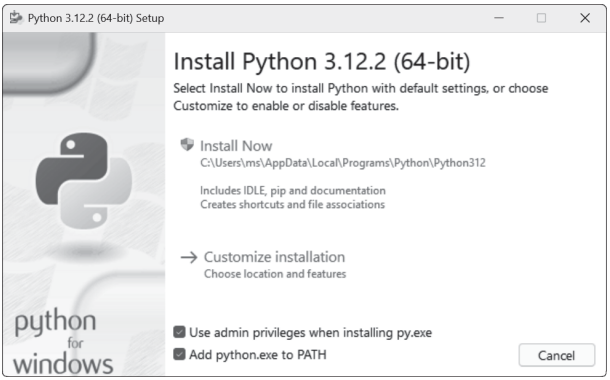

Abbildung 1.2 Python-Installationsprogramm für Windows

Nachdem die eigentliche Installation abgeschlossen ist, haben Sie die Möglichkeit, das Windows-typische Limit von 260 Zeichen für den Start von Kommandos aufzuheben. Wenn ein Python-Programm mit vielen Parametern aufgerufen wird, kann dieses Limit Fehler verursachen. Klicken Sie daher auf den Button DISABLE PATH LENGTH LIMIT. (Der Button wird nicht angezeigt, wenn das Limit schon bei einer früheren Installation oder auf einem anderen Weg deaktiviert wurde.)

Nach der Installation öffnen Sie ein Terminalfenster und verifizieren, ob sich Python starten lässt. Beachten Sie, dass der Kommandoname einfach `python` lautet (nicht `python3` wie unter Linux und macOS):

```
python --version
  Python 3.12.2
```

Vermeiden Sie Mehrfachinstallationen!

Unter Windows ergeben sich oft Probleme durch die parallele Installation mehrerer Python-Versionen. Der Microsoft Store sowie diverse Editoren und Entwicklungsumgebungen bieten Ihnen an, die jeweils neueste Version von Python zu installieren. Wenn Sie unbedacht einwilligen, ist Python mehrfach installiert. Welche Version dann tatsächlich zum Einsatz kommt, hängt von der Einstellung der PATH-Variablen und von der Konfiguration des Editors ab. Mit etwas Pech installieren Sie mit `pip` Zusatzmodule für die eine Version von Python, während Sie Ihr Script in der anderen Version ausführen. Das Script meldet dann einen Fehler, weil es das Modul nicht findet.

Meine Empfehlung für solche Fälle: Entfernen Sie zuerst sämtliche Python-Installationen, und führen Sie die Installation der gerade aktuellen Version dann *einmal* wie vorhin beschrieben aus.

macOS

Den einfachsten Weg, Python 3 unter macOS zu installieren, bietet das grafische Installationsprogramm (`*.pkg`-Datei, siehe Abbildung 1.3). Sie finden es auf der Python-Downloadseite:

https://python.org/downloads

Nach der Installation überzeugen Sie sich wie unter Linux in einem neu gestarteten Terminalfenster davon, dass alles geklappt hat:

```
python3 --version
  Python 3.12.2
```

Immer noch im Terminalfenster führen Sie schließlich dieses Kommando aus, wobei Sie gegebenenfalls `3.12` durch eine neuere Python-Versionsnummer ersetzen:

```
/Applications/Python\ 3.12/Install\ Certificates.command
```

Damit erreichen Sie, dass Root-Zertifikate installiert werden, mit denen Python-Programme die HTTPS-Verschlüsselung verifizieren können. Hin-

tergrundinformationen zu diesem Schritt geben die Readme-Datei, die während der Installation angezeigt wird, sowie Abschnitt 16.1, »Download und Upload von Dateien«.

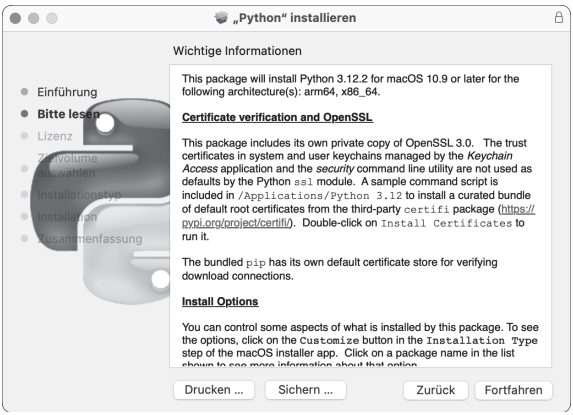

Abbildung 1.3 Installation von Python unter macOS

1.2 »Hello, World!« in der Python-Shell

Bei vielen Programmiersprachen müssen Sie zuerst in einem Editor Code verfassen, bevor Sie diesen ausprobieren können. Eine Besonderheit von Python besteht darin, dass Sie damit auch einzelne Anweisungen interaktiv ausführen können. Dazu öffnen Sie ein Terminalfenster und führen darin das Kommando `python` oder `python3` aus. In jedem Fall gelangen Sie nun in einen Kommandointerpreter, der oft auch als *Shell* bezeichnet wird.

Die zentrale Aufgabe dieses Kapitels, nämlich `Hello, World!` auszugeben, gelingt dort durch die simple Eingabe von `print("Hello, World!")` (siehe Abbildung 1.4).

Um die Python-Shell zu beenden, drücken Sie unter Linux oder macOS einfach Strg+D. Unter Windows gelingt das Kunststück mit Strg+Z und ↵. Auf allen Betriebssystemen funktioniert außerdem `exit()`.

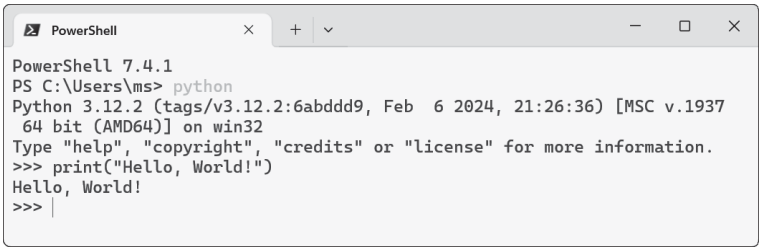

Abbildung 1.4 »Hello, World!« in einer Python-Shell, die hier in einem Terminal unter Windows ausgeführt wird

Python in der Shell kennenlernen

Die Python-Shell ist mehr als eine nette Spielerei für Python-Einsteiger. Auch Profis starten oft die Shell, um dort interaktiv einige Anweisungen auszuprobieren. Bei fehlerhaften Eingaben können Sie mit den Cursortasten durch die bisherigen Anweisungen scrollen, diese korrigieren und mit ↵ erneut ausführen.

Sie können in der Shell eigene Variablen definieren. Ausgaben gelingen sogar ohne `print`, weil der Interpreter das Ergebnis eines Ausdrucks ohnedies automatisch anzeigt:

```
>>> name='Michael'
>>> 'Hallo ' + name + '!'
'Hallo Michael!'
```

Die Shell eignet sich wunderbar, um die beachtlichen Rechenkünste von Python zu zeigen. Für ganze Zahlen gibt es keine Einschränkung des Zahlenraums, wie die Berechnungen von 2^5, 2^{100} und 2^{1000} mit dem Operator `**` beweisen:

```
>>> 2**5
32
>>> 2**100
1267650600228229401496703205376
>>> 2**1000
10715086071862673209484250490600018105614048117055336074437
```

```
75038837035105112493612249319837881569585812759467291755314
46825187145285692314043598457757469857480393456777482423094
85421074605062371141877954182153046474983581941267398767554
91655439460770629145711964776865421676604298316526243868372
05668069376
```

Auch zum Kennenlernen von Zeichenketten bietet sich die Shell an. (Details zum Umgang mit Zeichenketten folgen in Kapitel 6.)

```
>>> s='Lernen Sie Python kennen!'
>>> len(s)      # Länge der Zeichenkette ermitteln
25
>>> s.upper()   # in Großbuchstaben umwandeln
'LERNEN SIE PYTHON KENNEN!'
>>> s.lower()   # in Kleinbuchstaben umwandeln
'lernen sie python kennen!'
>>> s.find('Python')     # suchen und ...
11
>>> s.replace('e', 'x')  # ersetzen
'Lxrnxn Six Python kxnnxn!'
>>> s[5:10]              # Zeichenketten ausschneiden
'n Sie'
```

Ebenso spielerisch können Sie sich mit Listen oder anderen elementaren Python-Datenstrukturen anfreunden (siehe Kapitel 8, »Listen, Tupel, Sets und Dictionaries«):

```
>>> lst = [1, 2, 4, 8, 16]   # eine fünfteilige Liste
>>> lst[2]                   # das dritte Element
                             # (die Zählung beginnt mit 0)
4
>>> lst.extend([32])         # Element am Ende hinzufügen
>>> lst
[1, 2, 4, 8, 16, 32]
                             # die Formel x -> x*2+1 auf
                             # alle Elemente anwenden
>>> list(map(lambda x: x*2 + 1, lst))
[3, 5, 9, 17, 33, 65]
```

Sie können sogar mehrzeilige Anweisungen eingeben, z. B. `for`-Schleifen (siehe Kapitel 9, »Verzweigungen und Schleifen«). Dabei sind zwei Dinge zu beachten: Zum einen müssen die Anweisungen innerhalb der Schleife durch Leerzeichen eingerückt werden, und zum anderen müssen Sie die gesamte Eingabe durch *zweimaliges* ↵ abschließen. Der Python-Interpreter stellt der ersten Zeile >>> voran, bei allen Folgezeilen erscheinen drei Punkte:

```
>>> for i in range(4):
...     print(i)
...
0
1
2
3
```

Codedarstellung in diesem Buch

Sie können (und sollten!) viele Beispiele dieses Buchs direkt im Python-Interpreter ausprobieren. In den weiteren Listings im Buch verzichte ich aber darauf, Eingaben durch >>> zu kennzeichnen. Vielmehr werden Ausgaben ein wenig eingerückt, z. B. so:

```
print('Hello, World!')
   Hello, World!
```

1.3 »Hello, World!« als eigenständiges Programm

Nachdem Sie Python nun ein wenig kennengelernt haben, besteht Ihr nächstes Ziel darin, ein eigenständiges Programm zu entwickeln, das `Hello, World!` auf dem Bildschirm ausgibt. Im Prinzip reicht dazu ein beliebiger Editor wie Notepad++. Allerdings müssen Sie sich dann selbst um die Ausführung des Codes kümmern, was ein wenig mühsam ist (siehe auch Abschnitt 14.1, »Python-Scripts ausführen«).

Viel angenehmer ist es, Python mit einem für diese Sprache optimierten Programm zu erlernen: Ich empfehle Ihnen dafür den kostenlosen Editor *Visual Studio Code* (im Folgenden kurz *VSCode*). Wenn Sie später in die Profiliga aufsteigen, werden Sie sich vielleicht mit einer richtigen Python-Entwicklungsumgebung (z. B. *PyCharm*) oder mit den *Jupyter-Notebooks* (siehe Kapitel 20, »Wissenschaftliche Anwendung«) anfreunden.

VSCode samt Python-Erweiterung installieren

Installationsdateien für das Programm VSCode finden Sie für alle gängigen Betriebssysteme auf der folgenden Website:

https://code.visualstudio.com/download

VSCode ist ein universeller Editor für alle erdenklichen Programmiersprachen. Sobald Sie ein erstes Python-Programm unter `<name>.py` speichern oder eine fremde Python-Datei öffnen (z. B. eine Beispieldatei zu diesem Buch), bietet VSCode Ihnen an, die Python-Erweiterung zu installieren. Dieser Empfehlung sollten Sie unbedingt folgen.

Die Python-Erweiterung bringt zwei wesentliche Vorteile mit sich:

- VSCode »versteht« damit Ihren Python-Code besser, hebt verschiedene Codekomponenten klarer hervor und hilft bei Codeumbauten.
- VSCode sucht auf Ihrem Rechner nach der Python-Installation und blendet einen RUN-Button ein, mit dem Sie Ihr Programm bequem mit einem Klick ausführen können.

VSCode-Schnelleinstieg

Dieses Buch ist nicht der richtige Ort für eine VSCode-Einführung. Im Internet finden Sie bei Bedarf eine Menge Tutorials. Ich beschränke mich hier auf zwei Tipps:

- VSCode »denkt« in Verzeichnissen. Erstellen Sie auf Ihrem Rechner ein Verzeichnis für Ihre ersten Python-Experimente, unter Windows am besten innerhalb von `Dokumente`. In VSCode führen Sie dann FILE • OPEN FOLDER aus, um Ihr Verzeichnis zu öffnen. In der Folge können Sie alle Dateien in diesem Verzeichnis bequem bearbeiten. Um später Dateien

aus einem anderen Verzeichnis zu öffnen, wiederholen Sie File • Open Folder.

- Verwenden Sie beim Speichern neuer Codedateien die Kennung `*.py`. Spätestens dann erkennt VSCode, dass die Datei Python-Code enthält.

Das minimalistische Hello-World-Programm

Nachdem Sie in VSCode eine neue, noch leere Datei erzeugt haben, geben Sie die folgende Zeile ein und speichern die Datei als `HelloWorld.py`:

```
print('Hello, World!')
```

Sofern Sie die Python-Erweiterung installiert haben, können Sie das Programm mit dem Run-Button starten (siehe Abbildung 1.5).

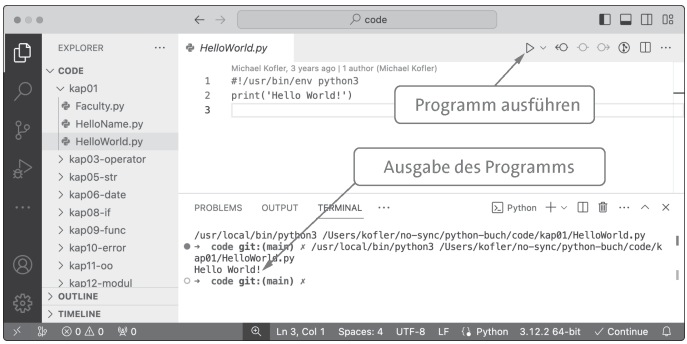

Abbildung 1.5 »Hello, World!« im Editor »VSCode« unter macOS

Shebang-Zeile

Alle Beispieldateien zu diesem Buch beginnen mit der folgenden Zeile:

```
#!/usr/bin/env python3
```

In Abschnitt 14.1, »Python-Scripts ausführen«, erkläre ich Ihnen, dass diese Zeile den merkwürdigen Namen »Shebang« hat. Der Shebang ist erforderlich, um Python-Programme unter Linux und macOS eigenstän-

dig (ohne den Run-Button des Editors) auszuführen. Solange Sie das nicht vorhaben, können Sie auf diese Anweisung verzichten. Windows ignoriert die Zeile, dort ist nur die Kennung `*.py` wichtig.

Noch ein Beispiel

Selbst für Kapitel 1 ist die simple Ausgabe von `Hello, World!` ein wenig langweilig. Starten Sie nochmals einen Editor, geben Sie den folgenden Code ein, und speichern Sie das Programm unter dem Namen `HelloName.py`:

```
# Beispieldatei HelloName.py
import time, locale
name = input('Geben Sie Ihren Namen an: ')
print('Hallo %s!' % name)
# Datum und Zeit in deutscher Lokalisierung
locale.setlocale(locale.LC_ALL, 'de_DE')  # Linux, macOS
locale.setlocale(locale.LC_ALL, 'german') # Windows
time = time.strftime('Heute ist %A, der %d. %B.')
print(time)
```

Wenn Sie das Programm ausführen, fragt es zuerst nach Ihrem Namen. Nachdem Sie ihn eingegeben haben, begrüßt das Programm Sie und zeigt das aktuelle Datum an (siehe Abbildung 1.6). Der Programmcode enthält eine Menge Anweisungen, die Sie noch nicht kennen, deren Bedeutung aber leicht zu verstehen ist:

- `import` liest (aktiviert) zwei Python-Erweiterungen, die *Module* genannt werden. `time` stellt Funktionen zum Umgang mit Datum und Uhrzeit zur Verfügung. `locale` kümmert sich um die Lokalisierung, also um die Anpassung an landessprachliche Besonderheiten.
- Die Funktion `input` gibt die angegebene Zeichenkette am Bildschirm aus und nimmt dann eine Eingabe entgegen. Im Beispielprogramm wird diese Eingabe in der Variablen `name` gespeichert.
- `print` gibt `Hallo %s!` aus, wobei `%s` aber durch den Inhalt der nachfolgend genannten Variablen ersetzt wird.

- `locale.setlocale(...)` sagt Python, dass es deutsche Spracheinstellungen anwenden soll. Ohne diese Anweisung verwendet Python standardmäßig eine englische Lokalisierung und würde den Wochentag auf Englisch ausgeben.
- `time.strftime(...)` ist eine Methode, die Datums- und Zeitdaten in der jeweiligen Landessprache formatiert. Das Ergebnis wird in einer zweiten Variablen, `time`, gespeichert.
- `print` gibt schließlich den Inhalt dieser Variablen aus.

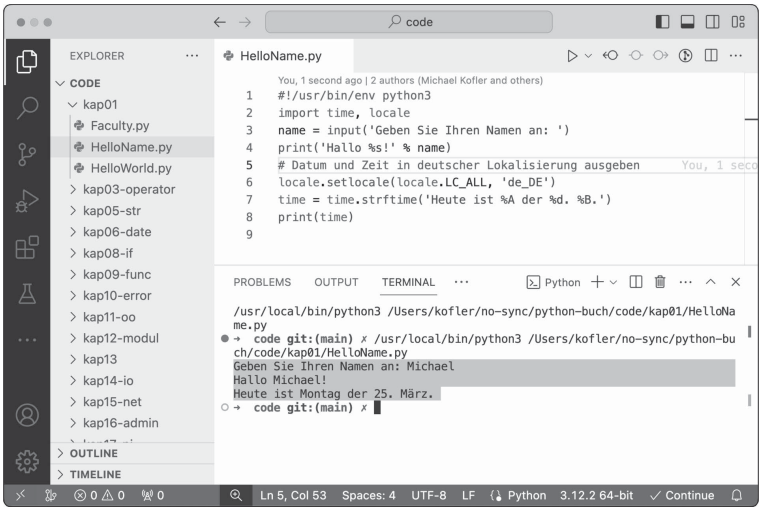

Abbildung 1.6 Ein etwas komplexeres Programm in »VSCode«

Selbstverständlich werden Sie alle hier aufgezählten Sprachelemente von Python in den folgenden Kapiteln noch im Detail kennenlernen.

Beispieldateien zum Buch

Viele Listings in diesem Buch sind nur wenige Zeilen lang. Sie können die Anweisungen direkt im Python-Interpreter ausprobieren.

Daneben gibt es aber auch längere Listings, die wie das vorige Beispiel in einer Datei zu speichern sind. Solche Listings beginnen mit dem Kommentar `# Beispieldatei name.py`. Das bedeutet, dass Sie sich die Tipparbeit sparen können. Sie finden die entsprechende Datei in den Beispieldateien zum Buch im Verzeichnis für das betreffende Kapitel (also z. B. in `kap06` für die Beispiele zu Kapitel 6, »Zeichenketten«). Die Beispieldateien zum Buch können Sie hier herunterladen:

https://www.rheinwerk-verlag.de/5882

»Richtige« Entwicklungsumgebungen

Wenn Sie Python gerade lernen, empfehle ich Ihnen, mit VSCode zu arbeiten oder vielleicht Thonny auszuprobieren (siehe *https://thonny.org*). Sie haben aber vielleicht schon gehört, dass fortgeschrittene Programmierer sogenannte *Entwicklungsumgebungen* (*Integrated Development Environments*, IDEs) verwenden, z. B. Visual Studio für C#, Eclipse für Java oder Xcode für Swift. Gibt es derartige Programme auch für Python?

Tatsächlich stehen eine Menge IDEs für Python zur Auswahl. Die folgende Webseite zählt rund 30 Programme auf. Bei einem Teil davon handelt es sich »nur« um Erweiterungen für Entwicklungsumgebungen, die ursprünglich für andere Sprachen konzipiert wurden. Viele der aufgezählten Programme sind kostenlos verfügbar; einige IDEs sind hingegen kommerzielle Programme.

https://wiki.python.org/moin/IntegratedDevelopmentEnvironments

IDEs sind bei komplexen Projekten, die aus mehreren Dateien bestehen und die viele externe Module einbinden, eine große Hilfe. Gute IDEs unterstützen Sie bei der Codeeingabe, machen die Python-Dokumentation rasch zugänglich und helfen bei der Fehlersuche. Zum Lernen von Python und für die in diesem Grundkurs präsentierten, durchwegs sehr einfachen Beispielprogramme kommen Sie aber gut ohne eine Python-Entwicklungsumgebung aus.

Ich rate Ihnen daher, mit der Auswahl und dem Einsatz einer IDE abzuwarten, bis Sie etwas Routine mit Python gewonnen haben. Andernfalls kann

es gut passieren, dass Sie viel Zeit investieren bzw. vergeuden, um sich mit den vielfältigen Funktionen diverser IDEs auseinanderzusetzen. Momentan sollte hingegen das Lernen von Python im Vordergrund stehen.

1.4 Elementare Syntaxregeln

Bevor ich in den folgenden Kapiteln im Detail verschiedene Sprachelemente von Python erläutere, möchte ich an dieser Stelle einen ersten Überblick über die Syntax von Python geben.

Anweisungen

Python-Anweisungen sind normalerweise einzeilig. Sie werden im Gegensatz zu vielen anderen Programmiersprachen nicht durch einen Strichpunkt oder ein anderes Zeichen abgeschlossen.

Mehrzeilige Anweisungen sind erlaubt, wenn ihr Anfang und Ende durch Klammern eindeutig hervorgeht, z. B. aufgrund offener Klammern. Wenn Python die mehrzeilige Struktur nicht erkennt (was leider der Regelfall ist), müssen Sie die Anweisungen mit dem Trennzeichen \ bilden:

```
print('abc',
      'efg')
a = 1 + 2 + \
    3 + 4
```

Anweisungen dürfen mit einem Strichpunkt abgeschlossen werden. Normalerweise ist dieser Strichpunkt optional und hat keine Auswirkungen auf die Programmausführung. Strichpunkte erlauben es aber, mehrere Anweisungen in einer Zeile zu formulieren:

```
a=1; b=2; c=3
```

Die obige Dreifachzuweisung können Sie auch auf eine andere Art durchführen, indem Sie sowohl die Variablen als auch die Werte in Gruppen angeben, deren Bestandteile jeweils durch Kommas getrennt werden.

Python-intern werden dabei Tupel gebildet. Beide Varianten sind *richtig*, aber die zweite Variante entspricht eher den Sprachkonzepten von Python.

```
a, b, c = 1, 2, 3
```

Blockelemente

In Python gibt es wie in jeder anderen Programmiersprache Sprachelemente, die einen ganzen Block weiterer Anweisungen einleiten, z. B. Verzweigungen mit `if`, Schleifen mit `for` und `while` oder Funktionsdefinitionen mit `def`. In Python sind derartige Sprachelemente immer mit einem Doppelpunkt gekennzeichnet. Alle weiteren Anweisungen, die zum entsprechenden Block gehören, müssen eingerückt werden. Dafür entfallen die in anderen Sprachen üblichen Klammern. Also:

```
if xxx:
    anweisung1a
    anweisung1b
else:
    anweisung2a
    anweisung2b
    anweisung2c
```

Wenn die Bedingung `xxx` erfüllt ist, werden die Anweisungen 1a und 1b ausgeführt; ist sie nicht erfüllt, werden stattdessen die Anweisungen 2a, 2b und 2c ausgeführt. Mehr Details zu `if` und `else` folgen in Kapitel 9, »Verzweigungen und Schleifen«.

Entscheidend ist in Python, dass die Codeeinrückung nicht wie bei anderen Programmiersprachen optional ist, sondern Teil der Syntax!

Richtig einrücken

Für das Ausmaß der Einrückung gibt es keine starren Regeln: Ein Zeichen reicht, empfohlen werden wegen der besseren Lesbarkeit vier Zeichen. Bei einigen Listings in diesem Buch beträgt die Einrückung aus Platzgründen nur zwei Zeichen.

Nicht empfohlen ist die Verwendung von Tabulatorzeichen. Die meisten Editoren können so konfiguriert werden, dass zum Einrücken immer Leerzeichen verwendet werden. Wenn Sie doch Tabulatoren verwenden, nimmt Python an, dass sich die Tabulatorpositionen an Vielfachen von acht Zeichen befinden.

Code darf auch direkt nach einem Blockelement angegeben werden. In einfachen Fällen lassen sich so einzeilige Bedingungen oder Schleifen formulieren:

```
if xxx: anweisung
```

Auf diese Weise lassen sich auch mehrere Anweisungen in einer Zeile ausführen:

```
if xxx: anweisung1; anweisung2; anweisung3
```

»print«

Beim Kennenlernen von Python sowie in ersten Testprogrammen ist die `print`-Funktion allgegenwärtig. Damit können Sie unkompliziert Variableninhalte oder Testnachrichten ausgeben. Die Bedeutung von `print` wird aber im Laufe der Zeit nachlassen: Je intensiver Sie programmieren, desto seltener werden Sie Ausgaben mit `print` durchführen – entweder schreiben Sie Resultate direkt in Dateien, oder Sie verwenden eine grafische Benutzeroberfläche zur Interaktion mit den Anwendern.

Die Syntax von `print` ist einfach: Sie übergeben einen oder mehrere Parameter in runden Klammern an die Funktion. `print` wandelt jeden der Parameter in Zeichenketten um und gibt alle Zeichenketten aus. Dabei wird zwischen den Parametern jeweils ein Leerzeichen und am Ende ein Zeilenumbruchzeichen gesetzt, sodass jede `print`-Anweisung in einer neuen Zeile startet. `print` ist also sehr unkompliziert zu verwenden und kommt mit nahezu jeder Art von Python-Objekt zurecht, also auch mit Listen, Tupeln und Sets. Probieren Sie es im Python-Interpreter aus!

```
>>> print(1, 2, 3/4, 'abc', 2==3)
1 2 0.75 abc False
```

```
>>> print('1/7 ist', 1/7)
1/7 ist 0.14285714285714285
>>> x, y, z = ['eine', 'Liste'], ('ein', 'Tupel'),
              {'ein', 'Set'}
>>> print(x, y, z)
['eine', 'Liste'] ('ein', 'Tupel') {'ein', 'Set'}
```

`print` kennt drei optionale Parameter:

- `sep` stellt die Zeichenkette ein, die zwischen den Parametern ausgegeben wird – standardmäßig `' '`.
- `end` definiert die Zeichenkette, die nach dem letzten Parameter ausgegeben wird – standardmäßig `'\n'` (also »neue Zeile«).
- `file` bestimmt, wo die Ausgabe durchgeführt wird. Normalerweise werden die Ausgaben zur Standardausgabe umgeleitet, also am Bildschirm angezeigt. `file` gibt Ihnen die Möglichkeit, die Ausgaben in eine Textdatei zu schreiben.

```
>>> print(1, 2, 3, sep='---')
1---2---3
>>> print(1, 2, 3, sep=';', end='.\nEOF\n')
1;2;3.
EOF
>>> f = open('out.txt', 'w')
>>> print(1, 2, 3, file=f)
>>> f.close()
```

»input«

So wie Sie mit `print` Ausgaben in einem Terminalfenster durchführen können, verarbeitet `input` Texteingaben. `input` gibt zuerst den im optionalen Parameter angegebenen Text aus und erwartet dann eine Eingabe, die mit ↵ abgeschlossen werden muss.

```
name = input('Geben Sie Ihren Namen an: ')
print('Ihr Name lautet:', name)
```

Leere Eingaben, also ein ↵ ohne Text, quittiert `input` mit einem `EOFError`. Wenn Ihr Programm das `readline`-Modul lädt (durch `import readline` am Beginn des Codes), dann stehen bei wiederholten Eingaben Editierfunktionen zur Verfügung. Beispielsweise kann der Nutzer Ihres Programms dann mit den Cursortasten zuvor eingegebene Zeichenketten wiederverwenden und ändern.

`input` liefert immer Zeichenketten zurück. Wenn Sie mit der Eingabe rechnen wollen, müssen Sie sich um die Umwandlung in eine Zahl kümmern, z. B. mit `int` für ganze Zahlen:

```
n = int(input('Geben Sie eine Zahl ein: '))
print(n*2)
```

Auch in diesem Fall kommt es zu einem Fehler, wenn die Eingabe (z. B. `abc`) keine gültige Zahl ist.

Module und »import«

Für den Einstieg wirkt Python oft sehr groß und komplex, aber in Wirklichkeit ist die Anzahl der unmittelbar in Python implementierten Funktionen durchaus überschaubar. Alle erdenklichen Zusatzfunktionen sind nicht im Sprachkern von Python realisiert, sondern in Form von Modulen, die selbst in Python programmiert wurden.

Module müssen vor ihrer Verwendung importiert werden. Dafür gibt es diverse Syntaxvarianten, von denen ich hier nur die wichtigsten nenne. (Kapitel 13, »Module«, geht auf dieses Thema ausführlicher ein.)

- `import modulname`: Diese Anweisung liest das Modul. Anschließend können Sie alle darin definierten Funktionen in der Schreibweise `modulname.funktionsname()` nutzen. Mit `import m1, m2, m3` können Sie auch mehrere Module auf einmal importieren.
- `import modulname as m`: Bei dieser Variante können die im Modul definierten Funktionen in der Form `m.funktionsname()` verwendet werden. Bei langen Modulnamen minimiert das den Tippaufwand und macht den Code übersichtlicher.

- `from modulname import n1, n2`: Bei dieser Variante können Sie die Funktionen oder Klassen `n1` und `n2` ohne das Voranstellen des Modulnamens verwenden.
- `from modulname import *`: Diese Anweisung importiert alle Symbole aus dem angegebenen Modul, deren Name nicht mit `__` beginnt (also mit zwei Unterstrichen zur Kennzeichnung interner Symbole).

  Vorsicht: Bei dieser Variante kann es passieren, dass Sie unbeabsichtigt den Inhalt gleichnamiger Variablen überschreiben!

Python-intern bewirkt `import`, dass die Datei `<modulname>.py` gelesen und ausgeführt wird. Es ist üblich, `import`-Anweisungen immer an den Anfang eines Python-Scripts zu setzen.

Viele Module enthalten einfach die Definition diverser Funktionen; damit sind diese Funktionen Python nun bekannt und können genutzt werden. Module können aber auch Code enthalten, der sofort ausgeführt wird, beispielsweise um Initialisierungsarbeiten durchzuführen.

Namenskonflikte zwischen Ihrem Script und einem Modul

Vermeiden Sie im lokalen Verzeichnis Dateinamen, die mit den Modulnamen übereinstimmen, die Sie verwenden!

Wenn Sie beispielsweise in einem Python-Script `import csv` ausführen und es im lokalen Verzeichnis die Datei `csv.py` gibt, dann wird diese Datei anstelle des gewünschten Python-Moduls zur Verarbeitung von CSV-Dateien importiert.

Kommentare

Einfache Kommentare werden mit dem Zeichen # eingeleitet und reichen bis zum Ende der Zeile:

```
# ein Kommentar
print('abc')  # noch ein Kommentar
```

Mit """ bzw. ''' können Sie mehrzeilige Kommentare bilden:

```
""" ein langer
    Kommentar """
```

Genau genommen dient """ zur Bildung mehrzeiliger Zeichenketten. Tatsächlich ist auch der obige Kommentar eine solche Zeichenkette – aber eine, die im Code zwar definiert, jedoch nicht genutzt wird.

Wenn mit """ eingeleitete Kommentare richtig platziert sind (z. B. unmittelbar nach der Definition einer Funktion oder einer Klasse), gelten sie als sogenannte *Docstrings* und werden vom Python-internen Dokumentationssystem ausgewertet. Auf die Details gehe ich in diesem Buch nicht ein, Sie können sie im Internet nachlesen:

https://python.org/dev/peps/pep-0257
https://en.wikipedia.org/wiki/Docstring

1.5 Wiederholungsfragen

- **W1:** Python-Scripts werden durch einen Interpreter ausgeführt. Was bedeutet das?
- **W2:** Sind in Python mehrzeilige Anweisungen erlaubt?
- **W3:** Wie können Sie mehrere Anweisungen in einer Zeile ausführen?
- **W4:** Welche Bedeutung hat eingerückter Code?
- **W5:** Wie können Sie bei `print` den Zeilenumbruch nach der Ausgabe verhindern?
- **W6:** Was sind Module, und wie werden sie verwendet?
- **W7:** Wie werden in Python Kommentare formuliert?

Kapitel 16

Netzwerkfunktionen

In diesem Kapitel stelle ich Ihnen einige Netzwerkfunktionen von Python vor. Dabei konzentriere ich mich auf die folgenden Themen:

- Down- und Upload von Dateien (HTTP, HTTPS, FTP)
- REST-APIs nutzen
- Mails versenden

Wie in den anderen Kapiteln kann ich auch hier nur einige besonders wichtige Funktionen herausgreifen. Eine vollständige Beschreibung aller Netzwerkfunktionen und -module würde ein ganzes Buch füllen. Ein guter Startpunkt für eigene Recherchen sind die folgenden Seiten:

https://docs.python.org/3/library/ipc.html (Low Level)
https://docs.python.org/3/library/internet.html (High Level)
https://wiki.python.org/moin/UsefulModules#Networking

16.1 Download und Upload von Dateien

Das Modul `urllib.request` stellt diverse Funktionen und Klassen für den Download von Dateien via HTTP oder HTTPS zur Verfügung. In einem XML-Beispiel im vorigen Kapitel habe ich die einfachste (und vermutlich populärste) Anwendungsvariante bereits gezeigt:

```
# Beispieldatei hello-download.py
import urllib.request
url      = 'https://kofler.info'
response = urllib.request.urlopen(url)
binary   = response.read()        # Download durchführen
txt      = binary.decode('utf-8') # als Text interpretieren
```

Um ein Dokument von einem HTTP- oder HTTPS-Server herunterzuladen, übergeben Sie seine Adresse (*Uniform Resource Locator*, kurz URL) an die Methode `urlopen`. Als Ergebnis erhalten Sie ein `HTTPResponse`-Objekt. Darauf wenden Sie die `read`-Methode an, um die HTML-Seite bzw. Datei vollständig herunterzuladen. `read` arbeitet synchron, das heißt, Ihr Programm wird erst nach dem Abschluss des Downloads fortgesetzt.

`read` liefert binäre Daten zurück (Datentyp `bytes`). Wenn es sich dabei um einen Text handelt, setzt die Umwandlung in eine Zeichenkette durch `binary.decode` voraus, dass Sie das Codierungsformat kennen. Ist das nicht der Fall, können Sie es mit `get_content_charset` ermitteln:

```
cs  = response.headers.get_content_charset()
txt = binary.decode(cs)
```

Beim Download und bei der Decodierung können alle möglichen Fehler auftreten (falsche URL, Timeout, keine Netzwerkverbindung, falscher Zeichensatz etc.). Daher sollten Sie entsprechenden Code immer mit `try/except` absichern.

Zertifikatsprobleme unter macOS

Bei der Kommunikation via HTTPS muss Python verifizieren, ob die Zertifikate der jeweiligen Website von einem autorisierten Dienst ausgestellt wurden. Die unter macOS standardmäßig installierten OpenSSL-Bibliotheken betrachtet Python als veraltet und nicht vertrauenswürdig. Stattdessen greift Python auf eine Zertifikatsammlung des `certifi`-Pakets zurück.

Aus technischen Gründen ist es während der Installation von Python unmöglich, dieses Paket zu installieren (siehe *https://bugs.python.org/issue29480*). Das macOS-Installationsprogramm von Python weist in einer Readme-Datei auf die Notwendigkeit einer manuellen Installation hin, aber erfahrungsgemäß wird dieser Text selten gelesen.

Wenn Sie also beim ersten Versuch, eine HTTPS-Verbindung herzustellen (z. B. mit der Methode `urlopen`), die SSL-Fehlermeldung *Certificate verify failed* erhalten, dann sind die fehlenden Root-Zertifikate die Ursache. Abhilfe:

Öffnen Sie ein Terminalfenster, und führen Sie dort das folgende Kommando aus:

```
/Applications/Python\ 3.12/Install\ Certificates.command
```

Große Dateien stückweise herunterladen

Das Codebeispiel hat mit `response.read()` das gesamte Dokument auf einmal heruntergeladen. Bei kleinen Dateien ist das zweckmäßig. Wenn Sie hingegen eine umfangreiche Datei Stück für Stück herunterladen möchten, ist es besser, an `read` die maximale Anzahl von Bytes zu übergeben, die Sie auf einmal verarbeiten möchten.

Das folgende Beispiel zeigt, wie Sie eine PDF-Datei stückweise via HTTPS herunterladen und in eine Datei speichern. Dabei werden immer wieder maximal 64 KiB heruntergeladen, bis `read` keine Ergebnisse mehr liefert. `print` gibt für jeden Schritt ein +-Zeichen auf dem Bildschirm aus. Die `flush`-Methode stellt sicher, dass die Ausgabe sofort erscheint. (Ohne `flush` würden die einzelnen Ausgaben zwischengespeichert, was zwar effizienter ist, aber die Idee eines visuellen Feedbacks zunichtemacht.)

```
# Beispieldatei download-chunk.py
import urllib.request
import sys
url       = 'https://hostname.../leseprobe.pdf'
chunksize = 64 * 1024  # 64 KiB
response  = urllib.request.urlopen(url)
with open('leseprobe.pdf', 'wb') as f:
    while True:
        chunk = response.read(chunksize)
        if not chunk:
            break
        f.write(chunk)
        # Feedback
        print('+', end='')
        sys.stdout.flush()
    print()
```

FTP

FTP ist ein veraltetes und unsicheres Protokoll, es ist aber immer noch weit verbreitet. Um eine Datei von einem FTP-Server herunterzuladen, verwenden Sie am besten das Modul `ftplib`.

Das folgende Beispielprogramm überträgt eine Datei von einem bzw. zu einem FTP-Server, der auf einem Raspberry Pi im lokalen Netzwerk läuft. Bevor Sie das Programm testen können, müssen Sie auf Ihrem Raspberry Pi mit `sudo apt install vsftpd` einen FTP-Server installieren und in dessen Konfigurationsdatei `/etc/vsftpd.conf` den Schreibzugriff erlauben.

Der Code ist leicht verständlich. Zuerst wird ein FTP-Objekt erzeugt, wobei der Hostname, der Login-Name und das Passwort übergeben werden. Anschließend lädt die Methode `retrbinary` die gewünschte Datei (hier `readme.txt`) vom FTP-Server herunter und speichert sie in einer lokalen Datei (Variable `file`).

Natürlich kann das Programm auch mit jedem beliebigen anderen FTP-Server kommunizieren. Die Variablen `hostname`, `user` und `pw` müssen Sie an Ihre Gegebenheiten anpassen. »Echten« Code sollten Sie mit `try/except` gegen mögliche Fehler absichern (Host nicht erreichbar, Login ungültig etc.).

```
# Beispieldatei ftp-pi.py
from ftplib import FTP
host = 'raspberry'
user = 'pi'
pw   = 'geheim'

# Datei readme.txt vom FTP-Server herunterladen
# und lokal speichern
ftp = FTP(host, user, pw)
fname = 'readme.txt'
with open(fname, 'wb') as file:
    ftp.retrbinary('RETR %s' % (fname), file.write)
```

Ein Upload erfolgt nach dem gleichen Schema, wobei Sie nun allerdings die Methode `storbinary` mit dem Kommando `STOR` ausführen:

```
fname = 'other.txt'
with open(fname, 'rb') as file:
    ftp.storbinary('STOR %s' % (fname), file)
```

Weitere FTP-Methoden fasst die `ftplib`-Dokumentation zusammen:

https://docs.python.org/3/library/ftplib.html

16.2 REST-APIs nutzen

Die Abfrage von Daten bei externen Diensten lässt sich besonders einfach mit einem *Application Programming Interface* realisieren. Durchgesetzt haben sich in den letzten Jahrzehnten sogenannte *REST-APIs*: Der *Representational State Transfer* beruht auf einem Datenaustausch über etablierte HTTP-Requests wie Get, Put oder Post. Sie senden also einen Get-Request und erhalten ein JSON-Dokument mit Informationen – den Wetterbericht, Börsendaten usw.

Es gibt im Internet eine Menge Dienste, die in der Grundform kostenlos sind (oftmals aber erst nach einer Registrierung). Zusatzfunktionen erfordern ein kostenpflichtiges Abo oder eine andere Form der Bezahlung. Manche Dienste ermöglichen auch die Speicherung von Daten mit Put oder Post.

Bei der Programmierung Ihrer Python-Scripts können Sie das im vorigen Abschnitt schon beschriebene `urllib`-Modul verwenden. Mehr Komfort bietet das Modul `requests`, das ich Ihnen hier kurz vorstelle. Das Modul hat den Nachteil, dass es extra installiert werden muss (siehe Abschnitt 13.3):

```
pip  install requests              # Windows
pip3 install requests              # macOS
sudo apt install python3-requests  # Debian, Ubuntu
sudo dnf install python3-requests  # Fedora, Red Hat
```

Requests ausführen

Nach `import requests` können Sie mit `get`, `put` usw. Requests ausführen. Die folgenden Zeilen zeigen einen einfachen Get-Request. Das binäre Ergebnis (`response.content`) wird mit `decode` in eine UTF-8-Zeichenkette umgewandelt und ausgegeben. Die eingesetzte Seite *https://httpbin.org* ist speziell zum Testen von REST-Requests gedacht.

```
# Beispielprogramm rest-httpbin.py
import requests
# Get-Request
response = requests.get('https://httpbin.org/get?q=123')
print(response.content.decode('utf-8'))
# {
#   "args": {
#     "q": "123"
#   }, ...
```

Wenn die REST-API ein JSON-Dokument zurückgibt, machen Sie daraus mit der integrierten Methode `json` einen Python-Objektbaum. (Sie brauchen also nicht auf das `json`-Modul zurückzugreifen.) Den HTTP-Status der Webserver-Antwort ermitteln Sie mit der Eigenschaft `status_code`.

```
response = requests.get('https://httpbin.org/get?q=123')
data = response.json()
print(data)
# {'args': {'q': '123'},
#  'headers': {'Accept': '*/*', ...
print(response.status_code)
# 200
```

Je nachdem, welche Art von Request Sie ausführen möchten, verwenden Sie anstelle von `requests.get` entsprechend `put`, `patch`, `delete` usw. An sämtliche Methoden können Sie diverse optionale Parameter übergeben:

- `header` erwartet ein Dictionary mit den Header-Daten.
- An `data` übergeben Sie wahlweise ein Dictionary mit Parametern (z. B. für einen Post-Request) oder eine Zeichenkette mit sonstigen Daten.

- Alternativ können Sie mit dem Parameter `json` ein Dictionary übergeben. Sein Inhalt wird im JSON-Format übertragen.
- Mit `files` können Sie lokale Dateien zum Server hochladen.

Die folgenden Zeilen zeigen einen Put-Request, bei dem Daten im JSON-Format zum Server übertragen werden. Um zu überprüfen, ob der Put-Request funktioniert, können Sie die Antwort ausgeben. Sie enthält eine Kopie der übergebenen Daten.

```
data = {'firstName': 'John', 'lastName': 'Doe'}
response = requests.put('https://httpbin.org/put',
                        json=data)
print(response.json())
```

Um einen Request mit einer Basic-Authentifizierung zu testen, können Sie bei *https://httpbin.org* Name und Passwort in die URL verpacken. Zur Authentifizierung übergeben Sie den Benutzernamen und das Passwort an die `auth`-Option. Wenn die Daten übereinstimmen, erhalten Sie den Status-Code 200 (OK).

```
url = 'https://httpbin.org/basic-auth/maria/topsecret'
response = requests.get(url, auth=('maria', 'topsecret'))
print("Status-Code:", response.status_code)
```

Beispiel: Aktuelles Wetter ermitteln

Das folgende Beispiel ermittelt das aktuelle Wetter an einem gegebenen Ort. Es greift auf *https://api.weatherapi.com* zurück. Diese API kann mit gewissen Einschränkungen kostenlos genutzt werden. Sie müssen sich allerdings einen API-Key besorgen. (Die Angabe Ihrer E-Mail-Adresse reicht aus, Sie müssen weder eine Kreditkartennummer noch sonstige persönliche Daten angeben.)

Ihr Programm muss nun eine Adresse zusammensetzen, die wie im folgenden Beispiel aussieht:

https://api.weatherapi.com/v1/current.json?key=1234&q=Graz&aqi=no

Mit dieser URL (*Uniform Resource Locator*) führen Sie einen Get-Request aus. Als Antwort erhalten Sie ein relativ verschachteltes JSON-Dokument wie im folgenden Listing:

```
{
  "location": {
    "name": "Graz",
    "region": "Steiermark",
    ...
    "localtime": "2024-03-28 9:27"
  },
  "current": {
    "temp_c": 9,
    "temp_f": 48.2,
    "condition": {
      "text": "Partly cloudy",
      ...
    },
    ...
  }
}
```

Das folgende Programm stellt die URL zusammen (siehe dazu auch Abschnitt 6.4, »Zeichenketten formatieren und konvertieren«) und extrahiert aus den JSON-Daten die Temperatur in Grad Celsius sowie die Beschreibung des aktuellen Wetters:

```
# Beispiel rest-weather.py
import requests
key  = "7901..."
city = "Graz"
base = "https://api.weatherapi.com/v1/current.json"
url  = "%s?key=%s&q=%s&aqi=no" % (base, key, city)
response = requests.get(url)
data = response.json()
condition = data['current']['condition']['text']
print("Das Wetter in %s: %s" % (city, condition))
print("Temperatur:", data['current']['temp_c'], "°C")
```

Bei meinem Test hat das Programm die folgende Ausgabe produziert:

```
Das Wetter in Graz: Partly cloudy
Temperatur: 9.0 °C
```

16.3 Mails versenden

Damit Sie aus einem Python-Script heraus Mails versenden können, benötigen Sie Zugang zu einem Mail-Server. Am einfachsten gelingt das, wenn auf Ihrem Rechner ohnedies ein konfigurierter Mail-Server läuft. Diese Voraussetzung ist allerdings nur auf (manchen) Linux-Servern erfüllt.

Um die E-Mail zusammenzustellen, verwenden Sie Funktionen bzw. Klassen aus dem `email`-Modul. Ich gehe hier davon aus, dass der Nachrichteninhalt aus reinem Text besteht (ohne HTML-Tags) und dass sowohl die Nachricht als auch das Thema (*Subject*) internationale Sonderzeichen enthalten darf. Beachten Sie aber, dass Sender und Empfänger bei dieser Variante ausschließlich aus ASCII-Zeichen zusammengesetzt werden müssen (also ohne ä, ö, ü oder ß).

Zum Versenden der Mail wird ein `SMTP`-Objekt erzeugt, das mit dem lokalen E-Mail-Server kommuniziert (`localhost`). Aus diesem Grund sind keine Authentifizierungsdaten erforderlich. Der Versand kann unmittelbar mit `send_message` initiiert werden. Wie üblich ist es zweckmäßig, den Code durch `try/except` abzusichern.

```
# Beispieldatei hello-mail.py
from email.mime.text import MIMEText
from email.header import Header
import smtplib
msg  = 'Lorem ipsum äöü ...'
subj = 'Die erste von Python versendete Mail äöü'
frm  = 'Absender <absender@host-abc.de>'
to   = 'Max Mustermann <max@mustermann.xyz>'
# E-Mail zusammenstellen
mail = MIMEText(msg, 'plain', 'utf-8')
mail['Subject'] = Header(subj, 'utf-8')
```

```
mail['From']    = frm
mail['To']      = to
# E-Mail versenden
smtp = smtplib.SMTP('localhost')
smtp.send_message(mime)
smtp.quit()
```

Mail-Versand an einen externen SMTP-Server

Um die E-Mail an einen externen SMTP-Server zu übergeben, der nicht auf demselben Rechner läuft wie Ihr Python-Script, erzeugen Sie abermals ein `SMTP`-Objekt. Mit `starttls` initiieren Sie eine verschlüsselte Verbindung mit dem Server. Anschließend führen Sie den Login durch und verwenden dann die Methode `sendmail`, um Ihre Nachricht zu versenden. Beachten Sie, dass Sie den Empfänger in eckige Klammern stellen müssen. `sendmail` erwartet im zweiten Parameter eine Liste mit allen Empfängern – auch wenn es wie in diesem Beispiel nur einen einzigen Empfänger gibt.

```
# Beispieldatei mail2.py
... wie bisher
# E-Mail mit lokalem Mail-Server versenden
smtp = smtplib.SMTP('smtp-server-hostname')
smtp.starttls()
smtp.login('loginname', 'password')
smtp.sendmail(frm, [to], mail.as_string())
smtp.quit()
```

Bei Bedarf können Sie mit `smtp.smtpport = n` explizit die Port-Nummer Ihres SMTP-Servers angeben. In der Regel verwendet das `email`-Modul aber automatisch den richtigen Port.

Google Mail

Die hier skizzierte Vorgehensweise funktioniert grundsätzlich auch mit Google Mail (Gmail). Anstelle des Accountpassworts müssen Sie allerdings ein sogenanntes *App-Passwort* verwenden, das Sie in den Kontoeinstellungen einrichten.

Dazu öffnen Sie das Dialogblatt Sicherheit und geben im Suchfeld »App-Passwörter« ein. App-Passwörter können nur verwendet werden, wenn Sie vorher die Zweifaktorauthentifizierung (2FA) aktiviert haben.

HTML-Mail

Viele Mails enthalten die Nachricht gleich doppelt: Einmal als reinen Text und ein zweites Mal als HTML-Code. Um derartige Mails in Python zusammenzustellen, benötigen Sie eine `MIMEMultipart`-Instanz vom Typ `alternative`, der Sie dann `MIMEText`-Objekte mit den beiden Varianten der Nachricht hinzufügen. Ein weiterer Vorteil dieser Vorgehensweise besteht darin, dass nun alle Teile der Mail (auch Absender und Empfänger) internationale Sonderzeichen enthalten dürfen.

```
# Beispielprogramm mail3.py
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
import smtplib
msg  = 'Lorem ipsum äöü ...'
html = '<html><body><p>Lorem ipsum<p>äöü ...</body></html>'
subj = 'Noch eine von Python versendete Mail äöü'
frm  = 'Sender mit äöü <bla@bla.com>'
to   = 'Empfänger mit äöü <bla@blabla.com>'
# E-Mail zusammenstellen
mail = MIMEMultipart('alternative')
mail['Subject'] = subj
mail['From']    = frm
mail['To']      = to
mail.attach(MIMEText(msg, 'plain'))
mail.attach(MIMEText(html, 'html'))
# E-Mail versenden
... wie bisher
```

Bild hinzufügen

Um einer E-Mail ein Bild hinzuzufügen, erzeugen Sie die Nachricht wieder als `MIMEMultipart`, aber dieses Mal ohne den `alternative`-Parameter. Sie müssen sich für eine Textform entscheiden, Plain oder HTML. Dafür können Sie mit `attach` nun beliebig viele in `MIMEImage`-Objekten verpackte Bilder hinzufügen:

```
# Beispieldatei mail4.py
from email.mime.image import MIMEImage
... wie bisher

# E-Mail zusammenstellen
mail = MIMEMultipart()
mail['Subject'] = subj
mail['From']    = frm
mail['To']      = to
mail.attach(MIMEText(html, 'html'))

# Datei mit Foto hinzufügen
with open('foto.jpg', 'rb') as f:
    img = MIMEImage(f.read())
mail.attach(img)

# E-Mail versenden
... wie bisher
```

Weitere Mail-Varianten

Die vollständige Dokumentation der diversen Mail-Klassen sowie eine Menge weiterer Beispiele finden Sie hier:

https://docs.python.org/3/library/email.html
https://docs.python.org/3/library/email.examples.html